

---

# Python Tutorial

*Δημοσίευση 3.10.18*

**Guido van Rossum  
and the Python development team**

Ιουλίου 08, 2025

**Python Software Foundation  
Email: [docs@python.org](mailto:docs@python.org)**



<b>1</b>	<b>Ανοίγοντας την όρεξή σας</b>	<b>3</b>
<b>2</b>	<b>Χρησιμοποιώντας τον Interpreter της Python</b>	<b>5</b>
2.1	Κάνοντας invoke τον Interpreter . . . . .	5
2.1.1	Εισαγωγή Ορίσματος . . . . .	6
2.1.2	Interactive Λειτουργία . . . . .	6
2.2	Ο Interpreter και το Περιβάλλον του . . . . .	7
2.2.1	Κωδικοποίηση Πηγαίου Κώδικα . . . . .	7
<b>3</b>	<b>Μία Άτυπη Εισαγωγή στην Python</b>	<b>9</b>
3.1	Χρησιμοποιώντας την Python ως Αριθμομηχανή . . . . .	10
3.1.1	Αριθμοί . . . . .	10
3.1.2	Strings . . . . .	11
3.1.3	Λίστες . . . . .	15
3.2	Πρώτα Βήματα Προς Τον Προγραμματισμό . . . . .	16
<b>4</b>	<b>Περισσότερα εργαλεία Ελέγχου Ροής</b>	<b>19</b>
4.1	Προτάσεις if . . . . .	19
4.2	Προτάσεις for . . . . .	20
4.3	Η συνάρτηση range () . . . . .	20
4.4	break and continue Statements, and else Clauses on Loops . . . . .	21
4.5	Προτάσεις pass . . . . .	22
4.6	Προτάσεις match . . . . .	23
4.7	Καθορισμός Συναρτήσεων . . . . .	25
4.8	Περισσότερο για τον Καθορισμό Συναρτήσεων . . . . .	27
4.8.1	Προεπιλεγμένες Τιμές Ορίσματος . . . . .	27
4.8.2	Ορίσματα Keyword . . . . .	28
4.8.3	Ειδικές παράμετροι . . . . .	29
4.8.4	Λίστες Αυθαίρετων Ορισμάτων . . . . .	32
4.8.5	Unpacking Λίστες Ορισμάτων . . . . .	32
4.8.6	Εκφράσεις Lambda . . . . .	33
4.8.7	Συμβολοσειρές Τεκμηρίωσης . . . . .	33
4.8.8	Annotations Συναρτήσεων . . . . .	34
4.9	Intermezzo: Στυλ Κώδικα . . . . .	34
<b>5</b>	<b>Δομές Δεδομένων</b>	<b>37</b>
5.1	Περισσότερα για τις Λίστες . . . . .	37
5.1.1	Χρήση Λιστών ως Στοίβες (Stacks) . . . . .	38
5.1.2	Χρήση λιστών ως Ουρές (Queues) . . . . .	39
5.1.3	Comprehensions Λίστας . . . . .	39
5.1.4	Comprehensions Ένθετων Λιστών . . . . .	41

5.2	Η δήλωση <code>del</code> . . . . .	41
5.3	Πλειάδες (Tuples) και Ακολουθίες . . . . .	42
5.4	Σύνολα (Sets) . . . . .	43
5.5	Λεξικά (Dictionaries) . . . . .	44
5.6	Τεχνικές Looping . . . . .	45
5.7	Περισσότερα για τις συνθήκες . . . . .	46
5.8	Σύγκριση ακολουθιών και άλλων τύπων . . . . .	47
<b>6</b>	<b>Modules</b>	<b>49</b>
6.1	Περισσότερα για τα Modules . . . . .	50
6.1.1	Εκτέλεση modules ως scripts . . . . .	51
6.1.2	To Search Path του Module . . . . .	52
6.1.3	«Compiled» Python αρχεία . . . . .	52
6.2	Standard Modules . . . . .	53
6.3	Η συνάρτηση <code>dir()</code> . . . . .	53
6.4	Πακέτα . . . . .	55
6.4.1	Εισάγοντας * από ένα Πακέτο . . . . .	56
6.4.2	Intra-package αναφορές . . . . .	57
6.4.3	Πακέτα σε Πολλαπλούς Καταλόγους . . . . .	57
<b>7</b>	<b>Είσοδος και Έξοδος</b>	<b>59</b>
7.1	Ομορφότερη Μορφοποίηση Εξόδου . . . . .	59
7.1.1	Μορφοποιημένα String Literals . . . . .	60
7.1.2	Η μέθοδος <code>String format()</code> . . . . .	61
7.1.3	Χειροκίνητη Μορφοποίηση Συμβολοσειρών . . . . .	62
7.1.4	Παλιά μορφοποίηση συμβολοσειράς . . . . .	63
7.2	Ανάγνωση και Εγγραφή Αρχείων . . . . .	63
7.2.1	Μέθοδοι Αντικειμένων Αρχείων . . . . .	64
7.2.2	Αποθήκευση δομημένων δεδομένων με <code>json</code> . . . . .	66
<b>8</b>	<b>Σφάλματα και Εξαιρέσεις</b>	<b>67</b>
8.1	Syntax Errors (Συντακτικά Σφάλματα) . . . . .	67
8.2	Exceptions (Εξαιρέσεις) . . . . .	67
8.3	Διαχείριση Εξαιρέσεων . . . . .	68
8.4	Raising Εξαιρέσεων . . . . .	70
8.5	Αλυσιδωτές Εξαιρέσεις . . . . .	71
8.6	Εξαιρέσεις που καθορίζονται από το χρήστη . . . . .	72
8.7	Καθορισμός ενεργειών καθαρισμού . . . . .	72
8.8	Προκαθορισμένες ενέργειες καθαρισμού . . . . .	73
<b>9</b>	<b>Κλάσεις</b>	<b>75</b>
9.1	Λίγα λόγια για Ονόματα και Αντικείμενα . . . . .	76
9.2	Εμβέλεια και Πεδία Ονομάτων στην Python . . . . .	76
9.2.1	Παράδειγμα Εμβέλειας και Χώρων Ονομάτων . . . . .	77
9.3	Μια πρώτη ματιά στις Κλάσεις . . . . .	78
9.3.1	Σύνταξη Ορισμού Κλάσης . . . . .	78
9.3.2	Αντικείμενα Κλάσης . . . . .	79
9.3.3	Αντικείμενα Στιγμιότυπων . . . . .	80
9.3.4	Αντικείμενα Μεθόδου . . . . .	80
9.3.5	Μεταβλητές Κλάσης και Στιγμιότυπου . . . . .	81
9.4	Τυχαίες Παρατηρήσεις . . . . .	82
9.5	Κληρονομικότητα . . . . .	83
9.5.1	Πολλαπλή Κληρονομικότητα . . . . .	84
9.6	Ιδιωτικές Μεταβλητές . . . . .	85
9.7	Μικροπράγματα . . . . .	86
9.8	Επαναλήπτες . . . . .	86
9.9	Γεννήτορες (Generators) . . . . .	87
9.10	Εκφράσεις Γεννητόρων . . . . .	88

<b>10 Σύντομη ξενάγηση στην Standard Βιβλιοθήκη</b>	<b>89</b>
10.1 Διεπαφή Λειτουργικού Συστήματος . . . . .	89
10.2 Wildcard Αρχεία . . . . .	90
10.3 Ορίσματα γραμμής εντολών . . . . .	90
10.4 Ανακατεύθυνση εξόδου σφάλματος και τερματισμός προγράμματος . . . . .	90
10.5 Ταίριασμα μοτίβων συμβολοσειρών . . . . .	91
10.6 Μαθηματικά . . . . .	91
10.7 Πρόσβαση στο Διαδίκτυο . . . . .	92
10.8 Ημερομηνίες και ώρες . . . . .	92
10.9 Συμπίεση Δεδομένων . . . . .	93
10.10 Μέτρηση επίδοσης . . . . .	93
10.11 Έλεγχος ποιότητας . . . . .	93
10.12 Batteries Included . . . . .	94
<b>11 Σύντομη περιήγηση στην Πρότυπη Βιβλιοθήκη — Μέρος II</b>	<b>95</b>
11.1 Μορφοποίηση εξόδου . . . . .	95
11.2 Templating . . . . .	96
11.3 Εργασία με δυαδικές διατάξεις εγγραφής δεδομένων . . . . .	97
11.4 Multi-threading . . . . .	98
11.5 Logging . . . . .	98
11.6 Αδύναμες αναφορές . . . . .	99
11.7 Εργαλεία για εργασία με λίστες . . . . .	99
11.8 Decimal Floating Point Arithmetic . . . . .	100
<b>12 Εικονικά Περιβάλλοντα και πακέτα</b>	<b>103</b>
12.1 Εισαγωγή . . . . .	103
12.2 Δημιουργία εικονικών περιβάλλοντων . . . . .	103
12.3 Διαχείριση Πακέτων με το pip . . . . .	104
<b>13 Και τώρα τι;</b>	<b>107</b>
<b>14 Διαδραστική Επεξεργασία Input και Αντικατάσταση Ιστορικού</b>	<b>109</b>
14.1 Συμπλήρωση Tab και Επεξεργασία Ιστορικού . . . . .	109
14.2 Εναλλακτικές λύσεις για τον Διαδραστικό Interpreter . . . . .	109
<b>15 Floating Point Arithmetic: Issues and Limitations</b>	<b>111</b>
15.1 Σφάλμα Αναπαράστασης . . . . .	114
<b>16 Παράρτημα</b>	<b>117</b>
16.1 Διαδραστική Λειτουργία . . . . .	117
16.1.1 Διαχείριση Σφαλμάτων . . . . .	117
16.1.2 Εκτελέσιμα Python Scripts . . . . .	117
16.1.3 Το διαδραστικό αρχείο εκκίνησης . . . . .	118
16.1.4 Τα Modules Προσαρμογής . . . . .	118
<b>A' Γλωσσάρι</b>	<b>121</b>
<b>B' About these documents</b>	<b>137</b>
B'.1 Contributors to the Python Documentation . . . . .	137
<b>Γ' Ιστορία και Άδεια</b>	<b>139</b>
Γ'.1 Η ιστορία του λογισμικού . . . . .	139
Γ'.2 Όροι και προϋποθέσεις για την πρόσβαση ή την χρήση της Python με άλλους τρόπους . . . . .	140
Γ'.2.1 PSF LICENSE AGREEMENT FOR PYTHON 3.10.18 . . . . .	140
Γ'.2.2 ΣΥΜΦΩΝΙΑ ΑΔΕΙΑΣ BEOPEN.COM ΓΙΑ PYTHON 2.0 . . . . .	141
Γ'.2.3 ΣΥΜΦΩΝΙΑ ΑΔΕΙΑΣ CNRI ΓΙΑ PYTHON 1.6.1 . . . . .	142
Γ'.2.4 ΣΥΜΦΩΝΙΑ ΑΔΕΙΑΣ CWI ΓΙΑ PYTHON 0.9.0 ΕΩΣ 1.2 . . . . .	143
Γ'.2.5 ZERO-CLAUSE BSD LICENSE FOR CODE IN THE PYTHON 3.10.18 DOCUMENTATION . . . . .	143
Γ'.3 Άδειες και Ευχαριστίες για Ενσωματωμένο Λογισμικό . . . . .	144

Γ'.3.1	Mersenne Twister . . . . .	144
Γ'.3.2	Sockets . . . . .	145
Γ'.3.3	Ασύγχρονες socket υπηρεσίες . . . . .	145
Γ'.3.4	Διαχείριση Cookie . . . . .	146
Γ'.3.5	Ανίχνευση εκτέλεσης . . . . .	146
Γ'.3.6	Συναρτήσεις UUencode και UUdecode . . . . .	147
Γ'.3.7	Κλήσεις Απομακρυσμένης Διαδικασίας XML . . . . .	147
Γ'.3.8	test_epoll . . . . .	148
Γ'.3.9	Επιλογή kqueue . . . . .	148
Γ'.3.10	SipHash24 . . . . .	149
Γ'.3.11	strtod και dtoa . . . . .	149
Γ'.3.12	OpenSSL . . . . .	150
Γ'.3.13	expat . . . . .	152
Γ'.3.14	libffi . . . . .	152
Γ'.3.15	zlib . . . . .	153
Γ'.3.16	cfuhash . . . . .	153
Γ'.3.17	libmpdec . . . . .	154
Γ'.3.18	W3C C14N σουίτα δοκιμής . . . . .	154
Γ'.3.19	Audioop . . . . .	155
<b>Δ' Copyright</b>		<b>157</b>
<b>Ευρετήριο</b>		<b>159</b>

Η Python είναι μια εύκολη στην εκμάθηση, ισχυρή γλώσσα προγραμματισμού. Έχει αποτελεσματικές δομές δεδομένων υψηλού επιπέδου και μια απλή αλλά αποτελεσματική προσέγγιση στον αντικειμενοστραφή προγραμματισμό. Το κομψό συντακτικό και η δυναμική τυποποίηση της Python, σε συνδυασμό με τη διερμηνευμένη φύση της, την καθιστούν ιδανική γλώσσα για scripting και ταχεία ανάπτυξη εφαρμογών σε πολλούς τομείς στις περισσότερες πλατφόρμες.

Ο διερμηνέας της Python και η εκτεταμένη τυπική (standard) βιβλιοθήκη διατίθενται ελεύθερα σε πηγαία ή δυαδική μορφή για όλες τις κύριες πλατφόρμες από την ιστοσελίδα της Python, <https://www.python.org/> και μπορούν να διανεμηθούν ελεύθερα. Ο ίδιος ιστότοπος περιέχει επίσης διανομές και δείκτες σε πολλά δωρεάν modules τρίτων, προγράμματα και εργαλεία Python, καθώς και πρόσθετη τεκμηρίωση.

Ο διερμηνέας της Python επεκτείνεται εύκολα με νέες συναρτήσεις και τύπους δεδομένων που υλοποιούνται σε C ή C++ (ή άλλες γλώσσες που μπορούν να κληθούν από τη C). Η Python είναι επίσης κατάλληλη ως γλώσσα επέκτασης για προσαρμόσιμες εφαρμογές.

Αυτό το tutorial εισάγει τον αναγνώστη ανεπίσημα στις βασικές έννοιες και δυνατότητες της γλώσσας και του συστήματος Python. Βοηθάει να έχετε πρόχειρο έναν διερμηνέα Python για πρακτική εμπειρία, αλλά όλα τα παραδείγματα είναι αυτοτελή, οπότε το tutorial μπορεί να διαβαστεί και εκτός σύνδεσης.

Για μια περιγραφή των τυποποιημένων αντικειμένων και ενοτήτων, δείτε library-index. Το reference-index δίνει έναν πιο επίσημο ορισμό της γλώσσας. Για να γράψετε επεκτάσεις σε C ή C++, διαβάστε το extending-index και το c-api-index. Υπάρχουν επίσης αρκετά βιβλία που καλύπτουν την Python σε βάθος.

Αυτό το tutorial δεν προσπαθεί να είναι περιεκτικό και να καλύψει κάθε χαρακτηριστικό ή ακόμη και κάθε συχνά χρησιμοποιούμενο χαρακτηριστικό. Αντίθετα, θα σας παρουσιάσει πολλά από τα πιο αξιοσημείωτα χαρακτηριστικά της Python και θα σας δώσει μια καλή ιδέα για τη γεύση και το ύφος της γλώσσας. Αφού το διαβάσετε, θα είστε σε θέση να διαβάζετε και να γράφετε modules και προγράμματα Python, και θα είστε έτοιμοι να μάθετε περισσότερα για τα διάφορα modules βιβλιοθηκών Python που περιγράφονται στο library-index.

Αξίζει επίσης να διαβάσετε το *Γλωσσάρι*.





---

Ανοίγοντας την όρεξή σας

---

Εάν κάνετε πολλή δουλειά σε υπολογιστές, τελικά θα διαπιστώσετε ότι υπάρχει κάποια εργασία που θα θέλατε να αυτοματοποιήσετε. Για παράδειγμα, μπορεί να θέλετε να πραγματοποιήσετε αναζήτηση και αντικατάσταση σε μεγάλο αριθμό αρχείων κειμένου ή να μετονομάσετε και να αναδιατάξετε μια δέσμη αρχείων φωτογραφιών με περίπλοκο τρόπο. Ίσως θα θέλατε να γράψετε μια μικρή προσαρμοσμένη βάση δεδομένων ή μια εξειδικευμένη εφαρμογή GUI ή ένα απλό παιχνίδι.

Εάν είστε επαγγελματίας προγραμματιστής λογισμικού, μπορεί να χρειαστεί να εργαστείτε με πολλές βιβλιοθήκες C/C++/Java, αλλά ο συνηθισμένος κύκλος εγγραφής/compile/δοκιμής/re-compile είναι πολύ αργός. Ίσως γράφετε ένα test suite για μια τέτοια βιβλιοθήκη και βρίσκετε τη δημιουργία κώδικα testing μια κουραστική διαδικασία. Ή ίσως έχετε γράψει ένα πρόγραμμα που θα μπορούσε να χρησιμοποιεί μια γλώσσα επέκτασης, και δεν θέλετε να σχεδιάσετε και να εφαρμόσετε μια εντελώς νέα γλώσσα για την εφαρμογή σας.

Η Python είναι απλώς η γλώσσα για εσάς.

Θα μπορούσατε να γράψετε ένα Unix shell script ή Windows batch αρχεία για ορισμένες από αυτές τις εργασίες, αλλά τα shell scripts είναι τα καλύτερα για τη μετακίνηση αρχείων και την αλλαγή δεδομένων κειμένου, δεν είναι κατάλληλα για εφαρμογές ή παιχνίδια GUI. Θα μπορούσατε να γράψετε ένα C/C++/Java πρόγραμμα, αλλά μπορεί να χρειαστεί πολύς χρόνος ανάπτυξης για να δημιουργηθεί μια πρώτη έκδοση του προγράμματος. Η Python είναι πιο απλή στη χρήση, διαθέσιμη σε λειτουργικά συστήματα Windows, macOS και Unix και θα σας βοηθήσει να ολοκληρώσετε τη δουλειά πιο γρήγορα.

Η Python είναι απλή στη χρήση, αλλά είναι μια πραγματική γλώσσα προγραμματισμού, που προσφέρει πολύ περισσότερη δομή και υποστήριξη για μεγάλα προγράμματα από ό,τι μπορούν να προσφέρουν τα shell scripts ή τα batch αρχεία. Από την άλλη πλευρά, η Python προσφέρει επίσης πολύ περισσότερο έλεγχο ασφαλιμάτων από τη C και, όντας μια γλώσσα πολύ υψηλού επιπέδου, έχει ενσωματωμένους τύπους δεδομένων υψηλού επιπέδου, όπως ευέλικτους πίνακες, και λεξικά. Λόγω των πιο γενικών τύπων δεδομένων της, η Python μπορεί να εφαρμοστεί σε έναν πολύ μεγαλύτερο τομέα προβλημάτων από την Awk ή ακόμα και την Perl, ωστόσο πολλά πράγματα είναι τουλάχιστον τόσο εύκολα στην Python όσο σε αυτές τις γλώσσες.

Η Python σας επιτρέπει να χωρίσετε το πρόγραμμά σας σε modules που μπορούν να επαναχρησιμοποιηθούν σε άλλα προγράμματα Python. Έρχεται με μια μεγάλη συλλογή standard modules που μπορείτε να χρησιμοποιήσετε ως βάση των προγραμμάτων σας — ή ως παράδειγμα για να ξεκινήσετε να μαθαίνετε να προγραμματίζετε σε Python. Ορισμένα από αυτά τα modules παρέχουν πράγματα όπως I/O αρχείων, κλήσεις συστήματος, sockets, ακόμη και interfaces σε toolkits γραφικών διεπαφής χρήστη, όπως το Tk.

Η Python είναι μια interpreted γλώσσα, η οποία μπορεί να σας εξοικονομήσει σημαντικό χρόνο κατά την ανάπτυξη του προγράμματος, επειδή δεν απαιτείται compilation και linking. Ο interpreter μπορεί να χρησιμοποιηθεί διαδραστικά, γεγονός που καθιστά εύκολο τον πειραματισμό με χαρακτηριστικά της γλώσσας, τη

σύνταξη προγραμμάτων ή τον έλεγχο συναρτήσεων κατά την ανάπτυξη προγράμματος από κάτω προς τα πάνω. Είναι επίσης μια εύχρηστη αριθμομηχανή.

Η Python επιτρέπει στα προγράμματα να γράφονται συμπαγή και ευανάγνωστα. Τα προγράμματα που είναι γραμμένα σε Python είναι συνήθως πολύ μικρότερα από τα αντίστοιχα προγράμματα C, C++ ή Java, για διάφορους λόγους:

- οι τύποι δεδομένων υψηλού επιπέδου σάς επιτρέπουν να εκφράσετε πολύπλοκες λειτουργίες σε μία μόνο δήλωση•
- η ομαδοποίηση δηλώσεων γίνεται με εσοχή αντί για αγκύλες αρχής και λήξης•
- δεν απαιτούνται δηλώσεις μεταβλητών ή ορισμάτων.

Η Python είναι *επεκτάσιμη*: αν ξέρετε πώς να προγραμματίζετε σε C είναι εύκολο να προσθέσετε μια νέα ενσωματωμένη συνάρτηση ή module στον interpreter, είτε για να εκτελέσετε κρίσιμες λειτουργίες με μέγιστη ταχύτητα, είτε για να συνδέσετε προγράμματα Python με βιβλιοθήκες που μπορεί να είναι διαθέσιμες μόνο σε δυαδική μορφή (όπως μια βιβλιοθήκη γραφικών για συγκεκριμένο προμηθευτή). Μόλις κολλήσετε πραγματικά, μπορείτε να συνδέσετε τον Python interpreter σε μια εφαρμογή γραμμένη σε C και να τη χρησιμοποιήσετε ως επέκταση ή γλώσσα εντολών για αυτήν την εφαρμογή.

Παρεμπιπτόντως, η γλώσσα πήρε το όνομά της από την εκπομπή του BBC «Monty Python's Flying Circus» και δεν έχει καμία σχέση με ερπετά. Η αναφορά σε σκετς Monty Python στην τεκμηρίωση όχι μόνο επιτρέπεται, αλλά και ενθαρρύνεται!

Τώρα που είστε όλοι ενθουσιασμένοι με την Python, θα θέλετε να την εξετάσετε με περισσότερες λεπτομέρειες. Επειδή ο καλύτερος τρόπος για να μάθετε μια γλώσσα είναι να τη χρησιμοποιήσετε, ο οδηγός εκμάθησης σας προσκαλεί να παίξετε με τον Python interpreter καθώς διαβάσετε.

Στο επόμενο κεφάλαιο, εξηγείται η μηχανική της χρήσης του interpreter. Αυτή είναι μάλλον πεζή πληροφορία, αλλά απαραίτητη για τη δοκιμή των παραδειγμάτων που παρουσιάζονται αργότερα.

Ο υπόλοιπος οδηγός εκμάθησης εισάγει διάφορα χαρακτηριστικά της γλώσσας και του συστήματος Python μέσω παραδειγμάτων, ξεκινώντας με απλές εκφράσεις, δηλώσεις και τύπους δεδομένων, μέσω συναρτήσεων και modules και τέλος αγγίζοντας προηγμένες έννοιες όπως εξαιρέσεις και κλάσεις που καθορίζονται από τον χρήστη.

---

## Χρησιμοποιώντας τον Interpreter της Python

---

### 2.1 Κάνοντας invoke τον Interpreter

The Python interpreter is usually installed as `/usr/local/bin/python3.10` on those machines where it is available; putting `/usr/local/bin` in your Unix shell's search path makes it possible to start it by typing the command:

```
python3.10
```

στο shell.<sup>1</sup> Εφόσον η επιλογή του καταλόγου όπου βρίσκεται ο interpreter είναι μια επιλογή εγκατάστασης, και άλλα μέρη είναι πιθανά• επικοινωνήστε με τον τοπικό σας guru της Python ή τον διαχειριστή συστήματος. (π.χ., `/usr/local/python` είναι μια δημοφιλής εναλλακτική τοποθεσία.)

On Windows machines where you have installed Python from the Microsoft Store, the `python3.10` command will be available. If you have the `py.exe` launcher installed, you can use the `py` command. See `setting-envvars` for other ways to launch Python.

Η πληκτρολόγηση ενός χαρακτήρα τέλους αρχείου (`Control-D` στο Unix, `Control-Z` στα Windows) στην κύρια γραμμή εντολών αναγκάζει τον interpreter να βγει με μηδενική κατάσταση εξόδου. Εάν αυτό δεν λειτουργεί, μπορείτε να βγείτε από τον διερμηνέα πληκτρολογώντας την ακόλουθη εντολή: `quit()`.

Οι δυνατότητες επεξεργασίας γραμμής του interpreter περιλαμβάνουν interactive επεξεργασία, αντικατάσταση ιστορικού και συμπλήρωση κώδικα σε συστήματα που υποστηρίζουν την [GNU Readline](#) βιβλιοθήκη. Ίσως ο πιο γρήγορος έλεγχος για να δείτε αν υποστηρίζεται η επεξεργασία της γραμμής εντολών είναι να πληκτρολογήσετε `Control-P` στην πρώτη γραμμή εντολών Python που λαμβάνετε. Εάν ηχήσει, έχετε επεξεργασία γραμμής εντολών• βλ. το παράρτημα [Διαδραστική Επεξεργασία Input και Αντικατάσταση Ιστορικού](#) για εισαγωγή στα πλήκτρα. Εάν δεν φαίνεται να συμβαίνει τίποτα ή αν ηχήσει το `^P`, η επεξεργασία της γραμμής εντολών δεν είναι διαθέσιμη• θα μπορείτε να χρησιμοποιήσετε μόνο το `backspace` για να αφαιρέσετε χαρακτήρες από την τρέχουσα γραμμή.

Ο interpreter λειτουργεί κάπως όπως το Unix shell: όταν καλείται μια standard είσοδο συνδεδεμένη σε μια συσκευή `tty`, διαβάσει και εκτελεί εντολές αλληλεπιδραστικά• όταν καλείται με ένα όρισμα ονόματος αρχείου ή με ένα αρχείο ως standard είσοδο, διαβάσει και εκτελεί ένα *script* από αυτό το αρχείο.

Ένας δεύτερος τρόπος εκκίνησης του interpreter είναι `python -c command [arg] . . .`, η οποία εκτελεί τις εντολές στο *command*, ανάλογο με την επιλογή `-c` του shell. Εφόσον οι δηλώσεις Python συχνά περιέχουν

---

<sup>1</sup> Στο Unix, ο Python 3.x interpreter από προεπιλογή δεν είναι εγκατεστημένος με το εκτελέσιμο αρχείο που ονομάζεται `python`, έτσι ώστε να μην έρχεται σε conflict με ένα εκτελέσιμο Python 2.x που εγκαθίσταται ταυτόχρονα.

κενά διαστήματα ή άλλους χαρακτήρες που είναι ειδικοί για το shell, συνήθως συνίσταται να αναφέρετε το *command* στο σύνολο του.

Ορισμένα Python modules είναι επίσης χρήσιμα ως scripts. Αυτά μπορούν να κληθούν χρησιμοποιώντας το `python -m module [arg] . . .`, το οποίο εκτελεί το πηγαίο αρχείο για το *module* σαν να είχατε γράψει το πλήρες όνομα του, στη γραμμή εντολών.

Όταν χρησιμοποιείται ένα script αρχείο, μερικές φορές είναι χρήσιμο να μπορείτε να εκτελέσετε το script και να εισέλθετε σε interactive λειτουργία μετά. Αυτό μπορεί να γίνει περνώντας το `-i` πριν από το script.

Όλες οι επιλογές της γραμμής εντολών περιγράφονται στο `using-on-general`.

## 2.1.1 Εισαγωγή Ορίσματος

Όταν είναι γνωστά στον interpreter, το όνομα του script και τα πρόσθετα ορίσματα στην συνέχεια μετατρέπονται σε λίστα συμβολοσειρών και εκχωρούνται στην μεταβλητή `argv` στο module `sys`. Μπορείτε να αποκτήσετε πρόσβαση σε αυτήν την λίστα εκτελώντας το `import sys`. Το μήκος της λίστας είναι τουλάχιστον ένα· όταν δεν δίνονται script και δεν δίνονται ορίσματα, το `sys.argv[0]` είναι μια κενή συμβολοσειρά. Όταν το όνομα του script δίνεται ως `'-'` (που σημαίνει standard είσοδος), το `sys.argv[0]` ορίζεται ως `'-'`. Όταν χρησιμοποιείται η εντολή `-c`, το `sys.argv[0]` ορίζεται σε `''-c'`. Όταν χρησιμοποιείται η εντολή `-m`, το `sys.argv[0]` ορίζεται ως το πλήρες όνομα του module που βρίσκεται. Οι επιλογές που βρέθηκαν μετά την εντολή `-c` ή το `-m module` δεν καταναλώνονται από τον επεξεργαστή επιλογών του interpreter της Python αλλά αφήνονται στο `sys.argv` για την εντολή ή το module να το διαχειριστεί.

## 2.1.2 Interactive Λειτουργία

Όταν διαβάζονται εντολές από ένα tty, ο interpreter λέγεται ότι βρίσκεται σε *interactive λειτουργία*. Σε αυτή τη λειτουργία υπενθυμίζει την επόμενη εντολή με την κύρια εντολή, συνήθως τρία σύμβολα μεγαλύτερο (`>>>`)· για τις γραμμές συνέχειας, υπενθυμίζει με τη *δευτερεύουσα εντολή*, από προεπιλογή τρεις τελείες (`. . .`). Ο interpreter εκτυπώνει ένα μήνυμα καλωσορίσματος που αναφέρει τον αριθμό έκδοσης και μια ειδοποίηση πνευματικών δικαιωμάτων πριν εκτυπώσει το πρώτο μήνυμα:

```
$ python3.10
Python 3.10 (default, June 4 2019, 09:25:04)
[GCC 4.8.2] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

Απαιτούνται γραμμές συνέχειας όταν εισάγεται μια κατασκευή πολλών γραμμών. Για παράδειγμα ρίξτε μια ματιά σε αυτήν την δήλωση `if`:

```
>>> the_world_is_flat = True
>>> if the_world_is_flat:
...     print("Be careful not to fall off!")
...
Be careful not to fall off!
```

Για περισσότερα σχετικά με την interactive λειτουργία, βλ. [Διαδραστική Λειτουργία](#).

## 2.2 Ο Interpreter και το Περιβάλλον του

### 2.2.1 Κωδικοποίηση Πηγαίου Κώδικα

Από προεπιλογή, τα πηγαία αρχεία Python αντιμετωπίζονται ως κωδικοποιημένα στο UTF-8. Σε αυτήν την κωδικοποίηση, οι χαρακτήρες των περισσότερων γλωσσών στον κόσμο μπορούν να χρησιμοποιηθούν ταυτόχρονα σε literal συμβολοσειρές, αναγνωριστικά και σχόλια — αν και η τυπική βιβλιοθήκη χρησιμοποιεί μόνο χαρακτήρες ASCII για αναγνωριστικά, μια σύμβαση που πρέπει να ακολουθεί οποιοδήποτε φορητός κώδικας. Για να εμφανίσει σωστά όλους αυτούς τους χαρακτήρες, το λογισμικό επεξεργασίας σας πρέπει να αναγνωρίσει ότι το αρχείο είναι UTF-8 και πρέπει να χρησιμοποιεί μια γραμματοσειρά που να υποστηρίζει όλους τους χαρακτήρες του αρχείου.

Για να δηλώσετε μια κωδικοποίηση διαφορετική από την προεπιλεγμένη, θα πρέπει να προστεθεί μια ειδική γραμμή σχολίων ως *πρώτη* γραμμή του αρχείου. Η σύνταξη είναι η εξής:

```
# -*- coding: encoding -*-
```

όπου το *encoding* είναι ένα από τα έγκυρα `codecs` που υποστηρίζονται από την Python.

Για παράδειγμα, για να δηλώσετε ότι πρόκειται να χρησιμοποιηθεί η κωδικοποίηση Windows-1252, η πρώτη γραμμή του αρχείου του πηγαίου κώδικα θα πρέπει να είναι:

```
# -*- coding: cp1252 -*-
```

Μια εξαίρεση στον κανόνα *first line* είναι όταν ο πηγαίος κώδικας ξεκινά με μια γραμμή *UNIX «shebang»*. Σε αυτήν την περίπτωση, η δήλωση κωδικοποίησης θα πρέπει να προστεθεί ως δεύτερη γραμμή του αρχείου. Για παράδειγμα:

```
#!/usr/bin/env python3
# -*- coding: cp1252 -*-
```

### Υποσημειώσεις



---

## Μία Άτυπη Εισαγωγή στην Python

---

Στα ακόλουθα παραδείγματα, η είσοδος και η έξοδος διακρίνονται από την παρουσία ή την απουσία prompts (`>>>` και `...`): για να επαναλάβετε το παράδειγμα, πρέπει να πληκτρολογήσετε τα πάντα μετά το prompt, όταν αυτό υπάρχει• οι γραμμές που δεν αρχίζουν με prompt είναι έξοδος από τον διερμηνέα. Σημείωση ότι ένα δευτερεύον prompt μόνο του σε μια γραμμή σε ένα παράδειγμα σημαίνει ότι πρέπει να πληκτρολογήσετε μια κενή γραμμή• αυτό χρησιμοποιείται για να τερματίσετε μια εντολή πολλών γραμμών.

Πολλά από τα παραδείγματα σε αυτόν τον οδηγό, ακόμη και αυτά που εισάγονται στο διαδραστικό prompt, περιέχουν σχόλια. Τα σχόλια στην Python ξεκινούν με τον χαρακτήρα hashtag, #, και εκτείνονται μέχρι το τέλος της γραμμής. Ένα σχόλιο μπορεί να εμφανιστεί στην αρχή μιας σειράς ή μετά από κενά διαστήματα ή κώδικα, αλλά όχι μέσα σε μία συμβολοσειρά. Ένας χαρακτήρας hashtag μέσα σε μία συμβολοσειρά είναι απλώς ένας χαρακτήρας hashtag. Δεδομένου ότι τα σχόλια αποσκοπούν στην αποσαφήνιση του κώδικα και δεν ερμηνεύονται από την Python, μπορούν να παραλείπονται κατά την πληκτρολόγηση παραδειγμάτων.

Μερικά παραδείγματα:

```
# this is the first comment
spam = 1  # and this is the second comment
          # ... and now a third!
text = "# This is not a comment because it's inside quotes."
```

## 3.1 Χρησιμοποιώντας την Python ως Αριθμομηχανή

Ας δοκιμάσουμε μερικές απλές εντολές της Python. Ξεκινήστε τον διερμηνέα και περιμένετε το πρώτο prompt, `>>>`. (Δεν θα πάρει πολύ χρόνο.)

### 3.1.1 Αριθμοί

The interpreter acts as a simple calculator: you can type an expression at it and it will write the value. Expression syntax is straightforward: the operators `+`, `-`, `*` and `/` work just like in most other languages (for example, Pascal or C); parentheses `( )` can be used for grouping. For example:

```
>>> 2 + 2
4
>>> 50 - 5*6
20
>>> (50 - 5*6) / 4
5.0
>>> 8 / 5 # division always returns a floating point number
1.6
```

Οι ακέραιοι αριθμοί (π.χ. 2, 4, 20) έχουν τον τύπο `int`, οι αριθμοί με δεκαδικά ψηφία (π.χ. 5.0, 1.6) έχουν τον τύπο `float`. Θα δούμε περισσότερα για τους αριθμητικούς τύπους αργότερα σε αυτόν τον οδηγό.

Η διαίρεση (`/`) πάντα επιστρέφει ένα `float`. Για να κάνετε *floor division* (ακέραια διαίρεση) και να πάρετε ένα ακέραιο αποτέλεσμα, μπορείτε να χρησιμοποιήσετε τον τελεστή `//`.<sup>1</sup> για να υπολογίσετε το το υπόλοιπο μίας διαίρεσης, χρησιμοποιήστε τον τελεστή `%`:

```
>>> 17 / 3 # classic division returns a float
5.666666666666667
>>>
>>> 17 // 3 # floor division discards the fractional part
5
>>> 17 % 3 # the % operator returns the remainder of the division
2
>>> 5 * 3 + 2 # floored quotient * divisor + remainder
17
```

Στην Python, είναι εφικτό να χρησιμοποιήσετε τον τελεστή `**` για να υπολογίσετε δυνάμεις<sup>1</sup>:

```
>>> 5 ** 2 # 5 squared
25
>>> 2 ** 7 # 2 to the power of 7
128
```

Το σύμβολο της ισότητας (`=`) χρησιμοποιείται για την ανάθεση μιας τιμής σε μια μεταβλητή. Στη συνέχεια, δεν εμφανίζεται αποτέλεσμα πριν από το επόμενο διαδραστικό prompt:

```
>>> width = 20
>>> height = 5 * 9
>>> width * height
900
```

Αν μία μεταβλητή δεν έχει «οριστεί» (δεν της έχει αποδοθεί κάποια τιμή), η προσπάθεια χρήσης της θα σας δώσει ένα σφάλμα:

```
>>> n # try to access an undefined variable
Traceback (most recent call last):
```

(συνέχεια στην επόμενη σελίδα)

<sup>1</sup> Δεδομένου ότι το `**` έχει μεγαλύτερη προτεραιότητα από το `-`, το `-3**2` θα ερμηνευτεί ως `-(3**2)` και έτσι θα προκύψει `-9`. Για να το αποφύγετε αυτό και να πάρετε 9, μπορείτε να χρησιμοποιήσετε `(-3)**2`.



(συνεχίζεται από την προηγούμενη σελίδα)

```
File "<stdin>", line 1, in <module>
NameError: name 'n' is not defined
```

Υπάρχει πλήρης υποστήριξη για δεκαδικά ψηφία• τελεστές με τελεστέους μικτού τύπου μετατρέπουν τον ακέραιο τελεστέο σε δεκαδικό:

```
>>> 4 * 3.75 - 1
14.0
```

Στη διαδραστική λειτουργία, η τελευταία εκτυπωμένη έκφραση εκχωρείται στη μεταβλητή `_`. Αυτό σημαίνει ότι όταν χρησιμοποιείτε την Python ως αριθμομηχανή γραφείου, είναι κάπως πιο εύκολο να συνεχίσετε προηγούμενους υπολογισμούς, για παράδειγμα:

```
>>> tax = 12.5 / 100
>>> price = 100.50
>>> price * tax
12.5625
>>> price + _
113.0625
>>> round(_, 2)
113.06
```

Αυτή η μεταβλητή θα πρέπει να αντιμετωπίζεται από τον χρήστη ως μόνο για ανάγνωση. Μην της αναθέτετε ρητά μια τιμή — θα δημιουργούσατε μια ανεξάρτητη τοπική μεταβλητή με το ίδιο όνομα αποκρύπτοντας την ενσωματωμένη μεταβλητή με τη μαγική της συμπεριφορά.

Εκτός από `int` και `float`, η Python υποστηρίζει και άλλους τύπους αριθμών, όπως `Decimal` και `Fraction`. Η Python έχει επίσης ενσωματωμένη υποστήριξη για `complex numbers` (μιγαδικούς αριθμούς), και χρησιμοποιεί την κατάληξη `j` ή `J` για να δηλώσει το φανταστικό μέρος (π.χ. `3+5j`).

### 3.1.2 Strings

Besides numbers, Python can also manipulate strings, which can be expressed in several ways. They can be enclosed in single quotes (`'...'`) or double quotes (`"..."`) with the same result<sup>2</sup>. `\` can be used to escape quotes:

```
>>> 'spam eggs' # single quotes
'spam eggs'
>>> 'doesn\'t' # use \' to escape the single quote...
"doesn't"
>>> "doesn't" # ...or use double quotes instead
"doesn't"
>>> '"Yes," they said.'
'"Yes," they said.'
>>> "\"Yes,\" they said."
'"Yes," they said.'
>>> '"Isn\'t," they said.'
'"Isn\'t," they said.'
```

In the interactive interpreter, the output string is enclosed in quotes and special characters are escaped with backslashes. While this might sometimes look different from the input (the enclosing quotes could change), the two strings are equivalent. The string is enclosed in double quotes if the string contains a single quote and no double quotes, otherwise it is enclosed in single quotes. The `print()` function produces a more readable output, by omitting the enclosing quotes and by printing escaped and special characters:

<sup>2</sup> Σε αντίθεση με άλλες γλώσσες, οι ειδικοί χαρακτήρες όπως το `\n` έχουν την ίδια σημασία τόσο με απλά (`'...'`) όσο και με διπλά (`"..."`) εισαγωγικά. Η μόνη διαφορά μεταξύ των δύο είναι ότι μέσα σε απλά εισαγωγικά δεν χρειάζεται να αποφύγετε το `"` (αλλά πρέπει να αποφύγετε το `\`) και το αντίστροφο.

```
>>> "Isn't," they said.  
"Isn't," they said.  
>>> print("Isn't," they said.)  
"Isn't," they said.  
>>> s = 'First line.\nSecond line.' # \n means newline  
>>> s # without print(), \n is included in the output  
'First line.\nSecond line.'  
>>> print(s) # with print(), \n produces a new line  
First line.  
Second line.
```

Εάν δεν θέλετε οι χαρακτήρες που προηγούνται από μία \ να ερμηνεύονται ως ειδικοί χαρακτήρες, μπορείτε να χρησιμοποιήσετε ακατέργαστες συμβολοσειρές προσθέτοντας ένα r πριν από το πρώτο εισαγωγικό:

```
>>> print('C:\some\name') # here \n means newline!  
C:\some  
ame  
>>> print(r'C:\some\name') # note the r before the quote  
C:\some\name
```

Υπάρχει μια δυσδιάκριτη πτυχή στις ακατέργαστες συμβολοσειρές: μια ακατέργαστη συμβολοσειρά δεν μπορεί να τελειώνει σε περιττό αριθμό χαρακτήρων \” δείτε το λήμμα των Συχνών Ερωτήσεων για περισσότερες πληροφορίες και λύσεις.

String literals can span multiple lines. One way is using triple-quotes: `"""..."""` or `'''...'''`. End of lines are automatically included in the string, but it's possible to prevent this by adding a \ at the end of the line. The following example:

```
print("""\n  
Usage: thingy [OPTIONS]  
    -h                Display this usage message  
    -H hostname       Hostname to connect to  
""")
```

produces the following output (note that the initial newline is not included):

```
Usage: thingy [OPTIONS]  
    -h                Display this usage message  
    -H hostname       Hostname to connect to
```

Οι συμβολοσειρές μπορούν να συνδεθούν (κολληθούν η μία στην άλλη) με τον τελεστή +, και να επαναληφθούν με τον τελεστή \*:

```
>>> # 3 times 'un', followed by 'ium'  
>>> 3 * 'un' + 'ium'  
'unununium'
```

Δύο ή παραπάνω συμβολοσειρές (που περικλείονται σε εισαγωγικά) η μία δίπλα στην άλλη συνδέονται αυτόματα.

```
>>> 'Py' 'thon'  
'Python'
```

Αυτή η λειτουργία είναι ιδιαίτερα χρήσιμη όταν θέλετε να σπάσετε μεγάλες συμβολοσειρές:

```
>>> text = ('Put several strings within parentheses '  
...        'to have them joined together.')  
>>> text  
'Put several strings within parentheses to have them joined together.'
```

Ωστόσο, αυτό λειτουργεί μόνο με κυριολεκτικές συμβολοσειρές, όχι με μεταβλητές ή εκφράσεις:

```
>>> prefix = 'Py'
>>> prefix 'thon' # can't concatenate a variable and a string literal
File "<stdin>", line 1
    prefix 'thon'
        ^^^^^
SyntaxError: invalid syntax
>>> ('un' * 3) 'ium'
File "<stdin>", line 1
    ('un' * 3) 'ium'
        ^^^^^
SyntaxError: invalid syntax
```

Αν θέλετε να συνδέσετε μεταβλητές ή μία μεταβλητή και μία κυριολεκτική συμβολοσειρά, χρησιμοποιήστε το +:

```
>>> prefix + 'thon'
'Python'
```

Οι συμβολοσειρές μπορούν να είναι *προσπελάσιμες* (μέσω ευρετηρίου), με τον πρώτο χαρακτήρα να έχει τον δείκτη 0. Δεν υπάρχει ξεχωριστός τύπος για χαρακτήρες· ένας χαρακτήρας είναι απλώς μία συμβολοσειρά με μέγεθος ένα:

```
>>> word = 'Python'
>>> word[0] # character in position 0
'P'
>>> word[5] # character in position 5
'n'
```

Οι δείκτες μπορούν να είναι και αρνητικοί αριθμοί, για να ξεκινήσετε την αρίθμηση από τα δεξιά:

```
>>> word[-1] # last character
'n'
>>> word[-2] # second-last character
'o'
>>> word[-6]
'P'
```

Σημειώστε ότι αφού το -0 είναι το ίδιο με το 0, οι αρνητικοί δείκτες ξεκινούν από το -1.

In addition to indexing, *slicing* is also supported. While indexing is used to obtain individual characters, *slicing* allows you to obtain substring:

```
>>> word[0:2] # characters from position 0 (included) to 2 (excluded)
'Py'
>>> word[2:5] # characters from position 2 (included) to 5 (excluded)
'tho'
```

Οι δείκτες ενός τεμαχισμού έχουν χρήσιμες προεπιλογές· η παράλειψη του πρώτου δείκτη έχει ως προεπιλογή το μηδέν, η παράλειψη του δεύτερου δείκτη έχει ως προεπιλογή το μέγεθος της συμβολοσειράς που τεμαχίζεται.

```
>>> word[:2] # character from the beginning to position 2 (excluded)
'Py'
>>> word[4:] # characters from position 4 (included) to the end
'on'
>>> word[-2:] # characters from the second-last (included) to the end
'on'
```

Προσέξτε πώς η αρχή περιλαμβάνεται πάντα, ενώ το τέλος πάντα εξαιρείται. Αυτό εξασφαλίζει ότι το `s[:i]` + `s[i:]` είναι πάντα ίσο με `s`:

```
>>> word[:2] + word[2:]
'Python'
>>> word[:4] + word[4:]
'Python'
```

Ένας τρόπος να θυμάστε πώς λειτουργούν οι τεμαχισμοί είναι να σκεφτείτε ότι οι δείκτες δείχνουν *μεταξύ* χαρακτήρων, με το αριστερό άκρο του πρώτου χαρακτήρα να αριθμείται με 0. Τότε το δεξιό άκρο του τελευταίου χαρακτήρα μιας συμβολοσειράς  $n$  χαρακτήρων έχει δείκτη  $n$ , για παράδειγμα:

```
+---+---+---+---+---+---+
| P | y | t | h | o | n |
+---+---+---+---+---+---+
0   1   2   3   4   5   6
-6  -5  -4  -3  -2  -1
```

Η πρώτη σειρά αριθμών δίνει τη θέση των δεικτών 0...6 στην συμβολοσειρά• η δεύτερη σειρά δίνει τους αντίστοιχους αρνητικούς δείκτες. Ο τεμαχισμός από  $i$  έως  $j$  αποτελείται από όλους τους χαρακτήρες μεταξύ των άκρων με ετικέτες  $i$  και  $j$ , αντίστοιχα.

Για μη αρνητικούς δείκτες, το μήκος ενός τεμαχισμού είναι η διαφορά των δεικτών, εάν και οι δύο είναι εντός των ορίων. Για παράδειγμα, το μήκος του `word[1:3]` είναι 2.

Η απόπειρα χρήσης ενός πολύ μεγάλου δείκτη θα οδηγήσει σε σφάλμα:

```
>>> word[42] # the word only has 6 characters
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
IndexError: string index out of range
```

Ωστόσο, οι δείκτες εκτός εύρους αντιμετωπίζονται χωρίς σφάλμα όταν χρησιμοποιούνται για τεμαχισμούς:

```
>>> word[4:42]
'on'
>>> word[42:]
''
```

Οι συμβολοσειρές της Python δεν μπορούν να αλλάξουν — είναι *immutable*. Επομένως, η ανάθεση σε μια συγκεκριμένη θέση στη συμβολοσειρά οδηγεί σε σφάλμα:

```
>>> word[0] = 'J'
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'str' object does not support item assignment
>>> word[2:] = 'py'
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'str' object does not support item assignment
```

Εάν χρειάζεστε μια διαφορετική συμβολοσειρά, θα πρέπει να δημιουργήσετε μια νέα:

```
>>> 'J' + word[1:]
'Jython'
>>> word[:2] + 'py'
'Pypy'
```

Η ενσωματωμένη συνάρτηση `len()` επιστρέφει το μήκος μιας συμβολοσειράς:

```
>>> s = 'supercalifragilisticexpialidocious'
>>> len(s)
34
```

Δείτε επίσης:

**textseq** Οι συμβολοσειρές είναι παραδείγματα *τύπων ακολουθίας* και υποστηρίζουν τις κοινές πράξεις που υποστηρίζονται από τέτοιους τύπους.

**string-methods** Οι συμβολοσειρές υποστηρίζουν έναν μεγάλο αριθμό μεθόδων για βασικούς μετασχηματισμούς και αναζήτηση.

**f-strings** Κυριολεκτικές συμβολοσειρές που έχουν ενσωματωμένες εκφράσεις.

**formatstrings** Πληροφορίες σχετικά με τη μορφοποίηση συμβολοσειρών με τη μέθοδο `str.format()`.

**old-string-formatting** Οι παλιές λειτουργίες μορφοποίησης που καλούνται όταν οι συμβολοσειρές είναι ο αριστερός τελεστής του τελεστή `%` περιγράφονται λεπτομερέστερα εδώ.

### 3.1.3 Λίστες

Η Python γνωρίζει έναν αριθμό *σύνθετων* τύπων δεδομένων, που χρησιμοποιούνται για την ομαδοποίηση άλλων τιμών. Ο πιο ευέλικτος είναι ο τύπος *λίστα*, ο οποίος μπορεί να γραφτεί ως μια λίστα διαχωρισμένων με κόμμα τιμών (στοιχείων) ανάμεσα σε τετράγωνες αγκύλες. Οι λίστες μπορεί να περιέχουν στοιχεία διαφορετικών τύπων, αλλά συνήθως όλα τα στοιχεία έχουν τον ίδιο τύπο.

```
>>> squares = [1, 4, 9, 16, 25]
>>> squares
[1, 4, 9, 16, 25]
```

Όπως οι συμβολοσειρές (και όλοι οι άλλοι ενσωματωμένοι τύποι *sequence*), οι λίστες μπορούν να δεικτοδοτηθούν και να τεμαχιστούν:

```
>>> squares[0] # indexing returns the item
1
>>> squares[-1]
25
>>> squares[-3:] # slicing returns a new list
[9, 16, 25]
```

Όλοι οι τεμαχισμοί επιστρέφουν μια νέα λίστα που περιέχει τα ζητούμενα στοιχεία. Αυτό σημαίνει ότι ο ακόλουθος τεμαχισμός επιστρέφει ένα shallow copy (ρηχό αντίγραφο) της λίστας:

```
>>> squares[:]
[1, 4, 9, 16, 25]
```

Οι λίστες υποστηρίζουν επίσης λειτουργίες όπως σύνδεση:

```
>>> squares + [36, 49, 64, 81, 100]
[1, 4, 9, 16, 25, 36, 49, 64, 81, 100]
```

Σε αντίθεση με τις συμβολοσειρές, οι οποίες είναι *immutable*, οι λίστες είναι *mutable* τύπος, δηλαδή είναι δυνατόν να αλλάξετε το περιεχόμενό τους:

```
>>> cubes = [1, 8, 27, 65, 125] # something's wrong here
>>> 4 ** 3 # the cube of 4 is 64, not 65!
64
>>> cubes[3] = 64 # replace the wrong value
>>> cubes
[1, 8, 27, 64, 125]
```

You can also add new items at the end of the list, by using the `append()` *method* (we will see more about methods later):

```
>>> cubes.append(216) # add the cube of 6
>>> cubes.append(7 ** 3) # and the cube of 7
>>> cubes
[1, 8, 27, 64, 125, 216, 343]
```

Η ανάθεση σε τεμαχισμούς είναι επίσης δυνατή, και αυτό μπορεί ακόμη και να αλλάξει το μέγεθος της λίστας ή να τη διαγράψει εντελώς:

```
>>> letters = ['a', 'b', 'c', 'd', 'e', 'f', 'g']
>>> letters
['a', 'b', 'c', 'd', 'e', 'f', 'g']
>>> # replace some values
>>> letters[2:5] = ['C', 'D', 'E']
>>> letters
['a', 'b', 'C', 'D', 'E', 'f', 'g']
>>> # now remove them
>>> letters[2:5] = []
>>> letters
['a', 'b', 'f', 'g']
>>> # clear the list by replacing all the elements with an empty list
>>> letters[:] = []
>>> letters
[]
```

Η ενσωματωμένη συνάρτηση `len()` εφαρμόζεται επίσης στις λίστες:

```
>>> letters = ['a', 'b', 'c', 'd']
>>> len(letters)
4
```

Είναι δυνατό να εμφωλεύσετε λίστες (να δημιουργήσετε λίστες που περιέχουν άλλες λίστες), για παράδειγμα:

```
>>> a = ['a', 'b', 'c']
>>> n = [1, 2, 3]
>>> x = [a, n]
>>> x
[['a', 'b', 'c'], [1, 2, 3]]
>>> x[0]
['a', 'b', 'c']
>>> x[0][1]
'b'
```

## 3.2 Πρώτα Βήματα Προς Τον Προγραμματισμό

Of course, we can use Python for more complicated tasks than adding two and two together. For instance, we can write an initial sub-sequence of the [Fibonacci series](#) as follows:

```
>>> # Fibonacci series:
... # the sum of two elements defines the next
... a, b = 0, 1
>>> while a < 10:
...     print(a)
...     a, b = b, a+b
...
0
1
1
2
3
5
8
```

Αυτό το παράδειγμα εισάγει διάφορα νέα χαρακτηριστικά.

- Η πρώτη γραμμή περιέχει μια *πολλαπλή ανάθεση*: οι μεταβλητές `a` και `b` παίρνουν ταυτόχρονα τις νέες τιμές 0 και 1. Στην τελευταία γραμμή αυτό χρησιμοποιείται ξανά, αποδεικνύοντας ότι οι εκφράσεις στη

δεξιά πλευρά αξιολογούνται όλες πρώτα πριν γίνει οποιαδήποτε ανάθεση. Οι εκφράσεις στην δεξιά πλευρά αξιολογούνται από αριστερά προς τα δεξιά.

- Ο βρόχος `while` εκτελείται όσο η συνθήκη (εδώ: `a < 10`) παραμένει αληθής. Στην Python, όπως και στη C, οποιαδήποτε μη μηδενική ακέραια τιμή είναι αληθής, το μηδέν είναι ψευδές. Η συνθήκη μπορεί επίσης να είναι μια συμβολοσειρά ή μια λίστα, στην πραγματικότητα οποιαδήποτε ακολουθία• οτιδήποτε με μη μηδενικό μήκος είναι αληθές, κενές ακολουθίες είναι ψευδείς. Το τεστ που χρησιμοποιείται στο παράδειγμα είναι μια απλή σύγκριση. Οι τυπικοί τελεστές σύγκρισης γράφονται όπως στη C: `<` (μικρότερο από), `>` (μεγαλύτερο από), `==` (ίσο με), `<=` (μικρότερο ή ίσο με), `>=` (μεγαλύτερο ή ίσο με) και `!=` (μη ίσο με).
- Στο σώμα του βρόχου υπάρχει *εσοχή*: η εσοχή είναι ο τρόπος της Python να ομαδοποιεί εντολές. Στο διαδραστικό prompt, πρέπει να πληκτρολογήσετε ένα `tab` ή κενό(α) για κάθε γραμμή με εσοχή. Στην πράξη θα προετοιμάσετε πιο περίπλοκες εισαγωγές για την Python με έναν επεξεργαστή κειμένου• όλοι οι αξιοπρεπείς επεξεργαστές κειμένου διαθέτουν την δυνατότητα αυτόματης εσοχής. Όταν εισάγεται μια σύνθετη εντολή διαδραστικά, πρέπει να ακολουθείται από μια κενή γραμμή για να υποδηλώνει την ολοκλήρωση (αφού ο συντακτικός αναλυτής δεν μπορεί να μαντέψει πότε πληκτρολογήσατε την τελευταία γραμμή). Σημείωση ότι κάθε γραμμή μέσα σε ένα βασικό μπλοκ πρέπει να έχει την ίδια εσοχή.
- The `print()` function writes the value of the argument(s) it is given. It differs from just writing the expression you want to write (as we did earlier in the calculator examples) in the way it handles multiple arguments, floating point quantities, and strings. Strings are printed without quotes, and a space is inserted between items, so you can format things nicely, like this:

```
>>> i = 256*256
>>> print('The value of i is', i)
The value of i is 65536
```

Το όρισμα-κλειδί `end` μπορεί να χρησιμοποιηθεί για την αποφυγή της νέας γραμμής μετά την έξοδο, ή για να τελειώσετε την έξοδο με μια διαφορετική συμβολοσειρά:

```
>>> a, b = 0, 1
>>> while a < 1000:
...     print(a, end=', ')
...     a, b = b, a+b
...
0,1,1,2,3,5,8,13,21,34,55,89,144,233,377,610,987,
```

## Υποσημειώσεις





---

## Περισσότερα εργαλεία Ελέγχου Ροής

---

Besides the `while` statement just introduced, Python uses the usual flow control statements known from other languages, with some twists.

### 4.1 Προτάσεις `if`

Ίσως ο πιο γνωστός τύπος statement είναι η πρόταση `if`. Για παράδειγμα:

```
>>> x = int(input("Please enter an integer: "))
Please enter an integer: 42
>>> if x < 0:
...     x = 0
...     print('Negative changed to zero')
... elif x == 0:
...     print('Zero')
... elif x == 1:
...     print('Single')
... else:
...     print('More')
...
More
```

Μπορεί να υπάρχουν μηδέν ή περισσότερα μέρη `elif` και το τμήμα `else` είναι προαιρετικό. Το keyword “`elif`” είναι συντομογραφία του “`else if`”, και είναι χρήσιμη για να αποφύγετε την υπερβολική εσοχή. Μια ακολουθία keyword: `if ... elif ... elif ...` είναι υποκατάστατο των δηλώσεων `switch` ή `case` που υπάρχουν σε άλλες γλώσσες.

Εάν συγκρίνετε την ίδια τιμή με πολλές σταθερές ή ελέγχετε για συγκεκριμένους τύπους ή χαρακτηριστικά, μπορεί επίσης να βρείτε χρήσιμη τη δήλωση `match`. Για περισσότερες λεπτομέρειες, ανατρέξτε στο [Προτάσεις `match`](#).

## 4.2 Προτάσεις for

Η δήλωση `for` στην Python διαφέρει λίγο από αυτό που μπορεί να έχετε συνηθίσει στη C ή στην Pascal. Αντί να επαναλαμβάνετε πάντα μια αριθμητική πρόοδο αριθμών (όπως στη Pascal) ή να δίνετε στον χρήστη τη δυνατότητα να ορίσει τόσο το βήμα επανάληψης όσο και τη συνθήκη διακοπής (όπως C), η δήλωση της Python `for` επαναλαμβάνεται πάνω από τα στοιχεία οποιασδήποτε ακολουθίας (λίστας ή συμβολοσειράς), με τη σειρά που εμφανίζονται στην ακολουθία. Για παράδειγμα (χωρίς λογοπαίγνιο):

```
>>> # Measure some strings:
... words = ['cat', 'window', 'defenestrate']
>>> for w in words:
...     print(w, len(w))
...
cat 3
window 6
defenestrate 12
```

Ο κώδικας που τροποποιεί μια συλλογή ενώ επαναλαμβάνεται πάνω από την ίδια συλλογή μπορεί να είναι δύσκολος για να γίνει σωστός. Αντίθετα, είναι συνήθως πιο απλό να κάνετε `loop` πάνω από ένα αντίγραφο συλλογής ή να δημιουργήσετε μια νέα συλλογή:

```
# Create a sample collection
users = {'Hans': 'active', 'Éléonore': 'inactive', 'Barney': 'active'}

# Strategy: Iterate over a copy
for user, status in users.copy().items():
    if status == 'inactive':
        del users[user]

# Strategy: Create a new collection
active_users = {}
for user, status in users.items():
    if status == 'active':
        active_users[user] = status
```

## 4.3 Η συνάρτηση range ()

Εάν χρειάζεται να κάνετε επανάληψη σε μια ακολουθία αριθμών, η ενσωματωμένη (built-in) συνάρτηση `range ()` είναι χρήσιμη. Δημιουργεί αριθμητικές προόδους:

```
>>> for i in range(5):
...     print(i)
...
0
1
2
3
4
```

Το δεδομένο τελικό σημείο δεν αποτελεί ποτέ μέρος της παραγόμενης ακολουθίας· το `range (10)` δημιουργεί 10 τιμές, τους δείκτες για στοιχεία μιας ακολουθίας μήκους 10. Είναι δυνατόν να αφήσουμε το εύρος να ξεκινά από άλλο αριθμό ή για να καθορίσετε μια διαφορετική προσαύξηση (ακόμη και αρνητική, μερικές φορές αυτό ονομάζεται “βήμα”):

```
>>> list(range(5, 10))
[5, 6, 7, 8, 9]

>>> list(range(0, 10, 3))
```

(συνέχεια στην επόμενη σελίδα)

(συνεχίζεται από την προηγούμενη σελίδα)

```
[0, 3, 6, 9]

>>> list(range(-10, -100, -30))
[-10, -40, -70]
```

Για να γίνουν `iterate` οι δείκτες μια ακολουθίας, μπορείτε να συνδυάσετε τις `range()` και `len()` ως εξής:

```
>>> a = ['Mary', 'had', 'a', 'little', 'lamb']
>>> for i in range(len(a)):
...     print(i, a[i])
...
0 Mary
1 had
2 a
3 little
4 lamb
```

Στις περισσότερες τέτοιες περιπτώσεις, ωστόσο, είναι βολικό να χρησιμοποιήσετε τη συνάρτηση `enumerate()`, δείτε [Τεχνικές Looping](#).

Ένα περίεργο πράγμα συμβαίνει αν απλώς εκτυπώσετε ένα `range`:

```
>>> range(10)
range(0, 10)
```

Με πολλούς τρόπους το αντικείμενο που επιστρέφεται από το `range()` συμπεριφέρεται σαν να είναι μια λίστα, αλλά στην πραγματικότητα δεν είναι. Είναι ένα αντικείμενο που επιστρέφει τα διαδοχικά στοιχεία της επιθυμητής ακολουθίας όταν κάνετε επανάληψη πάνω του, αλλά δεν μπαίνει πραγματικά στη λίστα, εξοικονομώντας έτσι χώρο.

Λέμε ότι ένα τέτοιο αντικείμενο είναι *iterable*, δηλαδή, κατάλληλο ως στόχος για συναρτήσεις και κατασκευές που περιμένουν κάτι από το οποίο μπορούν να λάβουν διαδοχικά στοιχεία μέχρι να εξαντληθεί η προσφορά. Είδαμε ότι η δήλωση `for` είναι μια τέτοια κατασκευή, ενώ ένα παράδειγμα συνάρτησης που παίρνει ένα *iterable* είναι η `sum()`:

```
>>> sum(range(4)) # 0 + 1 + 2 + 3
6
```

Αργότερα θα δούμε περισσότερες συναρτήσεις που επιστρέφουν *iterables* και λαμβάνουν τους *iterables* ως ορίσματα. Στο κεφάλαιο [Δομές Δεδομένων](#), θα συζητήσουμε λεπτομερέστερα για το `list()`.

## 4.4 break and continue Statements, and else Clauses on Loops

The `break` statement, like in C, breaks out of the innermost enclosing `for` or `while` loop.

Loop statements may have an `else` clause; it is executed when the loop terminates through exhaustion of the iterable (with `for`) or when the condition becomes false (with `while`), but not when the loop is terminated by a `break` statement. This is exemplified by the following loop, which searches for prime numbers:

```
>>> for n in range(2, 10):
...     for x in range(2, n):
...         if n % x == 0:
...             print(n, 'equals', x, '*', n//x)
...             break
...     else:
...         # loop fell through without finding a factor
...         print(n, 'is a prime number')
...
2 is a prime number
```

(συνέχεια στην επόμενη σελίδα)

(συνεχίζεται από την προηγούμενη σελίδα)

```
3 is a prime number
4 equals 2 * 2
5 is a prime number
6 equals 2 * 3
7 is a prime number
8 equals 2 * 4
9 equals 3 * 3
```

(Yes, this is the correct code. Look closely: the `else` clause belongs to the `for` loop, **not** the `if` statement.)

When used with a loop, the `else` clause has more in common with the `else` clause of a `try` statement than it does with that of `if` statements: a `try` statement's `else` clause runs when no exception occurs, and a loop's `else` clause runs when no `break` occurs. For more on the `try` statement and exceptions, see [Διαχείριση Εξαιρέσεων](#).

The `continue` statement, also borrowed from C, continues with the next iteration of the loop:

```
>>> for num in range(2, 10):
...     if num % 2 == 0:
...         print("Found an even number", num)
...         continue
...     print("Found an odd number", num)
...
Found an even number 2
Found an odd number 3
Found an even number 4
Found an odd number 5
Found an even number 6
Found an odd number 7
Found an even number 8
Found an odd number 9
```

## 4.5 Προτάσεις `pass`

Η δήλωση `pass` δεν κάνει τίποτα. Μπορεί να χρησιμοποιηθεί όταν απαιτείται συντακτικά μια πρόταση, αλλά το πρόγραμμα δεν απαιτεί καμία ενέργεια. Για παράδειγμα:

```
>>> while True:
...     pass # Busy-wait for keyboard interrupt (Ctrl+C)
...
```

Αυτό χρησιμοποιείται συνήθως για τη δημιουργία ελάχιστων κλάσεων:

```
>>> class MyEmptyClass:
...     pass
...
```

Ένα άλλο μέρος `pass` που μπορεί να χρησιμοποιηθεί είναι ως `place-holder` για μια συνάρτηση ή το σώμα υπό όρους όταν εργάζεστε σε νέο κώδικα, επιτρέποντας σας να συνεχίσετε να σκέφτεστε σε ένα πιο αφηρημένο επίπεδο. Το `pass` αγνοείται σιωπηλά:

```
>>> def initlog(*args):
...     pass # Remember to implement this!
...
```

## 4.6 Προτάσεις match

Μια δήλωση `match` παίρνει μια έκφραση και συγκρίνει την τιμή της με διαδοχικά μοτίβα που δίνονται ως ένα ή περισσότερα μπλοκ πεζών-κεφαλαίων. Αυτή είναι επιφανειακά παρόμοια με μια πρόταση `switch` στην C, Java ή JavaScript (και πολλές άλλες γλώσσες), αλλά είναι πιο παρόμοια με την αντιστοίχιση προτύπων σε γλώσσες όπως η Rust ή η Haskell. Εκτελείται μόνο το πρώτο μοτίβο που ταιριάζει και μπορεί επίσης να εξαγάγει στοιχεία (στοιχεία ακολουθίας ή ιδιότητες αντικειμένου) από την τιμή σε μεταβλητές.

Η απλούστερη φόρμα συγκρίνει μια τιμή θέματος με ένα ή περισσότερα literals:

```
def http_error(status):
    match status:
        case 400:
            return "Bad request"
        case 404:
            return "Not found"
        case 418:
            return "I'm a teapot"
        case _:
            return "Something's wrong with the internet"
```

Σημειώστε το τελευταίο μπλοκ: το «variable name» `_` λειτουργεί ως *μπалаντέρ* και δεν αποτυγχάνει ποτέ να ταιριάζει. Εάν δεν ταιριάζει κανένα case, κανένας από τους κλάδους δεν εκτελείται.

Μπορείτε να συνδυάσετε πολλά γράμματα σε ένα μόνο μοτίβο χρησιμοποιώντας το `|` («ή»):

```
case 401 | 403 | 404:
    return "Not allowed"
```

Τα μοτίβα μπορεί να μοιάζουν με αναθέσεις unpacking, και μπορούν να χρησιμοποιηθούν για τη σύνδεση μεταβλητών:

```
# point is an (x, y) tuple
match point:
    case (0, 0):
        print("Origin")
    case (0, y):
        print(f"Y={y}")
    case (x, 0):
        print(f"X={x}")
    case (x, y):
        print(f"X={x}, Y={y}")
    case _:
        raise ValueError("Not a point")
```

Μελετήστε το ένα προσεκτικά! Το πρώτο μοτίβο έχει δύο literals, και μπορεί να θεωρηθεί ως επέκταση του literal μοτίβου που φαίνεται παραπάνω. Αλλά τα επόμενα δύο μοτίβα συνδυάζουν ένα literal σε μια μεταβλητή, και η μεταβλητή *δεσμεύει* μια τιμή από το θέμα (`point`). Το τέταρτο μοτίβο συλλαμβάνει δύο τιμές, γεγονός που το κάνει εννοιολογικά παρόμοιο με την ανάθεση unpacking `(x, y) = point`.

Εάν χρησιμοποιείτε κλάσεις για τη δομή των δεδομένων σας, μπορείτε να χρησιμοποιήσετε το όνομα της κλάσης ακολουθούμενο από μια λίστα ορισμάτων που μοιάζει με έναν κατασκευαστή, αλλά με τη δυνατότητα να συλλαμβάνει χαρακτηριστικά σε μεταβλητές:

```
class Point:
    x: int
    y: int

def where_is(point):
    match point:
        case Point(x=0, y=0):
            print("Origin")
```

(συνέχεια στην επόμενη σελίδα)

(συνεχίζεται από την προηγούμενη σελίδα)

```

case Point(x=0, y=y):
    print(f"Y={y}")
case Point(x=x, y=0):
    print(f"X={x}")
case Point():
    print("Somewhere else")
case _:
    print("Not a point")

```

Μπορείτε να χρησιμοποιήσετε παραμέτρους θέσης με ορισμένες ενσωματωμένες κλάσεις που παρέχουν μια σειρά για τα χαρακτηριστικά τους (π.χ. κλάσεις δεδομένων). Μπορείτε επίσης να ορίσετε μια συγκεκριμένη θέση για χαρακτηριστικά σε μοτίβα, ορίζοντας το ειδικό χαρακτηριστικό `__match_args__` στις κλάσεις σας. Εάν έχει οριστεί σε («x», «y»), τα ακόλουθα μοτίβα είναι όλα ισοδύναμα (και όλα δεσμεύουν το χαρακτηριστικό `y` στη μεταβλητή `var`):

```

Point(1, var)
Point(1, y=var)
Point(x=1, y=var)
Point(y=var, x=1)

```

Ένας συνιστώμενος τρόπος για να διαβάσετε τα μοτίβα είναι να τα δείτε ως μια εκτεταμένη μορφή αυτού που θα βάλατε στα αριστερά μιας ανάθεσης, για να κατανοήσετε ποιες μεταβλητές θα οριστούν σε τι. Μόνο τα ανεξάρτητα ονόματα (όπως `var` παραπάνω) εκχωρούνται από μια δήλωση αντιστοίχισης. Ονόματα με κουκκίδες (όπως `foo.bar`), ονόματα χαρακτηριστικών (τα `x=` και `y=` παραπάνω) ή ονόματα κλάσεων (αναγνωρίζονται από το «(...)» που βρίσκεται δίπλα όπως το `Point` παραπάνω) δεν ανατίθενται ποτέ.

Patterns can be arbitrarily nested. For example, if we have a short list of points, we could match it like this:

```

match points:
    case []:
        print("No points")
    case [Point(0, 0)]:
        print("The origin")
    case [Point(x, y)]:
        print(f"Single point {x}, {y}")
    case [Point(0, y1), Point(0, y2)]:
        print(f"Two on the Y axis at {y1}, {y2}")
    case _:
        print("Something else")

```

Μπορούμε να προσθέσουμε μια πρόταση `if` σε ένα μοτίβο, γνωστό ως «guard». Εάν το `guard` είναι `false`, το `match` συνεχίζει για να δοκιμάσει το επόμενο μπλοκ πεζών-κεφαλαίων. Λάβετε υπόψη ότι η σύλληψη της τιμής γίνεται πριν ο `guard` αξιολογηθεί:

```

match point:
    case Point(x, y) if x == y:
        print(f"Y=X at {x}")
    case Point(x, y):
        print(f"Not on the diagonal")

```

Πολλά άλλα βασικά χαρακτηριστικά αυτής της δήλωσης:

- Όπως το unpacking αναθέσεων, τα μοτίβα πλειάδας (tuple) και λίστας έχουν ακριβώς την ίδια σημασία και ταιριάζουν πραγματικά με αυθαίρετες ακολουθίες. Μια σημαντική εξαίρεση είναι ότι δεν ταιριάζουν με iterators ή συμβολοσειρές.
- Τα μοτίβα ακολουθίας υποστηρίζουν εκτεταμένο unpacking: `[x, y, *rest]` και `(x, y, *rest)` λειτουργεί παρόμοια με το unpacking αναθέσεων. Το όνομα μετά το `*` μπορεί επίσης να είναι `_`, οπότε το `(x, y, *_)` αντιστοιχεί σε μια ακολουθία τουλάχιστον δύο στοιχείων χωρίς να δεσμεύει τα υπόλοιπα στοιχεία.

- Μοτίβα αντιστοίχισης: {"bandwidth": b, "latency": l} καταγράφει τις τιμές "bandwidth" και "latency" από ένα λεξικό. Σε αντίθεση με τα μοτίβα ακολουθίας, επιπλέον κλειδιά αγνοούνται. Υποστηρίζεται επίσης το unpacking όπως το `**rest`. (Αλλά το `**_` θα ήταν περιττό, επομένως δεν επιτρέπεται.)
- Τα δευτερεύοντα μοτίβα μπορούν να αποτυπωθούν χρησιμοποιώντας το keyword `as`:

```
case (Point(x1, y1), Point(x2, y2) as p2): ...
```

θα καταγράψει το δεύτερο στοιχείο της εισόδου ως `p2` (εφόσον η είσοδος είναι μια ακολουθία δύο σημείων)

- Τα περισσότερα literals συγκρίνονται με ισότητα, ωστόσο τα singletons `True`, `False` και `None` συγκρίνονται με ταυτότητα.
- Τα μοτίβα μπορούν να χρησιμοποιούν ονομασμένες σταθερές. Αυτά πρέπει να είναι ονόματα με κουκκίδες για να μην ερμηνεύονται ως capture μεταβλητή:

```
from enum import Enum
class Color(Enum):
    RED = 'red'
    GREEN = 'green'
    BLUE = 'blue'

color = Color(input("Enter your choice of 'red', 'blue' or 'green': "))

match color:
    case Color.RED:
        print("I see red!")
    case Color.GREEN:
        print("Grass is green")
    case Color.BLUE:
        print("I'm feeling the blues :(")
```

Για πιο λεπτομερή επεξήγηση και πρόσθετα παραδείγματα, μπορείτε να δείτε το [PEP 636](#) το οποίο είναι γραμμένο σε μορφή εκμάθησης.

## 4.7 Καθορισμός Συναρτήσεων

Μπορούμε να δημιουργήσουμε μια συνάρτηση που γράφει τη σειρά Fibonacci σε ένα αυθαίρετο όριο:

```
>>> def fib(n):    # write Fibonacci series up to n
...     """Print a Fibonacci series up to n."""
...     a, b = 0, 1
...     while a < n:
...         print(a, end=' ')
...         a, b = b, a+b
...     print()
...
>>> # Now call the function we just defined:
... fib(2000)
0 1 1 2 3 5 8 13 21 34 55 89 144 233 377 610 987 1597
```

Το keyword `def` εισάγει μια συνάρτηση *ορισμός*. Πρέπει να ακολουθείται από το όνομα της συνάρτησης και τη λίστα των τυπικών παραμέτρων σε παρένθεση. Οι δηλώσεις που σχηματίζουν το σώμα της συνάρτησης ξεκινούν από την επόμενη γραμμή και πρέπει να είναι εσοχές.

Η πρώτη δήλωση του σώματος της συνάρτησης μπορεί προαιρετικά να είναι ένα literal συμβολοσειράς: αυτό το literal συμβολοσειράς είναι η συμβολοσειρά τεκμηρίωσης της συνάρτησης ή *docstring*. (Περισσότερα για τα docstring μπορείτε να βρείτε στην ενότητα [Συμβολοσειρές Τεκμηρίωσης](#).) Υπάρχουν εργαλεία που χρησιμοποιούν docstrings για την αυτόματη παραγωγή διαδικτυακής ή έντυπης τεκμηρίωσης ή για να αφήσουν τον

χρήστη να περιηγηθεί διαδραστικά στον κώδικα· είναι καλή πρακτική να συμπεριλαμβάνονται docstrings στον κώδικα που γράφετε, για αυτό κάντε το συνήθεια.

Η εκτέλεση μιας συνάρτησης εισάγει έναν νέο πίνακα συμβόλων που χρησιμοποιείται για τις τοπικές μεταβλητές της συνάρτησης. Πιο συγκεκριμένα, όλες οι εκχωρήσεις μεταβλητών σε μια συνάρτηση αποθηκεύουν την τιμή στον πίνακα τοπικών συμβόλων· ενώ οι αναφορές μεταβλητών κοιτάζονται πρώτα στον πίνακα τοπικών συμβόλων, στη συνέχεια στους πίνακες τοπικών συμβόλων των συναρτήσεων που περικλείουν, μετά στον πίνακα καθολικών συμβόλων και, τέλος, στον πίνακα ενσωματωμένων ονομάτων. Έτσι, οι καθολικές μεταβλητές και οι μεταβλητές των συναρτήσεων που περικλείουν δεν μπορούν να εκχωρηθούν ως μια τιμή μέσα σε μια συνάρτηση (εκτός εάν, για καθολικές μεταβλητές, που ονομάζονται σε μια δήλωση `global` ή, για μεταβλητές συναρτήσεων που περικλείουν, ονομάζονται ως μια δήλωση `nonlocal`), αν και μπορεί να αναφέρονται.

Οι πραγματικές παράμετροι (ορίσματα) σε μια κλήση συνάρτησης εισάγονται στον τοπικό πίνακα συμβόλων της καλούμενης συνάρτησης όταν αυτή καλείται· έτσι, τα ορίσματα μεταβιβάζονται χρησιμοποιώντας *call by value* (όπου η *value* είναι πάντα ένα αντικείμενο *reference*, όχι την τιμή του αντικειμένου).<sup>1</sup> Όταν μια συνάρτηση καλεί μια άλλη συνάρτηση ή καλεί τον εαυτό της αναδρομικά, δημιουργείται ένας νέος πίνακας τοπικών συμβόλων για αυτήν την κλήση.

Ένας ορισμός συνάρτησης συσχετίζει το όνομα της συνάρτησης με το αντικείμενο συνάρτησης στον τρέχοντα πίνακα συμβόλων. Ο διερμηνέας αναγνωρίζει το αντικείμενο στο οποίο επισημαίνεται αυτό το όνομα ως συνάρτηση που ορίζεται από τον χρήστη. Άλλα ονόματα μπορούν επίσης να δείχνουν το ίδιο αντικείμενο συνάρτησης και μπορούν επίσης να χρησιμοποιηθούν για πρόσβαση στη συνάρτηση:

```
>>> fib
<function fib at 10042ed0>
>>> f = fib
>>> f(100)
0 1 1 2 3 5 8 13 21 34 55 89
```

Προερχόμενοι από άλλες γλώσσες, μπορεί να αντιπαθείτε ότι το `fib` δεν είναι μια συνάρτηση αλλά μια διαδικασία, καθώς δεν επιστρέφει μια τιμή. Στην πραγματικότητα, ακόμη και συναρτήσεις χωρίς δήλωση `return` επιστρέφουν μια τιμή, αν και μάλλον βαρετή. Αυτή η τιμή ονομάζεται `None`. Η εγγραφή της τιμής `None` από τον διερμηνέα, εάν θα ήταν η μόνη τιμή που γράφεται. Μπορείτε να το δείτε αν το θέλετε πραγματικά χρησιμοποιώντας τη `print()`:

```
>>> fib(0)
>>> print(fib(0))
None
```

Είναι απλό να γράψετε μια συνάρτηση που επιστρέφει μια λίστα με τους αριθμούς της σειράς Fibonacci, αντί να την εκτυπώσετε:

```
>>> def fib2(n): # return Fibonacci series up to n
...     """Return a list containing the Fibonacci series up to n."""
...     result = []
...     a, b = 0, 1
...     while a < n:
...         result.append(a) # see below
...         a, b = b, a+b
...     return result
...
>>> f100 = fib2(100) # call it
>>> f100 # write the result
[0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89]
```

Αυτό το παράδειγμα, ως συνήθως, δείχνει μερικά νέα χαρακτηριστικά Python:

- Η δήλωση `return` επιστρέφει με μια τιμή από μια συνάρτηση. Το `return` χωρίς όρισμα έκφρασης, επιστρέφει το `None`. Η πτώση του τέλους μιας συνάρτησης επιστρέφει επίσης `None`.

<sup>1</sup> Στην πραγματικότητα, η κλήση με αναφορά αντικείμενου θα ήταν μια καλύτερη περιγραφή, καθώς εάν μεταβιβαστεί ένα μεταβλητό αντικείμενο, ο καλών θα δει τυχόν αλλαγές που κάνει ο καλών σε αυτό (στοιχεία που εισάγονται σε μια λίστα).



- The statement `result.append(a)` calls a *method* of the list object `result`. A method is a function that “belongs” to an object and is named `obj.methodname`, where `obj` is some object (this may be an expression), and `methodname` is the name of a method that is defined by the object’s type. Different types define different methods. Methods of different types may have the same name without causing ambiguity. (It is possible to define your own object types and methods, using *classes*, see [Κλάσεις](#)) The method `append()` shown in the example is defined for list objects; it adds a new element at the end of the list. In this example it is equivalent to `result = result + [a]`, but more efficient.

## 4.8 Περισσότερο για τον Καθορισμό Συναρτήσεων

Είναι επίσης δυνατός ο ορισμός συναρτήσεων με μεταβλητό αριθμό ορισμάτων. Υπάρχουν τρεις μορφές, που μπορούν να συνδυαστούν.

### 4.8.1 Προεπιλεγμένες Τιμές Ορίσματος

Η πιο χρήσιμη φόρμα είναι να καθορίσετε μια προεπιλεγμένη τιμή για ένα ή περισσότερα ορίσματα. Αυτό δημιουργεί μια συνάρτηση που μπορεί να κληθεί με λιγότερα ορίσματα από αυτά που έχει ορίσει ότι επιτρέπει. Για παράδειγμα:

```
def ask_ok(prompt, retries=4, reminder='Please try again!'):
    while True:
        ok = input(prompt)
        if ok in ('y', 'ye', 'yes'):
            return True
        if ok in ('n', 'no', 'nop', 'nope'):
            return False
        retries = retries - 1
        if retries < 0:
            raise ValueError('invalid user response')
    print(reminder)
```

Αυτή η συνάρτηση μπορεί να κληθεί με διάφορους τρόπους:

- δίνοντας μόνο το υποχρεωτικό όρισμα: `ask_ok('Do you really want to quit?')`
- δίνοντας ένα από τα προαιρετικά ορίσματα: `ask_ok('OK to overwrite the file?', 2)`
- ή δίνοντας όλα τα ορίσματα: `ask_ok('OK to overwrite the file?', 2, 'Come on, only yes or no!')`

Αυτό το παράδειγμα εισάγει επίσης το keyword `in`. Αυτό ελέγχει εάν μια ακολουθία περιέχει ή όχι μια συγκεκριμένη τιμή.

Οι προεπιλεγμένες τιμές αξιολογούνται στο σημείο του ορισμού της συνάρτησης στο πεδίο που *ορίζεται*, έτσι ώστε

```
i = 5

def f(arg=i):
    print(arg)

i = 6
f()
```

θα εκτυπώσει 5.

**Σημαντική προειδοποίηση:** Η προεπιλεγμένη τιμή αξιολογείται μόνο μία φορά. Αυτό κάνει τη διαφορά όταν η προεπιλογή είναι ένα μεταβλητό αντικείμενο, όπως μια λίστα, λεξικό ή στιγμιότυπα των περισσότερων κλάσεων. Για παράδειγμα, η ακόλουθη συνάρτηση συσσωρεύει τα ορίσματα που διαβάζονται σε αυτό σε επόμενες κλήσεις:

```
def f(a, L=[]):
    L.append(a)
    return L

print(f(1))
print(f(2))
print(f(3))
```

Αυτό θα εκτυπώσει

```
[1]
[1, 2]
[1, 2, 3]
```

Εάν δεν θέλετε να γίνεται κοινή χρήση της προεπιλογής μεταξύ των επόμενων κλήσεων, μπορείτε να γράψετε τη συνάρτηση ως εξής:

```
def f(a, L=None):
    if L is None:
        L = []
    L.append(a)
    return L
```

## 4.8.2 Ορίσματα Keyword

Οι συναρτήσεις μπορούν επίσης να κληθούν χρησιμοποιώντας το *keyword arguments* της μορφής `kwarg=value`. Για παράδειγμα, την ακόλουθη συνάρτηση:

```
def parrot(voltage, state='a stiff', action='vroom', type='Norwegian Blue'):
    print("-- This parrot wouldn't", action, end=' ')
    print("if you put", voltage, "volts through it.")
    print("-- Lovely plumage, the", type)
    print("-- It's", state, "!")
```

δέχεται ένα απαιτούμενο όρισμα (`voltage`) και τρία προαιρετικά ορίσματα (`state`, `action`, και `type`). Αυτή η συνάρτηση μπορεί να κληθεί με οποιονδήποτε από τους ακόλουθους τρόπους:

```
parrot(1000) # 1 positional argument
parrot(voltage=1000) # 1 keyword argument
parrot(voltage=1000000, action='VOOOOOM') # 2 keyword arguments
parrot(action='VOOOOOM', voltage=1000000) # 2 keyword arguments
parrot('a million', 'bereft of life', 'jump') # 3 positional arguments
parrot('a thousand', state='pushing up the daisies') # 1 positional, 1 keyword
```

αλλά όλες οι ακόλουθες κλήσεις θα ήταν άκυρες:

```
parrot() # required argument missing
parrot(voltage=5.0, 'dead') # non-keyword argument after a keyword argument
parrot(110, voltage=220) # duplicate value for the same argument
parrot(actor='John Cleese') # unknown keyword argument
```

Σε μια κλήση συνάρτησης, τα keyword ορίσματα πρέπει να ακολουθούν ορίσματα θέσης. Όλα τα keyword ορίσματα που διαβάζονται πρέπει να ταιριάζουν με ένα από τα ορίσματα που γίνονται δεκτά από τη συνάρτηση (π.χ. το `actor` δεν είναι έγκυρο όρισμα για τη συνάρτηση `parrot`), και η διάταξη τους δεν είναι σημαντική. Αυτό περιλαμβάνει επίσης μη προαιρετικά ορίσματα (π.χ. `parrot(voltage=1000)` είναι επίσης έγκυρο). Κανένα όρισμα δεν μπορεί να λάβει μια τιμή περισσότερες από μία φορές. Ακολουθεί ένα παράδειγμα που αποτυγχάνει λόγω αυτού του περιορισμού:

```
>>> def function(a):
...     pass
...
>>> function(0, a=0)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: function() got multiple values for argument 'a'
```

Όταν υπάρχει μια τελική επίσημη παράμετρος της μορφής `*name`, λαμβάνει ένα λεξικό (δείτε `typesmapping`) που περιέχει όλα τα keyword ορίσματα εκτός από αυτά που αντιστοιχούν σε μια επίσημη παράμετρο. Αυτό μπορεί να συνδυαστεί με μια επίσημη παράμετρος της μορφής `*name` (που περιγράφεται στην επόμενη υποενότητα) η οποία λαμβάνει ένα *tuple* που περιέχει τα ορίσματα θέσης πέρα από την επίσημη λίστα παραμέτρων. (Το `*name` πρέπει να εμφανίζεται πριν από το `**name`.) Για παράδειγμα, αν ορίσουμε μια συνάρτηση όπως αυτή:

```
def cheeseshop(kind, *arguments, **keywords):
    print("-- Do you have any", kind, "?")
    print("-- I'm sorry, we're all out of", kind)
    for arg in arguments:
        print(arg)
    print("-" * 40)
    for kw in keywords:
        print(kw, ":", keywords[kw])
```

Μπορεί να καλεστεί κάπως έτσι:

```
cheeseshop("Limburger", "It's very runny, sir.",
           "It's really very, VERY runny, sir.",
           shopkeeper="Michael Palin",
           client="John Cleese",
           sketch="Cheese Shop Sketch")
```

και φυσικά θα εκτυπώσει:

```
-- Do you have any Limburger ?
-- I'm sorry, we're all out of Limburger
It's very runny, sir.
It's really very, VERY runny, sir.
-----
shopkeeper : Michael Palin
client : John Cleese
sketch : Cheese Shop Sketch
```

Λάβετε υπόψη ότι η σειρά με την οποία εκτυπώνονται τα keyword ορίσματα είναι εγγυημένη ότι ταιριάζει με τη σειρά με την οποία δόθηκαν στην κλήση της συνάρτησης.

### 4.8.3 Ειδικές παράμετροι

Από προεπιλογή, τα ορίσματα μπορούν να μεταβιβαστούν σε μια συνάρτηση Python είτε με βάση τη θέση είτε ρητά με το keyword. Για αναγνωσιμότητα και απόδοση, είναι λογικό να περιοριστεί ο τρόπος με τον οποίο μπορούν να περάσουν τα ορίσματα, έτσι ώστε ένας προγραμματιστής να μην χρειάζεται να κοιτάξει τον ορισμό της συνάρτησης για να προσδιορίσει εάν τα στοιχεία μεταβιβάζονται κατά θέση, κατά θέση ή keyword, ή κατά keyword.

Ένας ορισμός συνάρτησης μπορεί να μοιάζει με αυτό:

```
def f(pos1, pos2, /, pos_or_kwd, *, kwd1, kwd2):
    -----
    |                |                |
    |                | Positional or keyword |
```

(συνέχεια στην επόμενη σελίδα)

(συνεχίζεται από την προηγούμενη σελίδα)

	- Keyword only
-- Positional only	

όπου τα / και \* είναι προαιρετικά. Εάν χρησιμοποιούνται, αυτά τα σύμβολα υποδεικνύουν το είδος της παραμέτρου με τον τρόπο που τα ορίσματα μπορούν να μεταβιβαστούν στη συνάρτηση: μόνο θέσης (positional-only), θέσης ή keyword (positional-or-keyword), και μόνο keyword (keyword-only). Οι keyword παράμετροι αναφέρονται επίσης ως ονομασμένες παράμετροι.

## Παράμετροι Θέσης ή Keyword

Εάν τα / και \* δεν υπάρχουν στον ορισμό της συνάρτησης, τα ορίσματα μπορούν να μεταβιβαστούν σε μια συνάρτηση ανά θέση ή κατά keyword.

## Παράμετροι Μόνο-Θέσης

Επανεξετάζοντας αυτό το θέμα λίγο πιο λεπτομερώς, είναι δυνατό να επισημάνετε ορισμένες παραμέτρους ως *μόνο θέσης*. Εάν *μόνο θέσης*, η σειρά των παραμέτρων έχει σημασία και οι παράμετροι δεν μπορούν να μεταβιβαστούν με keyword. Οι παράμετροι μόνο θέσης τοποθετούνται πριν από ένα / (προς τα εμπρός-κάθετος). Το / χρησιμοποιείται για να διαχωρίσει λογικά τις παραμέτρους μόνο θέσης από τις υπόλοιπες παραμέτρους. Εάν δεν υπάρχει το / στον ορισμό της συνάρτησης, δεν υπάρχουν παράμετροι μόνο θέσης.

Οι παράμετροι που ακολουθούν το / μπορεί να είναι *θέσης ή keyword* ή *μόνο keyword*.

## Ορίσματα μόνο Keyword

Για να επισημάνετε τις παραμέτρους ως *μόνο keyword*, υποδεικνύοντας ότι οι παράμετροι πρέπει να περάσουν από το keyword όρισμα, τοποθετήσετε ένα \* στη λίστα ορισμάτων ακριβώς πριν από την πρώτη παράμετρο *μόνο keyword*.

## Παραδείγματα Συναρτήσεων

Σκεφτείτε τα ακόλουθα παραδείγματα ορισμών συναρτήσεων δίνοντας ιδιαίτερη προσοχή στους δείκτες / και \*:

```
>>> def standard_arg(arg):
...     print(arg)
...
>>> def pos_only_arg(arg, /):
...     print(arg)
...
>>> def kwd_only_arg(*, arg):
...     print(arg)
...
>>> def combined_example(pos_only, /, standard, *, kwd_only):
...     print(pos_only, standard, kwd_only)
```

Ο πρώτος ορισμός συνάρτησης, `standard_arg`, η πιο οικεία μορφή, δεν θέτει περιορισμούς στη σύμβαση κλήσης και τα ορίσματα μπορούν να περάσουν από θέση ή από keyword:

```
>>> standard_arg(2)
2

>>> standard_arg(arg=2)
2
```

Η δεύτερη συνάρτηση `pos_only_arg` περιορίζεται στη χρήση μόνο παραμέτρων θέσης καθώς υπάρχει ένα / στον ορισμό της συνάρτησης:

```
>>> pos_only_arg(1)
1

>>> pos_only_arg(arg=1)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: pos_only_arg() got some positional-only arguments passed as keyword_
↳arguments: 'arg'
```

The third function `kwd_only_args` only allows keyword arguments as indicated by a `*` in the function definition:

```
>>> kwd_only_arg(3)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: kwd_only_arg() takes 0 positional arguments but 1 was given

>>> kwd_only_arg(arg=3)
3
```

Και το τελευταίο χρησιμοποιεί και τις τρεις συμβάσεις κλήσης στον ίδιο ορισμό συνάρτησης:

```
>>> combined_example(1, 2, 3)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: combined_example() takes 2 positional arguments but 3 were given

>>> combined_example(1, 2, kwd_only=3)
1 2 3

>>> combined_example(1, standard=2, kwd_only=3)
1 2 3

>>> combined_example(pos_only=1, standard=2, kwd_only=3)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: combined_example() got some positional-only arguments passed as keyword_
↳arguments: 'pos_only'
```

Τέλος, εξετάστε αυτόν τον ορισμό συνάρτησης που έχει μια πιθανή σύγκρουση μεταξύ του ορίσματος θέσης `name` και `**kwargs` που έχει ως κλειδί το `name`:

```
def foo(name, **kwargs):
    return 'name' in kwargs
```

Δεν υπάρχει καμία πιθανή κλήση που θα την κάνει να επιστρέψει `True` καθώς το keyword `'name'` θα συνδέεται πάντα με την πρώτη παράμετρο. Για παράδειγμα:

```
>>> foo(1, **{'name': 2})
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: foo() got multiple values for argument 'name'
>>>
```

Αλλά χρησιμοποιώντας / (ορίσματα μόνο θέσης), είναι δυνατό καθώς επιτρέπει το `name` ως όρισμα θέσης και το `'name'` ως κλειδί στα keyword ορίσματα:

```
def foo(name, /, **kwargs):
    return 'name' in kwargs
>>> foo(1, **{'name': 2})
True
```

Με άλλα λόγια, τα ονόματα των παραμέτρων μόνο θέσης μπορούν να χρησιμοποιηθούν σε `**kwargs` χωρίς ασάφεια.

## Ανακεφαλαίωση

Η περίπτωση χρήσης θα καθορίσει ποιες παραμέτρους θα χρησιμοποιηθούν στον ορισμό της συνάρτησης:

```
def f(pos1, pos2, /, pos_or_kwd, *, kwd1, kwd2):
```

Ως καθοδήγηση:

- Χρησιμοποιήστε τη θέση μόνο εάν θέλετε το όνομα των παραμέτρων να μην είναι διαθέσιμο στο χρήστη. Αυτό είναι χρήσιμο όταν τα ονόματα παραμέτρων δεν έχουν πραγματικό νόημα, εάν δεν θέλετε να επιβάλετε τη σειρά των ορισμάτων όταν καλείται η συνάρτηση ή εάν πρέπει να ληφθούν ορισμένες παράμετροι θέσης και αυθαίρετα keywords.
- Χρησιμοποιήστε keyword μόνο όταν τα ονόματα έχουν νόημα και ο ορισμός της συνάρτησης είναι πιο κατανοητός όταν είναι ρητός με ονόματα ή θέλετε να αποτρέψετε τους χρήστες να βασίζονται στη θέση του επιχειρήματος που μεταβιβάζεται.
- Για ένα API, χρησιμοποιήστε το μόνο θέσης για να αποτρέψετε τη διακοπή των αλλαγών τους API, εάν το όνομα της παραμέτρου τροποποιηθεί στο μέλλον.

## 4.8.4 Λίστες Αυθαίρετων Ορισμάτων

Τέλος, η λιγότερο συχνά χρησιμοποιούμενη επιλογή είναι να ορίσετε ότι μια συνάρτηση μπορεί να κληθεί με έναν αυθαίρετο αριθμό ορισμάτων. Αυτά τα ορίσματα θα τυλιχθούν σε μια πλειάδα (tuple) (βλ. *Πλειάδες (Tuples) και Ακολουθίες*). Πριν από τον μεταβλητό αριθμό ορισμάτων ενδέχεται να προκύψουν μηδέν ή περισσότερα κανονικά ορίσματα.

```
def write_multiple_items(file, separator, *args):
    file.write(separator.join(args))
```

Κανονικά, αυτά τα *variadic* ορίσματα θα είναι τελευταία στη λίστα των επίσημων παραμέτρων, επειδή συλλέγουν όλα τα υπόλοιπα ορίσματα εισόδου που μεταβιβάζονται στη συνάρτηση. Οποιοσδήποτε τυπικές παράμετροι που εμφανίζονται μετά την παράμετρο `*args` είναι “μόνο keyword” ορίσματα, που σημαίνει ότι μπορούν να χρησιμοποιηθούν μόνο ως λέξεις-κλειδιά και όχι ως ορίσματα θέσης.

```
>>> def concat(*args, sep="/"):
...     return sep.join(args)
...
>>> concat("earth", "mars", "venus")
'earth/mars/venus'
>>> concat("earth", "mars", "venus", sep=".")
'earth.mars.venus'
```

## 4.8.5 Unpacking Λίστες Ορισμάτων

Η αντίστροφη κατάσταση συμβαίνει όταν τα ορίσματα βρίσκονται ήδη σε μια λίστα ή πλειάδα (tuple), αλλά πρέπει να αποσυμπιεστούν για μια κλήση συνάρτησης που απαιτεί ξεχωριστά ορίσματα θέσης. Για παράδειγμα, η ενσωματωμένη (built-in) συνάρτηση `range()` αναμένει ξεχωριστά *start* και *stop* ορίσματα. Εάν δεν είναι διαθέσιμα ξεχωριστά, γράψτε την κλήση συνάρτησης με τον `*`-τελεστή για να αποσυμπιέσετε τα ορίσματα από μια λίστα ή πλειάδα (tuple):

```
>>> list(range(3, 6))           # normal call with separate arguments
[3, 4, 5]
>>> args = [3, 6]
```

(συνέχεια στην επόμενη σελίδα)

(συνεχίζεται από την προηγούμενη σελίδα)

```
>>> list(range(*args))           # call with arguments unpacked from a list
[3, 4, 5]
```

Με τον ίδιο τρόπο, τα λεξικά μπορούν να παραδίδουν keyword ορίσματα με τον `**`-τελεστή:

```
>>> def parrot(voltage, state='a stiff', action='vroom'):
...     print("-- This parrot wouldn't", action, end=' ')
...     print("if you put", voltage, "volts through it.", end=' ')
...     print("E's", state, "!")
...
>>> d = {"voltage": "four million", "state": "bleedin' demised", "action": "VOOM"}
>>> parrot(**d)
-- This parrot wouldn't VOOM if you put four million volts through it. E's bleedin
↪ demised !
```

## 4.8.6 Εκφράσεις Lambda

Μπορούν αν δημιουργηθούν μικρές ανώνυμες συναρτήσεις με το keyword `lambda`. Αυτή η συνάρτηση επιστρέφει το άθροισμα των δύο ορισμάτων της: `lambda a, b: a+b`. Οι συναρτήσεις `Lambda` μπορούν να χρησιμοποιηθούν όπου απαιτούνται αντικείμενα συνάρτησης. Περιορίζονται συντακτικά σε μία μόνο έκφραση. Σημασιολογικά, είναι απλώς syntactic sugar για έναν ορισμό κανονικής συνάρτησης. Όπως οι ορισμοί ένθετων συναρτήσεων, οι συναρτήσεις `lambda` μπορούν να παραπέμπουν σε μεταβλητές από το πεδίο που περιέχει:

```
>>> def make_incrementor(n):
...     return lambda x: x + n
...
>>> f = make_incrementor(42)
>>> f(0)
42
>>> f(1)
43
```

Το παραπάνω παράδειγμα χρησιμοποιεί μια έκφραση `lambda` για να επιστρέψει μια συνάρτηση. Μια άλλη χρήση είναι η μετάδοση μιας μικρής συνάρτησης ως όρισμα:

```
>>> pairs = [(1, 'one'), (2, 'two'), (3, 'three'), (4, 'four')]
>>> pairs.sort(key=lambda pair: pair[1])
>>> pairs
[(4, 'four'), (1, 'one'), (3, 'three'), (2, 'two')]
```

## 4.8.7 Συμβολοσειρές Τεκμηρίωσης

Ακολουθούν ορισμένες συμβάσεις σχετικά με το περιεχόμενο και τη μορφοποίηση των συμβολοσειρών τεκμηρίωσης.

Η πρώτη γραμμή πρέπει να είναι πάντα μια σύντομη, συνοπτική περίληψη του σκοπού του αντικειμένου. Για συντομία, δεν πρέπει να αναφέρει ρητά το όνομα ή τον τύπο του αντικειμένου, καθώς αυτά είναι διαθέσιμα με άλλα μέσα (εκτός εάν το όνομα είναι ρήμα που περιγράφει τη λειτουργία της συνάρτησης). Αυτή η γραμμή πρέπει να ξεκινά με κεφαλαίο γράμμα και να τελειώνει με τελεία.

Εάν υπάρχουν περισσότερες γραμμές στη συμβολοσειρά τεκμηρίωσης, η δεύτερη γραμμή θα πρέπει να είναι κενή, διαχωρίζοντας οπτικά τη σύντομη από την υπόλοιπη περιγραφή. Οι ακόλουθες γραμμές πρέπει να είναι μία ή περισσότερες παράγραφοι που περιγράφουν τις συμβάσεις κλήσης του αντικειμένου, τις παρενέργειές του κ.λπ..

Ο parser της Python δεν αφαιρεί την εσοχή από τα literals της συμβολοσειράς πολλών γραμμών στην Python, επομένως τα εργαλεία που επεξεργάζονται την τεκμηρίωση πρέπει να αφαιρέσουν την εσοχή εάν είναι επι-

θυμητό. Αυτό γίνεται χρησιμοποιώντας την ακόλουθη σύμβαση. Η πρώτη μη κενή γραμμή *μετά* την πρώτη γραμμή της συμβολοσειράς καθορίζει το μέγεθος της εσοχής για ολόκληρη τη συμβολοσειρά τεκμηρίωσης. (Δεν μπορούμε να χρησιμοποιήσουμε την πρώτη γραμμή αφού είναι γενικά δίπλα στα εισαγωγικά της συμβολοσειράς, επομένως η εσοχή της δεν είναι εμφανής στο literal της συμβολοσειράς.) Το κενό διάστημα «ισοδύναμο» σε αυτήν την εσοχή αφαιρείται στη συνέχεια από την αρχή όλων των γραμμών της συμβολοσειράς. Οι γραμμές που έχουν μικρότερη εσοχή δεν θα πρέπει να εμφανίζονται, αλλά αν εμφανιστούν θα πρέπει να αφαιρεθεί όλο το αρχικό κενό τους. Η ισοδυναμία των κενών διαστημάτων θα πρέπει να ελέγχεται μετά την επέκταση των καρτελών (σε 8 κενά, κανονικά).

Ακολουθεί ένα παράδειγμα ενός πολλαπλών γραμμών docstring:

```
>>> def my_function():
...     """Do nothing, but document it.
...
...     No, really, it doesn't do anything.
...     """
...     pass
>>> print(my_function.__doc__)
Do nothing, but document it.

    No, really, it doesn't do anything.
```

## 4.8.8 Annotations Συναρτήσεων

Το Function annotations είναι εντελώς προαιρετικές πληροφορίες μεταδεδομένων σχετικά με τους τύπους χρησιμοποιούνται από συναρτήσεις που καθορίζονται από το χρήστη (δείτε [PEP 3107](#) και [PEP 484](#) για περισσότερες πληροφορίες).

*Annotations* are stored in the `__annotations__` attribute of the function as a dictionary and have no effect on any other part of the function. Parameter annotations are defined by a colon after the parameter name, followed by an expression evaluating to the value of the annotation. Return annotations are defined by a literal `->`, followed by an expression, between the parameter list and the colon denoting the end of the `def` statement. The following example has a required argument, an optional argument, and the return value annotated:

```
>>> def f(ham: str, eggs: str = 'eggs') -> str:
...     print("Annotations:", f.__annotations__)
...     print("Arguments:", ham, eggs)
...     return ham + ' and ' + eggs
>>> f('spam')
Annotations: {'ham': <class 'str'>, 'return': <class 'str'>, 'eggs': <class 'str'>}
Arguments: spam eggs
'spam and eggs'
```

## 4.9 Intermezzo: Στυλ Κώδικα

Τώρα που πρόκειται να γράψετε μεγαλύτερα, και πιο σύνθετα κομμάτια της Python, είναι η κατάλληλη στιγμή να μιλήσετε για *στυλ κώδικα*. Οι περισσότερες γλώσσες μπορούν να γραφτούν (ή πιο συνοπτικές, *μοροφοποιημένες*) σε διαφορετικά στυλ· μερικές είναι πιο ευανάγνωστες από άλλες. Το να διευκολύνετε τους άλλους να διαβάσουν τον κώδικά σας είναι πάντα μια καλή ιδέα και η υιοθέτηση ενός ωραίου στυλ κώδικα βοηθάει πάρα πολύ σε αυτό.

Για την Python, το [PEP 8](#) έχει αναδειχθεί ως οδηγός στυλ στον οποίο τηρούν τα περισσότερα έργα· προωθεί ένα πολύ ευανάγνωστο και ευχάριστο στυλ κώδικα. Κάθε προγραμματιστής Python θα πρέπει να το διαβάσει κάποια στιγμή· εδώ είναι τα πιο σημαντικά σημεία που εξάγονται για εσάς:

- Χρησιμοποιήστε εσοχή 4 διαστημάτων και όχι tabs.



Τα 4 κενά είναι ένας καλός συμβιβασμός μεταξύ της μικρής εσοχής (επιτρέπει μεγαλύτερο βάθος εμφώλευσης) και της μεγάλης εσοχής (ευκολότερη στην ανάγνωση). Τα tabs δημιουργούν σύγχυση, και είναι καλύτερο να παραμείνουν απέξω.

- Τυλίξτε τις γραμμές έτσι ώστε να μην υπερβαίνουν τους 79 χαρακτήρες.

Αυτό βοηθά του χρήστες με μικρές οθόνες και καθιστά δυνατή την ύπαρξη πολλών αρχείων κώδικα δίπλα-δίπλα σε μεγαλύτερες οθόνες.

- Χρησιμοποιείτε κενές γραμμές για να διαχωρίσετε συναρτήσεις και κλάσεις και μεγαλύτερα μπλοκ κώδικα μέσα συναρτήσεις.
- Όταν είναι δυνατόν, βάλτε σχόλια σε μια δική τους γραμμή.
- Χρησιμοποιήστε docstrings.
- Χρησιμοποιήστε κενά γύρω από τελεστές και μετά από κόμματα, αλλά όχι απευθείας μέσα δε δομές αγκύλων: `a = f(1, 2) + g(3, 4)`.
- Ονομάστε τις κλάσεις και τις συναρτήσεις σας με συνέπεια· η σύμβαση είναι να χρησιμοποιείτε `UpperCamelCase` για τις κλάσεις και `lowercase_with_underscores` για τις συναρτήσεις και τις μεθόδους. Χρησιμοποιείτε πάντα το `self` ως όνομα για το πρώτο όρισμα μεθόδου (δείτε [Μια πρώτη ματιά στις Κλάσεις](#) για περισσότερα σχετικά με τις κλάσεις και τις μεθόδους).
- Μην χρησιμοποιείτε φανταχτερές κωδικοποιήσεις εάν ο κώδικας σας προορίζεται να χρησιμοποιηθεί σε διεθνή περιβάλλοντα. Η προεπιλογή της Python, UTF-8, ή ακόμα και το απλό ASCII λειτουργούν καλύτερα σε κάθε περίπτωση.
- Ομοίως, μη χρησιμοποιείτε χαρακτήρες που δεν είναι ASCII σε αναγνωριστικά εάν υπάρχει μόνο η παραμικρή πιθανότητα οι άνθρωποι που μιλούν διαφορετική γλώσσα να διαβάσουν ή να διατηρήσουν τον κώδικα.

## Υποσημειώσεις



## Δομές Δεδομένων

Αυτό το κεφάλαιο περιγράφει ορισμένα πράγματα τα οποία έχετε μάθει ήδη με περισσότερες λεπτομέρειες και προσθέτει επίσης μερικά νέα.

## 5.1 Περισσότερα για τις Λίστες

Ο τύπος δεδομένων λίστας έχει μερικές ακόμη μεθόδους. Ακολουθούν όλες οι μέθοδοι αντικειμένων τύπου λίστας:

`list.append(x)`

Add an item to the end of the list. Equivalent to `a[len(a):] = [x]`.

`list.extend(iterable)`

Extend the list by appending all the items from the iterable. Equivalent to `a[len(a):] = iterable`.

`list.insert(i, x)`

Εισάγει ένα στοιχείο σε μια δεδομένη θέση. Το πρώτο όρισμα είναι το index του στοιχείου πριν από το οποίο θα εισαχθεί, επομένως `a.insert(0, x)` εισάγεται στο μπροστινό μέρος της λίστας, και το `a.insert(len(a), x)` ισοδυναμεί με `a.append(x)`.

`list.remove(x)`

Καταργεί το πρώτο στοιχείο από τη λίστα του οποίου η τιμή είναι ίση με `x`. Κάνει `raise` ένα `ValueError` εάν δεν υπάρχει τέτοιο στοιχείο.

`list.pop([i])`

Remove the item at the given position in the list, and return it. If no index is specified, `a.pop()` removes and returns the last item in the list. (The square brackets around the `i` in the method signature denote that the parameter is optional, not that you should type square brackets at that position. You will see this notation frequently in the Python Library Reference.)

`list.clear()`

Remove all items from the list. Equivalent to `del a[:]`.

`list.index(x[, start[, end]])`

Επιστρέφει το μηδενικό index στη λίστα του πρώτου στοιχείου του οποίου η τιμή είναι ίση με `x`. Κάνει `raise` ένα `ValueError` εάν δεν υπάρχει τέτοιο στοιχείο.

Τα προαιρετικά ορίσματα `start` και `end` ερμηνεύονται όπως στη σημειογραφία slice και χρησιμοποιούνται για τον περιορισμό της αναζήτησης σε μια συγκεκριμένη υποακολουθία της λίστας. Ο επιστρεφόμενος δείκτης υπολογίζεται σε σχέση με την αρχή της πλήρους ακολουθίας αντί για το όρισμα `start`.

`list.count(x)`

Επιστρέφει τον αριθμό των φορών που εμφανίζεται το *x* στη λίστα.

`list.sort(*, key=None, reverse=False)`

Ταξινομεί τα στοιχεία της λίστας στη θέση τους (τα ορίσματα μπορούν να χρησιμοποιηθούν για προσαρμογή ταξινόμησης, βλ. `sorted()` για την εξήγησή τους).

`list.reverse()`

Αντιστρέφει τα στοιχεία της λίστας στη θέση τους.

`list.copy()`

Return a shallow copy of the list. Equivalent to `a[:]`.

Ένα παράδειγμα που χρησιμοποιεί τις περισσότερες από τις μεθόδους της λίστας:

```
>>> fruits = ['orange', 'apple', 'pear', 'banana', 'kiwi', 'apple', 'banana']
>>> fruits.count('apple')
2
>>> fruits.count('tangerine')
0
>>> fruits.index('banana')
3
>>> fruits.index('banana', 4)  # Find next banana starting a position 4
6
>>> fruits.reverse()
>>> fruits
['banana', 'apple', 'kiwi', 'banana', 'pear', 'apple', 'orange']
>>> fruits.append('grape')
>>> fruits
['banana', 'apple', 'kiwi', 'banana', 'pear', 'apple', 'orange', 'grape']
>>> fruits.sort()
>>> fruits
['apple', 'apple', 'banana', 'banana', 'grape', 'kiwi', 'orange', 'pear']
>>> fruits.pop()
'pear'
```

Μπορεί να έχετε παρατηρήσει ότι μέθοδοι όπως `insert`, `remove` or `sort` που τροποποιούν μόνο τη λίστα δεν έχουν εκτυπωμένη τιμή επιστροφής – επιστρέφουν το προεπιλεγμένο (default) `None`.<sup>1</sup> Αυτή είναι μια αρχή σχεδιασμού για όλες τις μεταβλητές δομές δεδομένων στην Python.

Another thing you might notice is that not all data can be sorted or compared. For instance, `[None, 'hello', 10]` doesn't sort because integers can't be compared to strings and `None` can't be compared to other types. Also, there are some types that don't have a defined ordering relation. For example, `3+4j < 5+7j` isn't a valid comparison.

### 5.1.1 Χρήση Λιστών ως Στοιβές (Stacks)

The list methods make it very easy to use a list as a stack, where the last element added is the first element retrieved («last-in, first-out»). To add an item to the top of the stack, use `append()`. To retrieve an item from the top of the stack, use `pop()` without an explicit index. For example:

```
>>> stack = [3, 4, 5]
>>> stack.append(6)
>>> stack.append(7)
>>> stack
[3, 4, 5, 6, 7]
>>> stack.pop()
7
>>> stack
[3, 4, 5, 6]
```

(συνέχεια στην επόμενη σελίδα)

<sup>1</sup> Άλλες γλώσσες ενδέχεται να επιστρέφουν το μεταλλαγμένο αντικείμενο, το οποίο επιτρέπει την αλυσιδωτή εκτέλεση μεθόδων, όπως `d->insert("a")->remove("b")->sort();`.

(συνεχίζεται από την προηγούμενη σελίδα)

```
>>> stack.pop()
6
>>> stack.pop()
5
>>> stack
[3, 4]
```

### 5.1.2 Χρήση λιστών ως Ουρές (Queues)

Είναι επίσης δυνατό να χρησιμοποιηθεί μια λίστα ως ουρά (queue), όπου το πρώτο στοιχείο που προστίθεται είναι το πρώτο στοιχείο που ανακτάται («first-in, first-out»). • ωστόσο, οι λίστες δεν είναι αποτελεσματικές για αυτόν τον σκοπό. Ενώ το να προσθέσεις και να αφαιρέσεις (στοιχεία) στο τέλος της λίστας είναι γρήγορο, κάνοντας αυτές τις προσθήκες και τις αφαιρέσεις (στοιχείων) στην αρχή της λίστας είναι αργό (επειδή όλα τα στοιχεία πρέπει να μετατοπιστούν κατά ένα).

Για να εφαρμόσετε μια ουρά (queue), χρησιμοποιήστε την `collections.deque` η οποία σχεδιάστηκε για να έχει γρήγορες προσθήκες και αφαιρέσεις και από τα δύο άκρα. Για παράδειγμα:ˆ

```
>>> from collections import deque
>>> queue = deque(["Eric", "John", "Michael"])
>>> queue.append("Terry")           # Terry arrives
>>> queue.append("Graham")         # Graham arrives
>>> queue.popleft()                 # The first to arrive now leaves
'Eric'
>>> queue.popleft()                 # The second to arrive now leaves
'John'
>>> queue                           # Remaining queue in order of arrival
deque(['Michael', 'Terry', 'Graham'])
```

### 5.1.3 Comprehensions Λίστας

Τα comprehensions λίστας παρέχουν ένα συνοπτικό τρόπο δημιουργίας λιστών. Οι συνήθεις εφαρμογές είναι η δημιουργία νέων λιστών όπου κάθε στοιχείο είναι το αποτέλεσμα κάποιων πράξεων που εφαρμόζονται σε κάθε μέλος μιας άλλης ακολουθίας ή iterable, ή η δημιουργία μιας υποακολουθίας αυτών των στοιχείων που ικανοποιούν μια συγκεκριμένη συνθήκη.

Για παράδειγμα, ας υποθέσουμε ότι θέλουμε να δημιουργήσουμε μια λίστα τετραγώνων όπως:

```
>>> squares = []
>>> for x in range(10):
...     squares.append(x**2)
...
>>> squares
[0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
```

Λάβετε υπόψη ότι αυτό δημιουργεί (ή αντικαθιστά) μια μεταβλητή με το όνομα `x` που εξακολουθεί να υπάρχει μετά την ολοκλήρωση της loop. Μπορούμε να υπολογίσουμε τη λίστα των τετραγώνων χωρίς παρενέργειες χρησιμοποιώντας:

```
squares = list(map(lambda x: x**2, range(10)))
```

ή, ισοδύναμα:

```
squares = [x**2 for x in range(10)]
```

που είναι πιο συνοπτικό και ευανάγνωστο.

Ένα comprehension λίστας αποτελείται από αγκύλες που περιέχουν μια έκφραση ακολουθούμενη από μια πρόταση `for`, στη συνέχεια μηδέν ή περισσότερες προτάσεις `for` ή `if`. Το αποτέλεσμα θα είναι μια νέα λίστα που προκύπτει από την αξιολόγηση της έκφρασης στο πλαίσιο των προτάσεων `for` και `if` που την ακολουθούν. Για παράδειγμα, αυτή η λίστα συνδυάζει τα στοιχεία δύο λιστών εάν δεν είναι ίσες:

```
>>> [(x, y) for x in [1,2,3] for y in [3,1,4] if x != y]
[(1, 3), (1, 4), (2, 3), (2, 1), (2, 4), (3, 1), (3, 4)]
```

και ισοδυναμεί με:

```
>>> combs = []
>>> for x in [1,2,3]:
...     for y in [3,1,4]:
...         if x != y:
...             combs.append((x, y))
...
>>> combs
[(1, 3), (1, 4), (2, 3), (2, 1), (2, 4), (3, 1), (3, 4)]
```

Σημειώστε πώς η σειρά των δηλώσεων `for` και `if` είναι ίδια και στα δύο αποσπάσματα.

Εάν η έκφραση είναι πλειάδα (π.χ. το `(x, y)` στο προηγούμενο παράδειγμα), πρέπει να μπει σε παρένθεση.

```
>>> vec = [-4, -2, 0, 2, 4]
>>> # create a new list with the values doubled
>>> [x*2 for x in vec]
[-8, -4, 0, 4, 8]
>>> # filter the list to exclude negative numbers
>>> [x for x in vec if x >= 0]
[0, 2, 4]
>>> # apply a function to all the elements
>>> [abs(x) for x in vec]
[4, 2, 0, 2, 4]
>>> # call a method on each element
>>> freshfruit = [' banana', ' loganberry ', 'passion fruit ']
>>> [weapon.strip() for weapon in freshfruit]
['banana', 'loganberry', 'passion fruit']
>>> # create a list of 2-tuples like (number, square)
>>> [(x, x**2) for x in range(6)]
[(0, 0), (1, 1), (2, 4), (3, 9), (4, 16), (5, 25)]
>>> # the tuple must be parenthesized, otherwise an error is raised
>>> [x, x**2 for x in range(6)]
File "<stdin>", line 1
    [x, x**2 for x in range(6)]
    ^^^^^^^
SyntaxError: did you forget parentheses around the comprehension target?
>>> # flatten a list using a listcomp with two 'for'
>>> vec = [[1,2,3], [4,5,6], [7,8,9]]
>>> [num for elem in vec for num in elem]
[1, 2, 3, 4, 5, 6, 7, 8, 9]
```

Τα comprehensions λίστας μπορεί να περιέχουν σύνθετες εκφράσεις και ένθετες συναρτήσεις:

```
>>> from math import pi
>>> [str(round(pi, i)) for i in range(1, 6)]
['3.1', '3.14', '3.142', '3.1416', '3.14159']
```

### 5.1.4 Comprehensions Ένθετων Λιστών

Η αρχική έκφραση σε ένα comprehension λίστας μπορεί να είναι οποιαδήποτε αυθαίρετη έκφραση, συμπεριλαμβανομένης ενός άλλου comprehension λίστας.

Σκεφτείτε το ακόλουθο παράδειγμα μιας μήτρας 3x4 που υλοποιήθηκε ως μια λίστα 3 λιστών μήκους 4:

```
>>> matrix = [
...     [1, 2, 3, 4],
...     [5, 6, 7, 8],
...     [9, 10, 11, 12],
... ]
```

Το ακόλουθο comprehension λίστας θα μεταφέρει γραμμές και στήλες:

```
>>> [[row[i] for row in matrix] for i in range(4)]
[[1, 5, 9], [2, 6, 10], [3, 7, 11], [4, 8, 12]]
```

As we saw in the previous section, the nested listcomp is evaluated in the context of the `for` that follows it, so this example is equivalent to:

```
>>> transposed = []
>>> for i in range(4):
...     transposed.append([row[i] for row in matrix])
...
>>> transposed
[[1, 5, 9], [2, 6, 10], [3, 7, 11], [4, 8, 12]]
```

το οποίο, με τη σειρά του, είναι το ίδιο με:

```
>>> transposed = []
>>> for i in range(4):
...     # the following 3 lines implement the nested listcomp
...     transposed_row = []
...     for row in matrix:
...         transposed_row.append(row[i])
...     transposed.append(transposed_row)
...
>>> transposed
[[1, 5, 9], [2, 6, 10], [3, 7, 11], [4, 8, 12]]
```

Στον πραγματικό κόσμο, θα πρέπει να προτιμάτε τις ενσωματωμένες (built-in) συναρτήσεις από τις σύνθετες εντολές ροής. Η συνάρτηση `zip()` θα έκανε εξαιρετική δουλειά για αυτήν την περίπτωση χρήσης:

```
>>> list(zip(*matrix))
[(1, 5, 9), (2, 6, 10), (3, 7, 11), (4, 8, 12)]
```

Δείτε *Unpacking Λίστες Ορισμάτων* για λεπτομέρειες σχετικά με τον αστερίσκο σε αυτήν τη γραμμή.

## 5.2 Η δήλωση `del`

There is a way to remove an item from a list given its index instead of its value: the `del` statement. This differs from the `pop()` method which returns a value. The `del` statement can also be used to remove slices from a list or clear the entire list (which we did earlier by assignment of an empty list to the slice). For example:

```
>>> a = [-1, 1, 66.25, 333, 333, 1234.5]
>>> del a[0]
>>> a
[1, 66.25, 333, 333, 1234.5]
>>> del a[2:4]
```

(συνέχεια στην επόμενη σελίδα)

(συνεχίζεται από την προηγούμενη σελίδα)

```
>>> a
[1, 66.25, 1234.5]
>>> del a[:]
>>> a
[]
```

Το `del` μπορεί επίσης να χρησιμοποιηθεί για τη διαγραφή ολόκληρων μεταβλητών:

```
>>> del a
```

Η αναφορά στο όνομα `a` στο εξής είναι ένα σφάλμα (τουλάχιστον μέχρι να του εκχωρηθεί άλλη τιμή). Θα βρούμε άλλες χρήσεις για το `del` αργότερα.

## 5.3 Πλειάδες (Tuples) και Ακολουθίες

Είδαμε ότι οι λίστες και οι συμβολοσειρές (strings) έχουν πολλές κοινές ιδιότητες, όπως λειτουργίες indexing και slicing. Είναι δύο παραδείγματα τύπων δεδομένων *sequence* (δείτε `typeseq`). Δεδομένου ότι η Python είναι μια εξελισσόμενη γλώσσα, άλλοι τύποι δεδομένων ακολουθίας μπορούν να προστεθούν. Υπάρχει επίσης ένας άλλος τυπικός τύπος δεδομένων ακολουθίας `type`: the *πλειάδα* (*tuple*).

Μια πλειάδα (*tuple*) αποτελείται από έναν αριθμό τιμών που χωρίζονται με κόμματα, για παράδειγμα:

```
>>> t = 12345, 54321, 'hello!'
>>> t[0]
12345
>>> t
(12345, 54321, 'hello!')
>>> # Tuples may be nested:
... u = t, (1, 2, 3, 4, 5)
>>> u
((12345, 54321, 'hello!'), (1, 2, 3, 4, 5))
>>> # Tuples are immutable:
... t[0] = 88888
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'tuple' object does not support item assignment
>>> # but they can contain mutable objects:
... v = ([1, 2, 3], [3, 2, 1])
>>> v
([1, 2, 3], [3, 2, 1])
```

Όπως βλέπετε, οι πλειάδες (*tuples*) στην έξοδο περικλείονται πάντα σε παρενθέσεις, έτσι ώστε οι ένθετες πλειάδες (*tuples*) να ερμηνεύονται σωστά• μπορούν να εισαχθούν με ή χωρίς περιβάλλουσες παρενθέσεις, αν και συχνά οι παρενθέσεις είναι απαραίτητες ούτως ή άλλως (αν η πλειάδα είναι μέρος μιας μεγαλύτερης έκφρασης). Δεν είναι δυνατή η αντιστοίχιση σε μεμονωμένα στοιχεία μιας πλειάδας, ωστόσο είναι δυνατό να δημιουργηθούν πλειάδες (*tuples*) που περιέχουν μεταβλητά αντικείμενα, όπως λίστες.

Αν και οι πλειάδες (*tuples*) μπορεί να φαίνονται παρόμοιες με λίστες, χρησιμοποιούνται συχνά σε διαφορετικές καταστάσεις και για διαφορετικούς σκοπούς. Οι πλειάδες (*tuples*) είναι *immutable*, και συνήθως περιέχουν μια ετερογενή ακολουθία στοιχείων στα οποία η πρόσβαση γίνεται μέσω unpacking (δείτε παρακάτω σε αυτήν την ενότητα) ή το indexing (ή ακόμα και κατά χαρακτηριστικό στην περίπτωση *namedtuples*). Οι λίστες είναι *mutable*, και τα στοιχεία τους είναι συνήθως ομοιογενή και προσπελάζονται με επανάληψη στη λίστα.

Ένα ειδικό πρόβλημα είναι η κατασκευή πλειάδων (*tuples*) που περιέχουν 0 ή 1 στοιχεία: η σύνταξη έχει κάποιες επιπλέον ιδιορρυθμίες για να τις προσαρμόσει. Οι κενές πλειάδες κατασκευάζονται από ένα κενό ζευγάρι παρενθέσεων, μια πλειάδα (*tuple*) με ένα στοιχείο δημιουργείται ακολουθώντας μια τιμή με κόμμα (δεν αρκεί να περικλείεται μια μόνο τιμή σε παρενθέσεις). Άσχημο, αλλά αποτελεσματικό. Για παράδειγμα:



```
>>> empty = ()
>>> singleton = 'hello',      # <-- note trailing comma
>>> len(empty)
0
>>> len(singleton)
1
>>> singleton
('hello',)
```

Η δήλωση `t = 12345, 54321, 'hello!'` είναι ένα παράδειγμα *tuple packing*: οι τιμές 12345, 54321 και 'hello!' είναι συσκευασμένες μαζί σε μια πλειάδα (tuple). Η αντίστροφη λειτουργία είναι επίσης εφικτή:

```
>>> x, y, z = t
```

Αυτό ονομάζεται, αρκετά σωστά, *sequence unpacking* και λειτουργεί για οποιαδήποτε ακολουθία στη δεξιά πλευρά. Το sequence unpacking απαιτεί να υπάρχουν τόσες μεταβλητές στην αριστερή πλευρά του συμβόλου ιότητας όσα στοιχεία υπάρχουν στην ακολουθία. Σημείωση ότι η πολλαπλή ανάθεση είναι στην πραγματικότητα απλώς ένας συνδυασμός tuple packing και sequence unpacking.

## 5.4 Σύνολα (Sets)

Η Python περιλαμβάνει επίσης έναν τύπο δεδομένων για *sets*. Ένα set είναι μια μη ταξινομημένη συλλογή χωρίς διπλότυπα στοιχεία. Οι βασικές χρήσεις περιλαμβάνουν τη δοκιμή ιδιότητας μέλους και την εξάλειψη διπλότυπων εγγραφών. Τα αντικείμενα συνόλου υποστηρίζουν επίσης μαθηματικές πράξεις όπως ένωση, τομή, διαφορά και συμμετρική διαφορά.

Τα άγκιστρα ή η συνάρτηση `set()` μπορούν να χρησιμοποιηθούν για τη δημιουργία συνόλων. Σημείωση: για να δημιουργήσετε ένα κενό σύνολο πρέπει να χρησιμοποιήσετε το `set()`, όχι το `{}` • το τελευταίο δημιουργεί ένα κενό λεξικό, μια δομή δεδομένων που θα συζητήσουμε στην επόμενη ενότητα.

Ακολουθεί μια σύντομη επίδειξη:

```
>>> basket = {'apple', 'orange', 'apple', 'pear', 'orange', 'banana'}
>>> print(basket)                # show that duplicates have been removed
{'orange', 'banana', 'pear', 'apple'}
>>> 'orange' in basket           # fast membership testing
True
>>> 'crabgrass' in basket
False

>>> # Demonstrate set operations on unique letters from two words
...
>>> a = set('abracadabra')
>>> b = set('alacazam')
>>> a                                # unique letters in a
{'a', 'r', 'b', 'c', 'd'}
>>> a - b                            # letters in a but not in b
{'r', 'd', 'b'}
>>> a | b                            # letters in a or b or both
{'a', 'c', 'r', 'd', 'b', 'm', 'z', 'l'}
>>> a & b                            # letters in both a and b
{'a', 'c'}
>>> a ^ b                            # letters in a or b but not both
{'r', 'd', 'b', 'm', 'z', 'l'}
```

Ομοίως με *list comprehensions*, υποστηρίζονται επίσης τα comprehensions των συνόλων:

```
>>> a = {x for x in 'abracadabra' if x not in 'abc'}
>>> a
{'r', 'd'}
```

## 5.5 Λεξικά (Dictionaries)

Another useful data type built into Python is the *dictionary* (see *typesmapping*). Dictionaries are sometimes found in other languages as «associative memories» or «associative arrays». Unlike sequences, which are indexed by a range of numbers, dictionaries are indexed by *keys*, which can be any immutable type; strings and numbers can always be keys. Tuples can be used as keys if they contain only strings, numbers, or tuples; if a tuple contains any mutable object either directly or indirectly, it cannot be used as a key. You can't use lists as keys, since lists can be modified in place using index assignments, slice assignments, or methods like `append()` and `extend()`.

Είναι καλύτερο να σκεφτείτε ένα λεξικό ως ένα σύνολο ζευγών *key: value*, με την προϋπόθεση ότι τα κλειδιά είναι μοναδικά (σε ένα λεξικό). Ένα ζεύγος αγκύλων δημιουργεί ένα κενό λεξικό: `{}`. Η τοποθέτηση μιας λίστας ζευγών *key:value* στο λεξικό, αυτός είναι επίσης ο τρόπος με τον οποίο γράφονται τα λεξικά στην έξοδο.

Οι κύριες λειτουργίες σε ένα λεξικό είναι η αποθήκευση μιας τιμής με κάποιο κλειδί και η εξαγωγή της τιμής που δίνεται στο κλειδί. Είναι επίσης δυνατή η διαγραφή ενός ζεύγους *key:value* με `del`. Εάν αποθηκεύετε χρησιμοποιώντας ένα κλειδί που βρίσκεται ήδη σε χρήση, η παλιά τιμή που σχετίζεται με αυτό το κλειδί έχει ξεχαστεί. Είναι σφάλμα να εξαγάγετε μια τιμή χρησιμοποιώντας ένα ανύπαρκτο κλειδί.

Η εκτέλεση του `list(d)` σε ένα λεξικό επιστρέφει μια λίστα με όλα τα κλειδιά που χρησιμοποιούνται στο λεξικό, με σειρά εισαγωγής (αν θέλετε να ταξινομηθεί, απλώς χρησιμοποιήστε το `sorted(d)`). Για να ελέγξετε εάν υπάρχει ένα μεμονωμένο κλειδί στο λεξικό, χρησιμοποιήστε τη λέξη-κλειδί `in`.

Ακολουθεί ένα μικρό παράδειγμα χρησιμοποιώντας ένα λεξικό:

```
>>> tel = {'jack': 4098, 'sape': 4139}
>>> tel['guido'] = 4127
>>> tel
{'jack': 4098, 'sape': 4139, 'guido': 4127}
>>> tel['jack']
4098
>>> del tel['sape']
>>> tel['irv'] = 4127
>>> tel
{'jack': 4098, 'guido': 4127, 'irv': 4127}
>>> list(tel)
['jack', 'guido', 'irv']
>>> sorted(tel)
['guido', 'irv', 'jack']
>>> 'guido' in tel
True
>>> 'jack' not in tel
False
```

Ο constructor `dict()` δημιουργεί λεξικά απευθείας από ακολουθίες ζευγών *key-value*:

```
>>> dict([('sape', 4139), ('guido', 4127), ('jack', 4098)])
{'sape': 4139, 'guido': 4127, 'jack': 4098}
```

Επιπλέον, τα *comprehensions* λεξικών μπορούν να χρησιμοποιηθούν για τη δημιουργία λεξικών από αυθαίρετες εκφράσεις κλειδιού και τιμών:

```
>>> {x: x**2 for x in (2, 4, 6)}
{2: 4, 4: 16, 6: 36}
```

Όταν τα κλειδιά είναι απλές συμβολοσειρές, μερικές φορές είναι πιο εύκολο να ορίσετε ζεύγη χρησιμοποιώντας ορίσματα λέξεων-κλειδιών:

```
>>> dict(sape=4139, guido=4127, jack=4098)
{'sape': 4139, 'guido': 4127, 'jack': 4098}
```

## 5.6 Τεχνικές Looping

When looping through dictionaries, the key and corresponding value can be retrieved at the same time using the `items()` method.

```
>>> knights = {'gallahad': 'the pure', 'robin': 'the brave'}
>>> for k, v in knights.items():
...     print(k, v)
...
gallahad the pure
robin the brave
```

Κατά το looping μέσω ακολουθίας, ο δείκτης θέσης και η αντίστοιχη τιμή μπορούν να ανακτηθούν ταυτόχρονα χρησιμοποιώντας τη συνάρτηση `enumerate()`.

```
>>> for i, v in enumerate(['tic', 'tac', 'toe']):
...     print(i, v)
...
0 tic
1 tac
2 toe
```

Για να κάνετε loop σε δύο ή περισσότερες ακολουθίες ταυτόχρονα, οι καταχωρίσεις μπορούν να αντιστοιχιστούν με τη συνάρτηση `zip()`.

```
>>> questions = ['name', 'quest', 'favorite color']
>>> answers = ['lancelot', 'the holy grail', 'blue']
>>> for q, a in zip(questions, answers):
...     print('What is your {0}? It is {1}'.format(q, a))
...
What is your name? It is lancelot.
What is your quest? It is the holy grail.
What is your favorite color? It is blue.
```

Για να κάνετε loop σε μια ακολουθία αντίστροφα, καθορίστε πρώτα την ακολουθία προς τα εμπρός και μετά καλέστε τη συνάρτηση `reversed()`.

```
>>> for i in reversed(range(1, 10, 2)):
...     print(i)
...
9
7
5
3
1
```

Για να κάνετε loop σε μια ακολουθία με ταξινομημένη σειρά, χρησιμοποιήστε τη συνάρτηση `sorted()`, η οποία επιστρέφει μια νέα ταξινομημένη λίστα αφήνοντας την πηγή αναλλοίωτη.

```
>>> basket = ['apple', 'orange', 'apple', 'pear', 'orange', 'banana']
>>> for i in sorted(basket):
...     print(i)
...
apple
apple
banana
orange
orange
pear
```

Η χρήση του `set()` σε μια ακολουθία εξαλείφει τα διπλά στοιχεία. Η χρήση του `sorted()` σε συνδυασμό με το `set()` σε μια ακολουθία είναι ένας ιδιωματικός τρόπος για να κάνετε loop πάνω από μοναδικά στοιχεία

της ακολουθίας σε ταξινομημένη σειρά.

```
>>> basket = ['apple', 'orange', 'apple', 'pear', 'orange', 'banana']
>>> for f in sorted(set(basket)):
...     print(f)
...
apple
banana
orange
pear
```

Μερικές φορές είναι δελεαστικό να αλλάζετε μια λίστα ενώ την περιηγείστε· ωστόσο, είναι συχνά πιο απλό και ασφαλές να δημιουργήσετε μια νέα λίστα.

```
>>> import math
>>> raw_data = [56.2, float('NaN'), 51.7, 55.3, 52.5, float('NaN'), 47.8]
>>> filtered_data = []
>>> for value in raw_data:
...     if not math.isnan(value):
...         filtered_data.append(value)
...
>>> filtered_data
[56.2, 51.7, 55.3, 52.5, 47.8]
```

## 5.7 Περισσότερα για τις συνθήκες

Οι συνθήκες που χρησιμοποιούνται στις δηλώσεις `while` και `if` μπορούν να περιέχουν οποιουδήποτε τελεστές, όχι μόνο συγκρίσεις.

Οι τελεστές σύγκρισης `in` και `not in` είναι δοκιμές ιδιότητας μέλους που καθορίζουν εάν μια τιμή βρίσκεται (ή όχι σε) ένα container. Οι τελεστές `is` και `is not` συγκρίνουν εάν δύο αντικείμενα είναι πραγματικά το ίδιο αντικείμενο. Όλοι οι τελεστές σύγκρισης έχουν την ίδια προτεραιότητα, η οποία είναι χαμηλότερη από αυτή όλων των αριθμητικών τελεστών.

Οι συγκρίσεις μπορούν να είναι αλυσιδωτές. Για παράδειγμα, `a < b == c` ελέγχει εάν το `a` είναι μικρότερο από `b` και επιπλέον το `b` ισούται με `c`.

Οι συγκρίσεις μπορούν να συνδυαστούν χρησιμοποιώντας τους λογικούς τελεστές `and` και `or`, και το αποτέλεσμα μιας σύγκρισης (ή οποιασδήποτε άλλης λογικής έκφρασης) μπορεί να ακυρωθεί με `not`. Αυτοί έχουν χαμηλότερες προτεραιότητες μεταξύ των τελεστών σύγκρισης, το `not` έχει την υψηλότερη προτεραιότητα και το `or` τη χαμηλότερη, έτσι ώστε το `A and not B or C` ισοδυναμεί με `(A and (not B)) or C`. Όπως πάντα, οι παρενθέσεις μπορούν να χρησιμοποιηθούν για να εκφράσουν την επιθυμητή σύνθεση.

Οι λογικοί τελεστές `and` και `or` είναι οι λεγόμενοι τελεστές *short-circuit*: τα ορίσματα τους αξιολογούνται από αριστερά προς τα δεξιά και η αξιολόγηση σταματά μόλις καθοριστεί το αποτέλεσμα. Για παράδειγμα, εάν το `A and C` είναι αληθές, αλλά το `B` είναι ψευδές, το `A and B and C` δεν αξιολογεί την έκφραση `C`. Όταν χρησιμοποιείται ως γενική τιμή και όχι ως λογική, η τιμή επιστροφής ενός *short-circuit* τελεστή είναι το τελευταίο αξιολογημένο όρισμα.

Είναι δυνατό να αντιστοιχίσετε το αποτέλεσμα μιας σύγκρισης ή άλλη δυαδική έκφραση σε μια μεταβλητή. Για παράδειγμα,

```
>>> string1, string2, string3 = '', 'Trondheim', 'Hammer Dance'
>>> non_null = string1 or string2 or string3
>>> non_null
'Trondheim'
```

Σημειώστε ότι στην Python, σε αντίθεση με την C, η ανάθεση εντός των εκφράσεων πρέπει να γίνεται ρητά με τον τελεστή walrus operator `:`. Αυτό αποφεύγει μια κοινή κατηγορία προβλημάτων που συναντώνται στα προγράμματα C: πληκτρολογώντας `=` σε μια έκφραση όταν προοριζόταν το `==`.

## 5.8 Σύγκριση ακολουθιών και άλλων τύπων

Τα αντικείμενα ακολουθίας μπορούν συνήθως να συγκριθούν με άλλα αντικείμενα με τον ίδιο τύπο ακολουθίας. Η σύγκριση χρησιμοποιεί *lexicographical* σειρά: πρώτα συγκρίνονται τα δύο πρώτα στοιχεία και αν διαφέρουν αυτό καθορίζει το αποτέλεσμα της σύγκρισης· εάν είναι ίσα, τα επόμενα δύο στοιχεία συγκρίνονται και ούτω καθεξής, έως ότου εξαντληθεί η μία από τις δύο ακολουθίες. Εάν δύο στοιχεία προς σύγκριση είναι τα ίδια ακολουθίες του ίδιου τύπου, η λεξικογραφική σύγκριση πραγματοποιείται αναδρομικά. Εάν όλα τα στοιχεία δύο ακολουθιών συγκρίνονται ίσα, οι ακολουθίες θεωρούνται ίσες. Εάν η μια ακολουθία είναι αρχική υποακολουθία της άλλης, η μικρότερη ακολουθία είναι η μικρότερη (ελάχιστη). Η λεξικογραφική ταξινόμηση συμβολοσειρών χρησιμοποιεί τον αριθμό κωδικού σημείου Unicode για να ταξινομήσει μεμονωμένους χαρακτήρες. Μερικά παραδείγματα συγκρίσεων μεταξύ ακολουθιών του ίδιου τύπου:

```
(1, 2, 3) < (1, 2, 4)
[1, 2, 3] < [1, 2, 4]
'ABC' < 'C' < 'Pascal' < 'Python'
(1, 2, 3, 4) < (1, 2, 4)
(1, 2) < (1, 2, -1)
(1, 2, 3) == (1.0, 2.0, 3.0)
(1, 2, ('aa', 'ab')) < (1, 2, ('abc', 'a'), 4)
```

Λάβετε υπόψη ότι η σύγκριση αντικειμένων διαφορετικών τύπων με `<` ή `>` είναι νόμιμη υπό τον όρο ότι τα αντικείμενα διαθέτουν κατάλληλες μεθόδους σύγκρισης. Για παράδειγμα, οι μικτές αριθμητικοί τύποι συγκρίνονται σύμφωνα με την αριθμητική τους τιμή, οπότε το 0 ισούται με 0.0, κλπ. Διαφορετικά, αντί να παρέχει μια αυθαίρετη παραγγελία, ο διερμηνέας θα κάνει `raise` μια εξαίρεση `TypeError`.

### Υποσημειώσεις



Modules

---

Εάν βγείτε από τον interpreter της Python και μπειτέ ξανά, οι ορισμοί που έχετε κάνει (συναρτήσεις και μεταβλητές) χάνονται. Επομένως, εάν θέλετε να γράψετε ένα κάπως μεγαλύτερο πρόγραμμα, είναι προτιμότερο να χρησιμοποιήσετε έναν επεξεργαστή κειμένου για να προετοιμάσετε την εισαγωγή για τον interpreter και την εκτέλεση του με αυτό το αρχείο ως input. Αυτό είναι γνωστό ως δημιουργία *script*. Καθώς το πρόγραμμα σας μεγαλώνει, μπορεί να θέλετε να το χωρίσετε σε πολλά αρχεία για ευκολότερη συντήρηση. Μπορεί επίσης να θέλετε να χρησιμοποιήσετε μια εύχρηστη συνάρτηση που έχετε γράψει σε πολλά προγράμματα χωρίς να αντιγράψετε τον ορισμό της σε κάθε πρόγραμμα.

Για να το υποστηρίξει αυτό, η Python έχει έναν τρόπο να βάζει ορισμούς σε ένα αρχείο και να τους χρησιμοποιεί σε ένα script ή σε ένα διαδραστικό instance του interpreter. Ένα τέτοιο αρχείο ονομάζεται *module*\*. *ορισμοί από μια ενότητα μπορούν να \*εισαχθούν σε άλλα modules ή στο κύριο module (η συλλογή των μεταβλητών στις οποίες έχετε πρόσβαση σε ένα script που εκτελείται στον ανώτερο επίπεδο και σε λειτουργία αριθμομηχανής).*

Ένα module είναι ένα αρχείο που περιέχει ορισμούς και δηλώσεις Python. Το όνομα αρχείου είναι το όνομα του module με το επίθημα `.py`. Μέσα σε ένα module, το όνομα του module (ως συμβολοσειρά) είναι διαθέσιμο ως τιμή της global μεταβλητής `__name__`. Για παράδειγμα, χρησιμοποιήστε το αγαπημένο σας πρόγραμμα επεξεργασίας κειμένου για να δημιουργήσετε ένα αρχείο που ονομάζεται `fibonacci.py` στον τρέχοντα κατάλογο με τα ακόλουθα περιεχόμενα:

```
# Fibonacci numbers module

def fib(n):    # write Fibonacci series up to n
    a, b = 0, 1
    while a < n:
        print(a, end=' ')
        a, b = b, a+b
    print()

def fib2(n):   # return Fibonacci series up to n
    result = []
    a, b = 0, 1
    while a < n:
        result.append(a)
        a, b = b, a+b
    return result
```

Τώρα εισάγετε τον Python interpreter και εισάγετε αυτό το module με την ακόλουθη εντολή:

```
>>> import fibo
```

Αυτό δεν προσθέτει τα ονόματα των συναρτήσεων που ορίζονται στο `fibo` απευθείας στον τρέχοντα *namespace* (βλ. *Εμβέλεια και Πεδία Ονομάτων στην Python* για περισσότερες λεπτομέρειες): προσθέτει μόνο το όνομα του module `fibo` εκεί. Χρησιμοποιώντας το όνομα του module μπορείτε να αποκτήσετε πρόσβαση στις λειτουργίες:

```
>>> fibo.fib(1000)
0 1 1 2 3 5 8 13 21 34 55 89 144 233 377 610 987
>>> fibo.fib2(100)
[0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89]
>>> fibo.__name__
'fibo'
```

Εάν σκοπεύετε να χρησιμοποιείτε συχνά μια συνάρτηση, μπορείτε να την αντιστοιχίσετε σε ένα τοπικό όνομα:

```
>>> fib = fibo.fib
>>> fib(500)
0 1 1 2 3 5 8 13 21 34 55 89 144 233 377
```

## 6.1 Περισσότερα για τα Modules

Ένα module μπορεί να περιέχει εκτελέσιμες δηλώσεις καθώς και ορισμούς συναρτήσεων. Αυτές οι δηλώσεις προορίζονται για την προετοιμασία του module. Εκτελούνται μόνο την *πρώτη* φορά που εμφανίζεται το όνομα του module σε μια δήλωση εισαγωγής.<sup>1</sup> (Εκτελούνται επίσης εάν το αρχείο εκτελείται ως script.)

Κάθε module έχει τον δικό της ιδιωτικό namespace, ο οποίος χρησιμοποιείται ως global namespace από όλες τις συναρτήσεις που ορίζονται στο module. Έτσι, ο συντάκτης μιας ενότητας μπορεί να χρησιμοποιήσει global μεταβλητές στο module χωρίς να ανησυχεί για τυχαία conflicts με τις global μεταβλητές του χρήστη. Από την άλλη πλευρά, εάν ξέρετε τι κάνετε, μπορείτε να αγγίξετε τις global μεταβλητές ενός module με το ίδιο notation που χρησιμοποιείται για να αναφέρεται στις συναρτήσεις, `modname.itemname`.

Τα modules μπορούν να εισάγουν άλλα modules. Είναι σύνηθες, αλλά δεν απαιτείται να τοποθετούνται όλες οι δηλώσεις `import` στην αρχή μιας ενότητας (ή σεναρίου, για αυτό το θέμα). Τα ονόματα των modules που εισάγονται, εάν τοποθετούνται στο ανώτερο επίπεδο του ένα module (εκτός οποιωνδήποτε συναρτήσεων ή κλάσεων), προστίθενται στον global namespace του module.

Υπάρχει μια παραλλαγή της δήλωσης `import` που εισάγει ονόματα από ένα module απευθείας στον χώρο στα `importing module's namespace`. Για παράδειγμα:

```
>>> from fibo import fib, fib2
>>> fib(500)
0 1 1 2 3 5 8 13 21 34 55 89 144 233 377
```

Αυτό δεν εισάγει το όνομα ενός module από το οποίο λαμβάνονται οι εισαγωγές στο τοπικό namespace (αρά στο παράδειγμα, το `fibo` δεν ορίζεται).

Υπάρχει ακόμη και μια παραλλαγή για την εισαγωγή όλων των ονομάτων που ορίζει μια ενότητα:

```
>>> from fibo import *
>>> fib(500)
0 1 1 2 3 5 8 13 21 34 55 89 144 233 377
```

Αυτό εισάγει όλα τα ονόματα εκτός από αυτά που ξεκινούν με κάτω παύλα (`_`). Στις περισσότερες περιπτώσεις, οι προγραμματιστές Python δεν χρησιμοποιούν αυτήν την δυνατότητα, καθώς εισάγει ένα άγνωστο σύνολο ονομάτων στον interpreter, κρύβοντας πιθανώς κάποια πράγματα που έχετε ήδη ορίσει.

<sup>1</sup> Στην πραγματικότητα, οι ορισμοί συναρτήσεων είναι επίσης “statements” που “εκτελούνται”· η εκτέλεση ενός ορισμού συνάρτησης σε επίπεδο module προσθέτει το όνομα της συνάρτησης στον καθολικό namespace του module.



Λάβετε υπόψη ότι γενικά η πρακτική της εισαγωγής \* από ένα module ή ένα πακέτο αποδοκιμάζεται, καθώς προκαλεί συχνά κακώς αναγνώσιμο κώδικα. Ωστόσο, είναι εντάξει να τον χρησιμοποιήσετε για να αποθηκεύσετε την πληκτρολόγηση σε διαδραστικές περιόδους σύνδεσης.

Εάν το όνομα του module ακολουθείται από `as`, τότε το όνομα που ακολουθεί `as` συνδέεται απευθείας με το εισαγόμενο module.

```
>>> import fibo as fib
>>> fib.fib(500)
0 1 1 2 3 5 8 13 21 34 55 89 144 233 377
```

Αυτό ουσιαστικά εισάγει το module με τον ίδιο τρόπο που θα κάνει το `import fibo`, με τη μόνη διαφορά ότι είναι διαθέσιμο ως `fib`.

Μπορεί επίσης να χρησιμοποιηθεί όταν χρησιμοποιείτε `from` με παρόμοια εφέ:

```
>>> from fibo import fib as fibonacci
>>> fibonacci(500)
0 1 1 2 3 5 8 13 21 34 55 89 144 233 377
```

**Σημείωση:** Για λόγους αποτελεσματικότητας, κάθε module εισάγεται μόνο μία φορά ανά περίοδο λειτουργίας του interpreter. Επομένως, εάν αλλάξετε τα modules σας, πρέπει να επανεκκινήσετε τον διερμηνέα – ή, εάν είναι μόνο ένα module που θέλετε να δοκιμάσετε διαδραστικά, χρησιμοποιήστε το `importlib.reload()`, π.χ. `import importlib; importlib.reload(modulename)`.

## 6.1.1 Εκτέλεση modules ως scripts

Όταν εκτελείτε ένα Python module με:

```
python fibo.py <arguments>
```

ο κώδικας στο module θα εκτελεστεί, ακριβώς σαν να τον εισαγάγετε, αλλά με το `name` να έχει οριστεί σε `"__main__"`. Αυτό σημαίνει ότι προσθέτοντας αυτόν τον κώδικα στο τέλος του module σας:

```
if __name__ == "__main__":
    import sys
    fib(int(sys.argv[1]))
```

μπορείτε να κάνετε το αρχείο χρησιμοποιήσιμο ως script καθώς και ως module που μπορεί να εισαχθεί, επειδή ο κώδικας που αναλύει την γραμμή εντολών εκτελείται μόνο εάν το module εκτελείται ως το «main» αρχείο:

```
$ python fibo.py 50
0 1 1 2 3 5 8 13 21 34
```

Εάν το module έχει εισαχθεί, ο κώδικας δεν εκτελείται:

```
>>> import fibo
>>>
```

Αυτό χρησιμοποιείται συχνά είτε για την παροχή ενός βολικού user interface σε ένα module, είτε για σκοπούς δοκιμής (η εκτέλεση του module ως script εκτελεί μια δοκιμαστική σουίτα).

## 6.1.2 To Search Path του Module

When a module named `spam` is imported, the interpreter first searches for a built-in module with that name. These module names are listed in `sys.builtin_module_names`. If not found, it then searches for a file named `spam.py` in a list of directories given by the variable `sys.path`. `sys.path` is initialized from these locations:

- Ο κατάλογος που περιέχει το `input script` (ή τον τρέχοντα κατάλογο όταν δεν έχει καθοριστεί αρχείο).
- `PYTHONPATH` (μια λίστα ονομάτων καταλόγου, με την ίδια σύνταξη με τη μεταβλητή του shell `PATH`).
- Η προεπιλογή που εξαρτάται από την εγκατάσταση (κατά σύμβαση, συμπεριλαμβανομένου ενός καταλόγου `site-packages`, που χειρίζεται το `module site`).

---

**Σημείωση:** Στα συστήματα αρχείων που υποστηρίζουν `symlinks`, ο κατάλογος που περιέχει το `input script` υπολογίζεται αφού ακολουθηθεί το `symlink`. Με άλλα λόγια, ο κατάλογος που περιέχει το `symlink` **δεν** προστίθεται στη διαδρομή αναζήτησης του `module`.

---

Μετά την προετοιμασία, τα προγράμματα Python μπορούν να τροποποιήσουν το `sys.path`. Ο κατάλογος που περιέχει το `script` που εκτελείται τοποθετείται στην αρχή της διαδρομής αναζήτησης, μπροστά από την τυπική διαδρομή της βιβλιοθήκης. Αυτό σημαίνει ότι τα `scripts` σε αυτόν τον κατάλογο θα είναι φορτωμένα αντί για τα `modules` με το ίδιο όνομα στον κατάλογο μιας βιβλιοθήκης. Αυτό είναι ένα σφάλμα, εκτός εάν προορίζεται η αντικατάσταση. Βλ. την ενότητα [Standard Modules](#) για περισσότερες πληροφορίες.

## 6.1.3 «Compiled» Python αρχεία

Για να επιταχύνει τη φόρτωση `modules`, η Python κάνει `cache` την `compiled` έκδοση κάθε `module` στον κατάλογο `__pycache__` κάτω από το όνομα `module.version.pyc`, όπου η έκδοση κωδικοποιεί τη μορφή του `compiled` αρχείου· γενικά περιέχει τον αριθμό έκδοσης της Python. Για παράδειγμα, στην έκδοση CPython 3.3 η `compiled` έκδοση του `spam.py` θα αποθηκευτεί ως `__pycache__/spam.cpython-33.pyc`. Αυτή η σύμβαση ονομασίας, επιτρέπει σε `compiled` `modules` από διαφορετικές εκδόσεις και διαφορετικές εκδόσεις της Python να συνυπάρχουν.

Η Python ελέγχει την ημερομηνία τροποποίησης του πηγαίου έναντι της `compiled` έκδοσης για να δει εάν είναι ξεπερασμένη και χρειάζεται να γίνει `compile` ξανά. Αυτή είναι μια εντελώς αυτόματη διαδικασία. Επίσης, τα `compiled` `modules` είναι ανεξάρτητες από πλατφόρμα, επομένως η ίδια βιβλιοθήκη μπορεί να κοινοποιηθεί ανάμεσα σε συστήματα με διαφορετικές αρχιτεκτονικές.

Η Python δεν ελέγχει την `cache` σε δύο περιπτώσεις. Πρώτον, πάντα κάνει `compile` ξανά και δεν αποθηκεύει το αποτέλεσμα για το `module` που φορτώνεται απευθείας από τη γραμμή εντολών. Δεύτερον, δεν ελέγχει τη μνήμη `cache` εάν δεν υπάρχει το `source` `module`. Για να υποστηρίξετε μια διανομή χωρίς πηγαίο (`compiled` μόνο), το `compiled` `module` πρέπει να βρίσκεται στον `source` κατάλογο και δεν πρέπει να υπάρχει `source` `module`.

Μερικές συμβουλές για ειδικούς:

- Μπορείτε να χρησιμοποιήσετε τους `switches` `-O` ή `-OO` στην εντολή Python για να μειώσετε το μέγεθος ενός `compiled` `module`. Το `-O` `switch` αφαιρεί τις `assert` `statements`, το `-OO` `switch` αφαιρεί τόσο τα `assert` `statements` όσο και τις `__doc__` συμβολοσειρές. Εφόσον ορισμένα προγράμματα μπορεί να βασίζονται στην ύπαρξη αυτών των διαθέσιμων, θα πρέπει να χρησιμοποιήσετε αυτήν την επιλογή μόνο εάν γνωρίζετε τι κάνετε. «Optimized» `modules` έχουν ένα `opt-` `tag` και είναι συνήθως μικρότερες. Οι μελλοντικές εκδόσεις μπορεί να αλλάξουν τα αποτελέσματα της βελτιστοποίησης.
- Ένα πρόγραμμα δεν εκτελείται πιο γρήγορα όταν διαβάζεται από ένα αρχείο `.pyc` από ό,τι όταν διαβάζεται από ένα αρχείο `.py`· το μόνο πράγμα που είναι πιο γρήγορο από τα αρχεία `.pyc` είναι η ταχύτητα με την οποία φορτώνονται.
- Το `module` `compileall` μπορεί να δημιουργήσει αρχεία `.pyc` για όλα τα `modules` σε ένα κατάλογο.
- Υπάρχουν περισσότερες λεπτομέρειες σχετικά με αυτή τη διαδικασία, συμπεριλαμβανομένου ενός διαγράμματος ροής των αποφάσεων, στο [PEP 3147](#).

## 6.2 Standard Modules

Η Python συνοδεύεται από μια βιβλιοθήκη standard modules, η οποία περιγράφεται σε ένα ξεχωριστό έγγραφο, την Αναφορά Βιβλιοθήκης Python («Library Reference» hereafter). Ορισμένα modules είναι ενσωματωμένα στον interpreter· αυτές παρέχουν πρόσβαση σε λειτουργίες που δεν αποτελούν μέρος του πυρήνα της γλώσσας, αλλά εντούτοις είναι ενσωματωμένα, είτε για αποτελεσματικότητα είτε για την παροχή πρόσβασης σε πρωτόγονα στοιχεία του λειτουργικού συστήματος όπως οι κλήσεις συστήματος. Το σύνολο τέτοιων modules είναι μια επιλογή διαμόρφωσης που εξαρτάται επίσης από την υποκείμενη πλατφόρμα. Για παράδειγμα, το module `winreg` παρέχεται μόνο σε συστήματα Windows. Ένα συγκεκριμένο module που αξίζει κάποια προσοχή είναι το `sys`, το οποίο είναι ενσωματωμένο στον interpreter της Python. Οι μεταβλητές `sys.ps1` και `sys.ps2` ορίζουν τις συμβολοσειρές που χρησιμοποιούνται ως κύρια και δευτερεύοντα prompts:

```
>>> import sys
>>> sys.ps1
'>>> '
>>> sys.ps2
'... '
>>> sys.ps1 = 'C> '
C> print('Yuck!')
Yuck!
C>
```

Αυτές οι δύο μεταβλητές ορίζονται μόνο εάν ο interpreter βρίσκεται σε διαδραστική λειτουργία.

Η μεταβλητή `sys.path` είναι μια λίστα συμβολοσειρών που καθορίζει τη διαδρομή αναζήτησης του διεργασιών για modules. Αρχικοποιείται σε μια προεπιλεγμένη διαδρομή που λαμβάνεται από τη μεταβλητή περιβάλλοντος `PYTHONPATH`, ή από μια ενσωματωμένη προεπιλογή εάν το `PYTHONPATH` δεν έχει οριστεί. Μπορείτε να το τροποποιήσετε χρησιμοποιώντας τυπικές λειτουργίες λίστας:

```
>>> import sys
>>> sys.path.append('/ufs/guido/lib/python')
```

## 6.3 Η συνάρτηση `dir()`

Η ενσωματωμένη συνάρτηση `dir()` χρησιμοποιείται για να ανακαλύψει ποια ονόματα ορίζει ένα module. Επιστρέφει μια ταξινομημένη λίστα συμβολοσειρών:

```
>>> import fibo, sys
>>> dir(fibo)
['__name__', 'fibo', 'fib2']
>>> dir(sys)
['__breakpointhook__', '__displayhook__', '__doc__', '__excepthook__',
 '__interactivehook__', '__loader__', '__name__', '__package__', '__spec__',
 '__stderr__', '__stdin__', '__stdout__', '__unraisablehook__',
 '__clear_type_cache__', '__current_frames__', '__debugmallocstats__', '__framework',
 '__getframe__', '__git__', '__home__', '__xoptions__', 'abiflags', 'addaudithook',
 'api_version', 'argv', 'audit', 'base_exec_prefix', 'base_prefix',
 'breakpointhook', 'builtin_module_names', 'byteorder', 'call_tracing',
 'callstats', 'copyright', 'displayhook', 'dont_write_bytecode', 'exc_info',
 'excepthook', 'exec_prefix', 'executable', 'exit', 'flags', 'float_info',
 'float_repr_style', 'get_asyncgen_hooks', 'get_coroutine_origin_tracking_depth',
 'getallocatedblocks', 'getdefaultencoding', 'getdlopenflags',
 'getfilesystemencodeerrors', 'getfilesystemencoding', 'getprofile',
 'getrecursionlimit', 'getrefcount', 'getsizeof', 'getswitchinterval',
 'gettrace', 'hash_info', 'hexversion', 'implementation', 'int_info',
 'intern', 'is_finalizing', 'last_traceback', 'last_type', 'last_value',
 'maxsize', 'maxunicode', 'meta_path', 'modules', 'path', 'path_hooks',
 'path_importer_cache', 'platform', 'prefix', 'ps1', 'ps2', 'pycache_prefix',
```

(συνέχεια στην επόμενη σελίδα)

(συνεχίζεται από την προηγούμενη σελίδα)

```
'set_asyncgen_hooks', 'set_coroutine_origin_tracking_depth', 'setfdopenflags',
'setprofile', 'setrecursionlimit', 'setswitchinterval', 'settrace', 'stderr',
'stdin', 'stdout', 'thread_info', 'unraisablehook', 'version', 'version_info',
'warnoptions']
```

Χωρίς ορίσματα, η `dir()` παραθέτει τα ονόματα που έχετε ορίσει αυτήν τη στιγμή:

```
>>> a = [1, 2, 3, 4, 5]
>>> import fibo
>>> fib = fibo.fib
>>> dir()
['__builtins__', '__name__', 'a', 'fib', 'fibo', 'sys']
```

Λάβετε υπόψη ότι παραθέτει όλους τους τύπους ονομάτων: μεταβλητές, modules, συναρτήσεις, κ.λπ.

Η `dir()` δεν παραθέτει τα ονόματα των ενσωματωμένων συναρτήσεων και μεταβλητών. Εάν θέλετε μια λίστα από αυτές, ορίζονται στην τυπική ενότητα `builtins`:

```
>>> import builtins
>>> dir(builtins)
['ArithmeticError', 'AssertionError', 'AttributeError', 'BaseException',
'BlockingIOError', 'BrokenPipeError', 'BufferError', 'BytesWarning',
'ChildProcessError', 'ConnectionAbortedError', 'ConnectionError',
'ConnectionRefusedError', 'ConnectionResetError', 'DeprecationWarning',
'EOFError', 'Ellipsis', 'EnvironmentError', 'Exception', 'False',
'FileExistsError', 'FileNotFoundError', 'FloatingPointError',
'FutureWarning', 'GeneratorExit', 'IOError', 'ImportError',
'ImportWarning', 'IndentationError', 'IndexError', 'InterruptedError',
'IsADirectoryError', 'KeyError', 'KeyboardInterrupt', 'LookupError',
'MemoryError', 'NameError', 'None', 'NotADirectoryError', 'NotImplemented',
'NotImplementedError', 'OSError', 'OverflowError',
'PendingDeprecationWarning', 'PermissionError', 'ProcessLookupError',
'ReferenceError', 'ResourceWarning', 'RuntimeError', 'RuntimeWarning',
'StopIteration', 'SyntaxError', 'SyntaxWarning', 'SystemError',
'SystemExit', 'TabError', 'TimeoutError', 'True', 'TypeError',
'UnboundLocalError', 'UnicodeDecodeError', 'UnicodeEncodeError',
'UnicodeError', 'UnicodeTranslateError', 'UnicodeWarning', 'UserWarning',
'ValueError', 'Warning', 'ZeroDivisionError', '_', '__build_class__',
'__debug__', '__doc__', '__import__', '__name__', '__package__', 'abs',
'all', 'any', 'ascii', 'bin', 'bool', 'bytearray', 'bytes', 'callable',
'chr', 'classmethod', 'compile', 'complex', 'copyright', 'credits',
'delattr', 'dict', 'dir', 'divmod', 'enumerate', 'eval', 'exec', 'exit',
'filter', 'float', 'format', 'frozenset', 'getattr', 'globals', 'hasattr',
'hash', 'help', 'hex', 'id', 'input', 'int', 'isinstance', 'issubclass',
'iter', 'len', 'license', 'list', 'locals', 'map', 'max', 'memoryview',
'min', 'next', 'object', 'oct', 'open', 'ord', 'pow', 'print', 'property',
'quit', 'range', 'repr', 'reversed', 'round', 'set', 'setattr', 'slice',
'sorted', 'staticmethod', 'str', 'sum', 'super', 'tuple', 'type', 'vars',
'zip']
```

## 6.4 Πακέτα

Packages are a way of structuring Python's module namespace by using «dotted module names». For example, the module name `A.B` designates a submodule named `B` in a package named `A`. Just like the use of modules saves the authors of different modules from having to worry about each other's global variable names, the use of dotted module names saves the authors of multi-module packages like NumPy or Pillow from having to worry about each other's module names.

Ας υποθέσουμε ότι θέλετε να σχεδιάσετε μια συλλογή από module (ένα «πακέτο») για τον ομοιόμορφο χειρισμό αρχείων ήχου και δεδομένων ήχου. Υπάρχουν πολλές διαφορετικές μορφές αρχείων ήχου (που συνήθως αναγνωρίζονται από την επέκτασή τους, για παράδειγμα: `.wav`, `.aiff`, `.au`), επομένως μπορεί να χρειαστεί να δημιουργήσετε και να διατηρήσετε μια αυξανόμενη συλλογή λειτουργιών για τη μετατροπή μεταξύ των διαφόρων μορφών αρχείων. Υπάρχουν επίσης πολλές διαφορετικές λειτουργίες που μπορεί να θέλετε να εκτελέσετε σε δεδομένα ήχου (όπως μίξη, προσθήκη ηχούς, εφαρμογή μιας λειτουργίας ισοσταθμιστή, δημιουργία τεχνητού στερεοφωνικού εφέ), επομένως επιπλέον θα γράφετε μια ατελείωτη ροή από modules για να εκτελέσετε αυτές τις λειτουργίες. Ακολουθεί μια πιθανή δομή για το πακέτο σας (που εκφράζεται ως ιεραρχικό σύστημα αρχείων):

<code>sound/</code>	Top-level package
<code>__init__.py</code>	Initialize the sound package
<code>formats/</code>	Subpackage for file format conversions
<code>__init__.py</code>	
<code>wavread.py</code>	
<code>wavwrite.py</code>	
<code>aiffread.py</code>	
<code>aiffwrite.py</code>	
<code>auread.py</code>	
<code>auwrite.py</code>	
<code>...</code>	
<code>effects/</code>	Subpackage for sound effects
<code>__init__.py</code>	
<code>echo.py</code>	
<code>surround.py</code>	
<code>reverse.py</code>	
<code>...</code>	
<code>filters/</code>	Subpackage for filters
<code>__init__.py</code>	
<code>equalizer.py</code>	
<code>vocoder.py</code>	
<code>karaoke.py</code>	
<code>...</code>	

Κατά την εισαγωγή του πακέτου, η Python πραγματοποιεί αναζήτηση στους καταλόγους στο `sys.path` αναζητώντας τον υποκατάλογο του πακέτου.

The `__init__.py` files are required to make Python treat directories containing the file as packages. This prevents directories with a common name, such as `string`, from unintentionally hiding valid modules that occur later on the module search path. In the simplest case, `__init__.py` can just be an empty file, but it can also execute initialization code for the package or set the `__all__` variable, described later.

Οι χρήστες του πακέτου μπορούν να εισάγουν μεμονωμένα module από το πακέτο, για παράδειγμα:

```
import sound.effects.echo
```

This loads the submodule `sound.effects.echo`. It must be referenced with its full name.

```
sound.effects.echo.echofilter(input, output, delay=0.7, atten=4)
```

Ένα εναλλακτικός τρόπος για την εισαγωγή του submodule είναι:

```
from sound.effects import echo
```

This also loads the submodule `echo`, and makes it available without its package prefix, so it can be used as follows:

```
echo.echofilter(input, output, delay=0.7, atten=4)
```

Μια άλλη παραλλαγή είναι η απευθείας εισαγωγή της επιθυμητής συνάρτησης ή μεταβλητής:

```
from sound.effects.echo import echofilter
```

Again, this loads the submodule `echo`, but this makes its function `echofilter()` directly available:

```
echofilter(input, output, delay=0.7, atten=4)
```

Λάβετε υπόψη ότι όταν χρησιμοποιείτε `from package import item`, το στοιχείο μπορεί να είναι είτε submodule (ή υποπακέτο) του πακέτου ή κάποιο άλλο όνομα που ορίζεται στο πακέτο, όπως μια συνάρτηση, κλάση ή μεταβλητή. Η `import` δήλωση ελέγχει πρώτα εάν το στοιχείο έχει οριστεί στο πακέτο, εάν όχι, υποθέτει ότι είναι ένα module και επιχειρεί να το φορτώσει, αν δεν το βρει δημιουργεί η εξαίρεση `ImportError`.

Αντίθετα, όταν χρησιμοποιείται σύνταξη όπως `import item.subitem.subsubitem`, κάθε στοιχείο εκτός από αυτό το τελευταίο πρέπει να είναι πακέτο• το τελευταίο στοιχείο μπορεί να είναι ένα module ή ένα πακέτο αλλά δεν μπορεί να είναι μια κλάση ή συνάρτηση ή μεταβλητή που ορίζεται από προηγούμενο στοιχείο.

### 6.4.1 Εισάγοντας \* από ένα Πακέτο

Τώρα τι συμβαίνει όταν ο χρήστης γράφει `from sound.effects import *`; Ιδανικά, θα ήλπιζε κανείς ότι αυτό θα βγει με κάποιο τρόπο στο σύστημα αρχείων, θα βρει κάποια submodules που υπάρχουν το πακέτο, και θα τα εισάγει όλα σε αυτό. Αυτό θα μπορούσε να πάρει πολύ χρόνο και η εισαγωγή submodules μπορεί να έχει ανεπιθύμητες παρενέργειες που θα έπρεπε να συμβούν όταν το submodule εισάγεται ρητά.

Η μόνη λύση είναι να παρέχει ο συντάκτης του πακέτου ένα ρητό ευρετήριο του πακέτου. Η δήλωση `import` χρησιμοποιεί την ακόλουθη σύμβαση: εάν ο κώδικας `__init__.py` του πακέτου ορίζει μια λίστα με το όνομα `__all__`, θεωρείται ότι είναι η λίστα με τα ονόματα των modules που θα πρέπει να εισαχθούν όταν συναντήσετε `from package import *`. Είναι στην διακριτή ευχέρεια του συντάκτη του πακέτου να διατηρεί αυτή τη λίστα ενημερωμένη, όταν κυκλοφορήσει μια νέα έκδοση του πακέτου. Οι συντάκτες του πακέτου ενδέχεται επίσης να αποφασίσουν να μην το υποστηρίξουν, εάν δεν βλέπουν ότι χρησιμοποιείται η εισαγωγή του `*` από το πακέτο τους. Για παράδειγμα το αρχείο `sound/effects/__init__.py` θα μπορούσε να περιέχει τον ακόλουθο κώδικα:

```
__all__ = ["echo", "surround", "reverse"]
```

This would mean that `from sound.effects import *` would import the three named submodules of the `sound.effects` package.

If `__all__` is not defined, the statement `from sound.effects import *` does *not* import all submodules from the package `sound.effects` into the current namespace; it only ensures that the package `sound.effects` has been imported (possibly running any initialization code in `__init__.py`) and then imports whatever names are defined in the package. This includes any names defined (and submodules explicitly loaded) by `__init__.py`. It also includes any submodules of the package that were explicitly loaded by previous `import` statements. Consider this code:

```
import sound.effects.echo
import sound.effects.surround
from sound.effects import *
```

In this example, the `echo` and `surround` modules are imported in the current namespace because they are defined in the `sound.effects` package when the `from...import` statement is executed. (This also works when `__all__` is defined.)

Αν και ορισμένα modules έχουν σχεδιαστεί για να εξάγουν μόνο ονόματα που ακολουθούν ορισμένα μοτίβα όταν χρησιμοποιείται το `import *`, εξακολουθεί να θεωρείται κακή πρακτική στον κώδικα παραγωγής.

Θυμηθείτε, δεν υπάρχει τίποτα κακό με τη χρήση του `from package import specific_submodule`! Στην πραγματικότητα, αυτή είναι η προτεινόμενη σημείωση, εκτός εάν το module εισαγωγής χρειάζεται να χρησιμοποιήσει submodules με το ίδιο όνομα από διαφορετικά πακέτα.

### 6.4.2 Intra-package αναφορές

When packages are structured into subpackages (as with the `sound` package in the example), you can use absolute imports to refer to submodules of siblings packages. For example, if the module `sound.filters.vocoder` needs to use the `echo` module in the `sound.effects` package, it can use `from sound.effects import echo`.

You can also write relative imports, with the `from module import name` form of import statement. These imports use leading dots to indicate the current and parent packages involved in the relative import. From the `surround` module for example, you might use:

```
from . import echo
from .. import formats
from ..filters import equalizer
```

Λάβετε υπόψη ότι οι σχετικές εισαγωγές βασίζονται στο όνομα του τρέχοντος module. Επειδή το όνομα του κύριου module είναι πάντα `"__main__"`, τα modules που προορίζονται για χρήση ως κύριο module μιας εφαρμογής Python πρέπει πάντα να χρησιμοποιούν απόλυτες εισαγωγές.

### 6.4.3 Πακέτα σε Πολλαπλούς Καταλόγους

Packages support one more special attribute, `__path__`. This is initialized to be a list containing the name of the directory holding the package's `__init__.py` before the code in that file is executed. This variable can be modified; doing so affects future searches for modules and subpackages contained in the package.

Ενώ αυτή η δυνατότητα δεν χρειάζεται συχνά, μπορεί να χρησιμοποιηθεί για την επέκταση του συνόλου των modules που βρίσκονται σε ένα πακέτο.

### Υποσημειώσεις





Είσοδος και Έξοδος

---

Υπάρχουν διάφοροι τρόποι για να παρουσιάσετε τα αποτελέσματα ενός προγράμματος: τα δεδομένα μπορούν να εκτυπωθούν σε μορφή αναγνώσιμη από τον άνθρωπο ή να εγγραφούν σε ένα αρχείο για μελλοντική χρήση. Αυτό το κεφάλαιο θα συζητήσει μερικές από τις δυνατότητες.

## 7.1 Ομορφότερη Μορφοποίηση Εξόδου

So far we've encountered two ways of writing values: *expression statements* and the `print()` function. (A third way is using the `write()` method of file objects; the standard output file can be referenced as `sys.stdout`. See the Library Reference for more information on this.)

Συχνά θα θέλετε περισσότερο έλεγχο στη μορφοποίηση της εξόδου σας παρά απλώς να εκτυπώνετε τιμές διαχωρισμένες με διάστημα. Υπάρχουν διάφοροι τρόποι για να μορφοποιήσετε την έξοδο.

- Για να χρησιμοποιήσετε *formatted string literals*, ξεκινήστε μια συμβολοσειρά με `f` or `F` πριν από το αρχικό εισαγωγικό ή το τριπλό εισαγωγικό. Μέσα σε αυτήν την συμβολοσειρά, μπορείτε να γράψετε μια έκφραση Python μεταξύ χαρακτήρων `{` και `}` που μπορεί να αναφέρεται σε μεταβλητές ή κυριολεκτικές τιμές.

```
>>> year = 2016
>>> event = 'Referendum'
>>> f'Results of the {year} {event}'
'Results of the 2016 Referendum'
```

- The `str.format()` method of strings requires more manual effort. You'll still use `{` and `}` to mark where a variable will be substituted and can provide detailed formatting directives, but you'll also need to provide the information to be formatted.

```
>>> yes_votes = 42_572_654
>>> no_votes = 43_132_495
>>> percentage = yes_votes / (yes_votes + no_votes)
>>> '{:-9} YES votes {:.2%}'.format(yes_votes, percentage)
' 42572654 YES votes 49.67%'
```

- Τέλος, μπορείτε να κάνετε μόνοι σας όλο τον χειρισμό συμβολοσειράς χρησιμοποιώντας λειτουργίες slicing και συνένωσης συμβολοσειρών για να δημιουργήσετε οποιαδήποτε διάταξη μπορείτε να φανταστείτε. Ο τύπος συμβολοσειράς έχει ορισμένες μεθόδους που εκτελούν χρήσιμες λειτουργίες για την προσθήκη συμβολοσειρών σε ένα δεδομένο πλάτος στήλης.

Όταν δεν χρειάζεστε φανταχτερή έξοδο, αλλά θέλετε απλώς μια γρήγορη εμφάνιση ορισμένων μεταβλητών για σκοπούς εντοπισμού σφαλμάτων, μπορείτε να μετατρέψετε οποιαδήποτε τιμή σε μια συμβολοσειρά με τις συναρτήσεις `repr()` ή `str()`.

Η συνάρτηση `str()` προορίζεται να επιστρέφει αναπαραστάσεις τιμών που είναι αρκετά αναγνώσιμες από τον άνθρωπο, ενώ το `repr()` προορίζεται για τη δημιουργία αναπαραστάσεων που μπορούν να διαβαστούν από τον διερμηνέα (ή θα επιβάλουν ένα `SyntaxError` αν δεν υπάρχει ισοδύναμη σύνταξη). Για αντικείμενα που δεν έχουν συγκεκριμένη αναπαράσταση για ανθρώπινη κατανάλωση, η `str()` θα επιστρέψει την ίδια τιμή με το `repr()`. Πολλές τιμές, όπως αριθμοί ή δομές όπως λίστες και λεξικά, έχουν την ίδια αναπαράσταση χρησιμοποιώντας οποιαδήποτε συνάρτηση. Τα strings, συγκεκριμένα, έχουν δύο διακριτές παραστάσεις.

Μερικά παραδείγματα:

```
>>> s = 'Hello, world.'
>>> str(s)
'Hello, world.'
>>> repr(s)
'"Hello, world."'
>>> str(1/7)
'0.14285714285714285'
>>> x = 10 * 3.25
>>> y = 200 * 200
>>> s = 'The value of x is ' + repr(x) + ', and y is ' + repr(y) + '...'
>>> print(s)
The value of x is 32.5, and y is 40000...
>>> # The repr() of a string adds string quotes and backslashes:
... hello = 'hello, world\n'
>>> hellos = repr(hello)
>>> print(hellos)
'hello, world\n'
>>> # The argument to repr() may be any Python object:
... repr((x, y, ('spam', 'eggs'))))
"(32.5, 40000, ('spam', 'eggs'))"
```

Το module `string` περιέχει μια κλάση `Template` που προσφέρει έναν ακόμη τρόπο αντικατάστασης τιμών σε συμβολοσειρές, χρησιμοποιώντας placeholders όπως `$x` και αντικαθιστώντας τις με τιμές από ένα λεξικό, αλλά προσφέρει πολύ λιγότερο έλεγχο της μορφοποίησης.

## 7.1.1 Μορφοποιημένα String Literals

Τα Formatted string literals (ονομάζονται επίσης f-strings για συντομία) σας επιτρέπουν να συμπεριλάβετε την τιμή των εκφράσεων Python μέσα σε μια συμβολοσειρά, θέτοντας πρόθεμα στη συμβολοσειρά με `f` ή `F` και γράφοντας εκφράσεις ως `{expression}`.

Ένας προαιρετικός αναθέτης (specifier) μορφής μπορεί να ακολουθεί την έκφραση. Αυτό επιτρέπει μεγαλύτερο έλεγχο στον τρόπο μορφοποίησης της τιμής. Το παρακάτω παράδειγμα στρογγυλοποιεί το `pi` σε τρία ψηφία μετά το δεκαδικό:

```
>>> import math
>>> print(f'The value of pi is approximately {math.pi:.3f}.')
The value of pi is approximately 3.142.
```

Η μετάδοση ενός ακέραιου αριθμού μετά το `:` θα έχει ως αποτέλεσμα αυτό το πεδίο να έχει πλάτος ελάχιστου αριθμού χαρακτήρων. Αυτό είναι χρήσιμο για την ευθυγράμμιση στηλών.

```
>>> table = {'Sjoerd': 4127, 'Jack': 4098, 'Dcab': 7678}
>>> for name, phone in table.items():
...     print(f'{name:10} ==> {phone:10d}')
...
Sjoerd      ==>      4127
```

(συνέχεια στην επόμενη σελίδα)

(συνεχίζεται από την προηγούμενη σελίδα)

Jack	==>	4098
Dcab	==>	7678

Μπορούν να χρησιμοποιηθούν άλλοι τροποποιητές για την μετατροπή της τιμής πριν τη μορφοποίηση της. Το `'!a'` ισχύει για `ascii()`, το `'!s'` ισχύει για `str()`, και το `'!r'` ισχύει για `repr()`:

```
>>> animals = 'eels'
>>> print(f'My hovercraft is full of {animals}.')
My hovercraft is full of eels.
>>> print(f'My hovercraft is full of {animals!r}.')
My hovercraft is full of 'eels'.
```

Ο αναθέτης (specifier) = μπορεί να χρησιμοποιηθεί για να επεκτείνει μια έκφραση στο κείμενο της έκφρασης, ένα σύμβολο ίσο, και μετά την αναπαράσταση της αξιολογούμενης έκφρασης:

```
>>> bugs = 'roaches'
>>> count = 13
>>> area = 'living room'
>>> print(f'Debugging {bugs=} {count=} {area=}')
Debugging bugs='roaches' count=13 area='living room'
```

Δείτε το self-documenting expressions για περισσότερες πληροφορίες σχετικά με τον αναθέτη (specifier) =. Για αναφορά σε αυτές τις προδιαγραφές μορφής, ανατρέξτε στον οδηγό αναφοράς για το `formatspec`.

## 7.1.2 Η μέθοδος String format()

Η βασική χρήση της μεθόδου `str.format()` μοιάζει με αυτό:

```
>>> print('We are the {} who say "{}!"'.format('knights', 'Ni'))
We are the knights who say "Ni!"
```

Οι αγκύλες και οι χαρακτήρες μέσα σε αυτές (που ονομάζονται πεδία μορφής) αντικαθίστανται με τα αντικείμενα που μεταβιβάζονται στη μέθοδο `str.format()`. Ένας αριθμός στις αγκύλες μπορεί να χρησιμοποιηθεί για να αναφέρεται στη θέση του αντικειμένου που μεταβιβάζεται στη μέθοδο `str.format()`.

```
>>> print('{0} and {1}'.format('spam', 'eggs'))
spam and eggs
>>> print('{1} and {0}'.format('spam', 'eggs'))
eggs and spam
```

Εάν χρησιμοποιούνται keyword ορίσματα στη μέθοδο `str.format()`, οι τιμές τους αναφέρονται χρησιμοποιώντας το όνομα του ορίσματος.

```
>>> print('This {food} is {adjective}'.format(
...     food='spam', adjective='absolutely horrible'))
This spam is absolutely horrible.
```

Τα ορίσματα θέσης και λέξης-κλειδιού μπορούν να συνδυαστούν αυθαίρετα:

```
>>> print('The story of {0}, {1}, and {other}'.format('Bill', 'Manfred',
...                                                other='Georg'))
The story of Bill, Manfred, and Georg.
```

Εάν έχετε μια συμβολοσειρά πολύ μακριάς μορφής που δεν θέλετε να χωρίσετε, θα ήταν ωραίο να αναφέρετε τις μεταβλητές που θα μορφοποιηθούν με βάση το όνομα αντί για τη θέση. Αυτό μπορεί να γίνει απλά περνώντας το λεξικό και χρησιμοποιώντας αγκύλες `'[]'` για πρόσβαση στα κλειδιά

```
>>> table = {'Sjoerd': 4127, 'Jack': 4098, 'Dcab': 8637678}
>>> print('Jack: {0[Jack]:d}; Sjoerd: {0[Sjoerd]:d}; '
...       'Dcab: {0[Dcab]:d}'.format(table))
Jack: 4098; Sjoerd: 4127; Dcab: 8637678
```

Αυτό θα μπορούσε επίσης να γίνει περνώντας το λεξικό `table` ως ορίσματα λέξεων-κλειδιών με την σημείωση `**`.

```
>>> table = {'Sjoerd': 4127, 'Jack': 4098, 'Dcab': 8637678}
>>> print('Jack: {Jack:d}; Sjoerd: {Sjoerd:d}; Dcab: {Dcab:d}'.format(**table))
Jack: 4098; Sjoerd: 4127; Dcab: 8637678
```

This is particularly useful in combination with the built-in function `vars()`, which returns a dictionary containing all local variables.

Για παράδειγμα, οι ακόλουθες γραμμές παράγουν ένα τακτοποιημένο σύνολο στηλών που δίνουν ακέραιους αριθμούς και τα τετράγωνα και τους κύβους τους:

```
>>> for x in range(1, 11):
...     print('{0:2d} {1:3d} {2:4d}'.format(x, x*x, x*x*x))
...
1    1    1
2    4    8
3    9   27
4   16   64
5   25  125
6   36  216
7   49  343
8   64  512
9   81  729
10 100 1000
```

Για μια πλήρη επισκόπηση της μορφοποίησης συμβολοσειρών με `str.format()`, δείτε `formatstrings`.

### 7.1.3 Χειροκίνητη Μορφοποίηση Συμβολοσειρών

Ακολουθεί ο ίδιος πίνακας τετραγώνων και κύβων, μορφοποιημένος χειροκίνητα:

```
>>> for x in range(1, 11):
...     print(repr(x).rjust(2), repr(x*x).rjust(3), end=' ')
...     # Note use of 'end' on previous line
...     print(repr(x*x*x).rjust(4))
...
1    1    1
2    4    8
3    9   27
4   16   64
5   25  125
6   36  216
7   49  343
8   64  512
9   81  729
10 100 1000
```

(Σημειώστε ότι το ένα κενό μεταξύ κάθε στήλης προστέθηκε με τον τρόπο που λειτουργεί το `print()`: προσθέτει πάντα κενά μεταξύ των ορισμάτων του.)

Η μέθοδος `str.rjust()` των αντικειμένων συμβολοσειράς τοποθετεί δεξιά μια συμβολοσειρά σε ένα πεδίο δεδομένου πλάτους συμπληρώνοντας την με κενά στα αριστερά. Υπάρχουν παρόμοιες μέθοδοι `str.ljust()` και `str.center()`. Αυτές οι μέθοδοι δεν γράφουν τίποτα, απλώς επιστρέφουν μια συμβολοσειρά. Εάν η συμβολοσειρά εισόδου είναι πολύ μεγάλη, δεν την περικόπτουν, αλλά την επιστρέφουν αμετά-

βλῃτη· αυτό θα μπερδέψει τη διάταξη της στήλης σας, αλλά αυτό είναι συνήθως καλύτερο από την εναλλακτική, που θα ήταν ψέματα για μια τιμή. (Αν θέλετε πραγματικά περικοπή, μπορείτε πάντα να προσθέσετε μια λειτουργία `slice`, όπως στο `x.ljust(n)[:n]`.)

Υπάρχει μια άλλη μέθοδος, η `str.zfill()`, η οποία συμπληρώνει μια αριθμητική συμβολοσειρά στα αριστερά με μηδενικά. Καταλαβαίνει τα σύμβολα `syn` και `πλην`:

```
>>> '12'.zfill(5)
'00012'
>>> '-3.14'.zfill(7)
'-003.14'
>>> '3.14159265359'.zfill(5)
'3.14159265359'
```

## 7.1.4 Παλιά μορφοποίηση συμβολοσειράς

The `%` operator (modulo) can also be used for string formatting. Given `'string' % values`, instances of `%` in `string` are replaced with zero or more elements of `values`. This operation is commonly known as string interpolation. For example:

```
>>> import math
>>> print('The value of pi is approximately %5.3f.' % math.pi)
The value of pi is approximately 3.142.
```

Περισσότερες πληροφορίες μπορείτε να βρείτε στην ενότητα `old-string-formatting`.

## 7.2 Ανάγνωση και Εγγραφή Αρχείων

Η `open()` επιστρέφει ένα *file object*, και χρησιμοποιείται πιο συχνά με δύο ορίσματα θέσης και ένα όρισμα λέξης-κλειδιού: `open(filename, mode, encoding=None)`

```
>>> f = open('workfile', 'w', encoding="utf-8")
```

Το πρώτο όρισμα είναι μια συμβολοσειρά που περιέχει το όνομα αρχείου. Το δεύτερο όρισμα είναι μια άλλη συμβολοσειρά που περιέχει μερικούς χαρακτήρες που περιγράφουν τον τρόπο με τον οποίο θα χρησιμοποιηθεί το αρχείο. Η *mode* μπορεί να είναι `'r'` όταν το αρχείο θα είναι μόνο για διάβασμα, `'w'` μόνο για εγγραφή (ένα υπάρχον αρχείο με το ίδιο όνομα θα διαγραφεί) και το `'a'` ανοίγει το αρχείο για προσάρτηση· οποιαδήποτε δεδομένα γράφονται στο αρχείο και προστίθενται αυτόματα στο τέλος. Το `'r+'` ανοίγει το αρχείο τόσο για ανάγνωση όσο και για γραφή. Το όρισμα *mode* είναι προαιρετικό· το `'r'` θα θεωρείται εάν παραληφθεί.

Κανονικά, τα αρχεία ανοίγουν σε *text mode*, που σημαίνει ότι διαβάζετε και γράφετε συμβολοσειρές από και προς το αρχείο, οι οποίες κωδικοποιούνται σε μια συγκεκριμένη *κωδικοποίηση*. Εάν δεν έχει καθοριστεί η *κωδικοποίηση*, η προεπιλογή είναι εξαρτώμενη από την πλατφόρμα (δείτε `open()`). Επειδή το UTF-8 είναι το σύγχρονο de-facto standard, `encoding="utf-8"` συνίσταται εκτός εάν γνωρίζετε ότι πρέπει να χρησιμοποιήσετε διαφορετική κωδικοποίηση. Η προσθήκη ενός `'b'` στη λειτουργία ανοίγει το αρχείο σε *binary mode*. Τα δεδομένα δυαδικής λειτουργίας διαβάζονται και γράφονται ως αντικείμενα `bytes`. Δεν μπορείτε να καθορίσετε *κωδικοποίηση* όταν ανοίγετε αρχείο σε δυαδική λειτουργία.

Στη λειτουργία κειμένου, η προεπιλογή (default) κατά την ανάγνωση είναι να μετατρέψετε τις καταλήξεις γραμμών για συγκεκριμένη πλατφόρμα (`\n` στο Unix, `\r\n` στα Windows) σε μόνο `\n`. Όταν γράφετε σε λειτουργία κειμένου, η προεπιλογή είναι να μετατρέπονται οι εμφανίσεις `\n` σε καταλήξεις γραμμών για συγκεκριμένη πλατφόρμα. Αυτή η παρασκηνακή τροποποίηση στα δεδομένα αρχείων είναι καλή για αρχεία κειμένου, αλλά θα καταστρέψει δυαδικά δεδομένα όπως αυτό σε αρχεία JPEG ή EXE. Να είστε πολύ προσεκτικοί να χρησιμοποιείτε τη δυαδική λειτουργία όταν διαβάζετε και γράφετε τέτοια αρχεία.

Είναι καλή πρακτική να χρησιμοποιείτε το keyword `with` όταν ασχολούμαστε με αντικείμενα αρχείου. Το πλεονέκτημα είναι ότι το αρχείο κλείνει σωστά μετά την ολοκλήρωση της εκτέλεσής του, ακόμα κι αν κάποια

στιγμή προκύψει εξαίρεση. Χρησιμοποιώντας το `with` είναι επίσης πολύ πιο σύντομο από την σύνταξη ισοδύναμου `try-finally blocks`:

```
>>> with open('workfile', encoding="utf-8") as f:
...     read_data = f.read()

>>> # We can check that the file has been automatically closed.
>>> f.closed
True
```

Εάν δεν χρησιμοποιείτε τη keyword `with`, τότε θα πρέπει να καλείτε τη `f.close()` για να κλείσετε το αρχείο και να ελευθερώσετε αμέσως τυχόν πόρους συστήματος που χρησιμοποιούνται από αυτό.

**Προειδοποίηση:** Η κλήση του `f.write()` χωρίς τη χρήση της keyword `with` ή η κλήση του `f.close()` μπορεί να οδηγήσει στα ορίσματα του `f.write()` να μην εγγραφεί πλήρως στο δίσκο, ακόμα και αν το πρόγραμμα εξέλθει με επιτυχία.

Μετά το κλείσιμο ενός αντικειμένου αρχείου, είτε μια δήλωση `with` είτε καλώντας `f.close()`, οι προσπάθειες χρήσης του αντικειμένου αρχείου θα αποτύχουν αυτόματα.

```
>>> f.close()
>>> f.read()
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ValueError: I/O operation on closed file.
```

## 7.2.1 Μέθοδοι Αντικειμένων Αρχείων

Τα υπόλοιπα παραδείγματα σε αυτήν την ενότητα θα υποθέσουν ότι ένα αντικείμενο αρχείου που ονομάζεται `f` έχει ήδη δημιουργηθεί.

Για να διαβάσετε τα περιεχόμενα ενός αρχείου, καλέστε το `f.read(size)`, το οποίο διαβάζει κάποια ποσότητα δεδομένων και την επιστρέφει ως συμβολοσειρά (σε λειτουργία κειμένου) ή ως αντικείμενο `bytes` (σε δυαδική λειτουργία). Το `size` είναι προαιρετικό αριθμητικό όρισμα. Όταν το `size` παραλείπεται ή είναι αρνητικό, ολόκληρο το περιεχόμενο του αρχείου θα διαβαστεί και θα επιστραφεί• είναι δικό σας πρόβλημα εάν το αρχείο είναι διπλάσιο από τη μνήμη του υπολογιστή σας. Διαφορετικά, διαβάζονται και επιστρέφονται το πολύ `size` χαρακτήρες (σε λειτουργία κειμένου) ή `byte size` (σε δυαδική λειτουργία). Εάν έχει φτάσει το τέλος του αρχείου, το `f.read()` θα επιστρέψει μια κενή συμβολοσειρά (`' '`).

```
>>> f.read()
'This is the entire file.\n'
>>> f.read()
''
```

Το `f.readline()` διαβάζει μία γραμμή από το αρχείο• ένας χαρακτήρας νέας γραμμής (`\n`) παραμένει στο τέλος της συμβολοσειράς, και παραλείπεται μόνο στην τελευταία γραμμή του αρχείου εάν το αρχείο δεν τελειώνει σε μια νέα γραμμή. Αυτό καθιστά την τιμή επιστροφής σαφή• εάν το `f.readline()` επιστρέφει μια κενή συμβολοσειρά, έχει φτάσει στο τέλος του αρχείου, ενώ μια κενή γραμμή αντιπροσωπεύεται από `' \n '`, μια συμβολοσειρά που περιέχει μόνο μία νέα γραμμή.

```
>>> f.readline()
'This is the first line of the file.\n'
>>> f.readline()
'Second line of the file\n'
>>> f.readline()
''
```

Για την ανάγνωση γραμμών από ένα αρχείο, μπορείτε να κάνετε `loop` πάνω από το αντικείμενο του αρχείου. Αυτό είναι αποδοτικό στη μνήμη, γρήγορο και οδηγεί σε απλό κώδικα:

```
>>> for line in f:
...     print(line, end='')
...
This is the first line of the file.
Second line of the file
```

Εάν θέλετε να διαβάσετε όλες τις γραμμές ενός αρχείου σε μια λίστα, μπορείτε επίσης να χρησιμοποιήσετε το `list(f)` ή `f.readlines()`.

Το `f.write(string)` γράφει τα περιεχόμενα του *string* στο αρχείο, επιστρέφοντας τον αριθμό των χαρακτήρων που γράφτηκαν.

```
>>> f.write('This is a test\n')
15
```

Άλλοι τύποι αντικειμένων πρέπει να μετατραπούν – είτε σε μια συμβολοσειρά (σε λειτουργία κειμένου) ή σε ένα αντικείμενο `bytes` (σε δυαδική λειτουργία) – πριν τα γράψετε:

```
>>> value = ('the answer', 42)
>>> s = str(value) # convert the tuple to string
>>> f.write(s)
18
```

Το `f.tell()` επιστρέφει έναν ακέραιο που δίνει την τρέχουσα θέση του αντικειμένου αρχείου στο αρχείο που αντιπροσωπεύεται ως αριθμό `byte` από την αρχή του αρχείου όταν βρίσκεται σε δυαδική λειτουργία και έναν αδιαφανή αριθμό όταν βρίσκεται σε λειτουργία κειμένου.

Για να αλλάξετε τη θέση του αντικειμένου, χρησιμοποιήστε το `f.seek(offset, whence)`. Η θέση υπολογίζεται από την προσθήκη *offset* σε ένα σημείο αναφοράς· το σημείο αναφοράς επιλέγεται από το όρισμα *whence*. Μια 0 τιμή *whence* μετρά από την αρχή του αρχείου το 1 χρησιμοποιεί την τρέχουσα θέση αρχείου και το 2 χρησιμοποιεί το τέλος του αρχείου ως σημείο αναφοράς. Το *whence* μπορεί να παραληφθεί και να οριστεί από προεπιλογή 0, χρησιμοποιώντας την αρχή του αρχείου ως σημείο αναφοράς.

```
>>> f = open('workfile', 'rb+')
>>> f.write(b'0123456789abcdef')
16
>>> f.seek(5) # Go to the 6th byte in the file
5
>>> f.read(1)
b'5'
>>> f.seek(-3, 2) # Go to the 3rd byte before the end
13
>>> f.read(1)
b'd'
```

Σε αρχεία κειμένου (αυτά που ανοίγουν χωρίς `b` στη λειτουργία `string`), επιτρέπονται μόνο αναζητήσεις σε σχέση με την αρχή του αρχείου (η εξαίρεση είναι η αναζήτηση μέχρι το ίδιο το αρχείο που τελειώνει με `seek(0, 2)`) και οι μόνες έγκυρες τιμές *offset* είναι αυτές που επιστρέφονται από το `f.tell()`, ή μηδέν. Οποιαδήποτε άλλη τιμή *offset* παράγει απροσδιόριστη συμπεριφορά.

File objects have some additional methods, such as `isatty()` and `truncate()` which are less frequently used; consult the Library Reference for a complete guide to file objects.



## 7.2.2 Αποθήκευση δομημένων δεδομένων με json

Strings can easily be written to and read from a file. Numbers take a bit more effort, since the `read()` method only returns strings, which will have to be passed to a function like `int()`, which takes a string like `'123'` and returns its numeric value 123. When you want to save more complex data types like nested lists and dictionaries, parsing and serializing by hand becomes complicated.

Αντί να χρειάζεται οι χρήστες να γράφουν και να διορθώνουν συνεχώς κώδικα για να αποθηκεύουν πολύπλοκους τύπους δεδομένων σε αρχεία, η Python σας επιτρέπει να χρησιμοποιείτε τη δημοφιλή μορφή ανταλλαγής δεδομένων που ονομάζεται **JSON (JavaScript Object Notation)**. Το standard module που ονομάζεται `json` μπορεί να λάβει ιεραρχίες δεδομένων Python, και να τις μετατρέψει σε αναπαράστασεις συμβολοσειρών· αυτή η διαδικασία ονομάζεται *serializing*. Η ανασύνθεση των δεδομένων από την αναπαράσταση συμβολοσειράς ονομάζεται *deserializing*. Μεταξύ σειριοποίησης και αποσειριοποίησης, η συμβολοσειρά που αντιπροσωπεύει το αντικείμενο μπορεί να έχει αποθηκευτεί σε ένα αρχείο ή δεδομένα ή να έχει σταλεί μέσω μιας σύνδεσης δικτύου σε κάποιο απομακρυσμένο μηχάνημα.

**Σημείωση:** Η μορφή JSON χρησιμοποιείται συνήθως από σύγχρονες εφαρμογές για να επιτρέψει την ανταλλαγή δεδομένων. Πολλοί προγραμματιστές είναι ήδη εξοικειωμένοι με αυτήν, γεγονός που την καθιστά καλή επιλογή για διαλειτουργικότητα.

Εάν έχετε ένα αντικείμενο `x`, μπορείτε να δείτε την αναπαράσταση συμβολοσειράς JSON με μια απλή γραμμή κώδικα:

```
>>> import json
>>> x = [1, 'simple', 'list']
>>> json.dumps(x)
'[1, "simple", "list"]'
```

Μια άλλη παραλλαγή της συνάρτησης `dumps()`, που ονομάζεται `dump()`, απλώς σειριοποιεί το αντικείμενο σε ένα *text file*. Έτσι, εάν το `f` είναι ένα αντικείμενο *text file* που ανοίγει για εγγραφή, μπορούμε να κάνουμε αυτό:

```
json.dump(x, f)
```

Για να αποκωδικοποιήσετε ξανά το αντικείμενο, εάν το `f` είναι ένα αντικείμενο *binary file* ή *text file* που έχει ανοίξει για ανάγνωση:

```
x = json.load(f)
```

**Σημείωση:** Τα αρχεία JSON πρέπει να είναι κωδικοποιημένα σε UTF-8. Χρησιμοποιήστε το `encoding="utf-8"` όταν ανοίγετε το αρχείο JSON ως *text file* τόσο για ανάγνωση όσο και για εγγραφή.

Αυτή η απλή τεχνική σειριοποίησης μπορεί να χειριστεί λίστες και λεξικά, αλλά η σειριοποίηση αυθαίρετων στιγμοτύπων κλάσεων σε JSON απαιτεί λίγη επιπλέον προσπάθεια. Η αναφορά για το module `json` περιέχει μια εξήγηση για αυτό.

**Δείτε επίσης:**

`pickle` - το `pickle` module

Σε αντίθεση με το *JSON*, το *pickle* είναι ένα πρωτόκολλο που επιτρέπει τη σειριοποίηση αυθαίρετα πολύπλοκων αντικειμένων Python. Ως εκ τούτου, είναι συγκεκριμένο για την Python και δεν μπορεί να χρησιμοποιηθεί για επικοινωνία με εφαρμογές γραμμένες σε άλλες γλώσσες. Είναι επίσης ανασφαλές από προεπιλογή: η αποσειριοποίηση `pickle` δεδομένων που προέρχονται από μια μη αξιόπιστη πηγή μπορεί να εκτελέσει αυθαίρετο κώδικα, εάν τα δεδομένα έχουν δημιουργηθεί από έναν έμπειρο εισβολέα.



---

## Σφάλματα και Εξαιρέσεις

---

Μέχρι τώρα τα μηνύματα σφαλμάτων (error messages) δεν ήταν περισσότερα από όσα αναφέρθηκαν, αλλά αν έχετε δοκιμάσει τα παραδείγματα, πιθανότατα έχετε δει μερικά. Υπάρχουν (τουλάχιστον) δύο διαφορετικά είδη σφαλμάτων: *syntax errors* (συντακτικά σφάλματα) και *exceptions* (εξαιρέσεις).

### 8.1 Syntax Errors (Συντακτικά Σφάλματα)

Τα syntax errors, γνωστά και ως parsing errors, είναι ίσως το πιο συνηθισμένο είδος παραπόνου που λαμβάνετε ενώ εξακολουθείτε να μαθαίνετε Python:

```
>>> while True print('Hello world')
File "<stdin>", line 1
    while True print('Hello world')
                ^
SyntaxError: invalid syntax
```

The parser repeats the offending line and displays a little “arrow” pointing at the earliest point in the line where the error was detected. The error is caused by (or at least detected at) the token *preceding* the arrow: in the example, the error is detected at the function `print()`, since a colon (':') is missing before it. File name and line number are printed so you know where to look in case the input came from a script.

### 8.2 Exceptions (Εξαιρέσεις)

Ακόμη και αν μια πρόταση ή μια έκφραση είναι συντακτικά σωστή, μπορεί να προκαλέσει σφάλμα όταν γίνεται προσπάθεια εκτέλεσής της. Τα σφάλματα που εντοπίζονται κατά την εκτέλεση ονομάζονται *εξαιρέσεις* και δεν είναι άνευ όρων μοιραία (fatal): σύντομα θα μάθετε πως να τα χειρίζεστε σε προγράμματα Python. Ωστόσο, οι περισσότερες εξαιρέσεις δεν αντιμετωπίζονται από προγράμματα και οδηγούν σε μηνύματα σφάλματος όπως φαίνεται εδώ:

```
>>> 10 * (1/0)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ZeroDivisionError: division by zero
>>> 4 + spam*3
```

(συνέχεια στην επόμενη σελίδα)

(συνεχίζεται από την προηγούμενη σελίδα)

```
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'spam' is not defined
>>> '2' + 2
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: can only concatenate str (not "int") to str
```

Η τελευταία γραμμή του μηνύματος σφάλματος υποδεικνύει τι συνέβη. Οι εξαιρέσεις υπάρχουν σε διαφορετικούς τύπους και ο τύπος εκτυπώνεται ως μέρος του μηνύματος: οι τύποι στο παράδειγμα είναι `ZeroDivisionError`, `NameError` και `TypeError`. Η συμβολοσειρά που εκτυπώνεται ως τύπος εξαιρέσεως είναι όνομα της ενσωματωμένης εξαίρεσης που προέκυψε. Αυτό ισχύει για όλες τις ενσωματωμένες (built-in) εξαιρέσεις, αλλά δεν χρειάζεται να ισχύει για εξαιρέσεις που ορίζονται από το χρήστη (αν και είναι μια χρήσιμη σύμβαση). Οι standard εξαιρέσεις είναι ενσωματωμένα (built-in) αναγνωριστικά (όχι δεσμευμένες λέξεις-κλειδιά).

Η υπόλοιπη γραμμή παρέχει λεπτομέρειες με βάση τον τύπο της εξαίρεσης και το τι την προκάλεσε.

Το προηγούμενο μέρος του μηνύματος σφάλματος εμφανίζει το περιβάλλον όπου συνέβη η εξαίρεση, με τη μορφή ανίχνευσης στοίβας. Γενικά περιέχει μια στοίβα ανίχνευσης γραμμών πηγής· ωστόσο, δεν θα εμφανίζει γραμμές που διαβάζονται από standard είσοδο.

Το `builtin-exceptions` παραθέτει τις ενσωματωμένες εξαιρέσεις και τις έννοιές τους.

## 8.3 Διαχείριση Εξαιρέσεων

Είναι δυνατό να γραφτεί κώδικας που χειρίζεται επιλεγμένες εξαιρέσεις. Κοιτάξτε το ακόλουθο παράδειγμα, το οποίο ζητά από τον χρήστη να εισάγει έναν έγκυρο ακέραιο αριθμό, αλλά επιτρέπει στον χρήστη να διακόψει το πρόγραμμα (χρησιμοποιώντας `Control-C` ή ό,τι υποστηρίζει το λειτουργικό σύστημα)· σημειώστε ότι μια διακοπή που δημιουργείται από τον χρήστη σηματοδοτείται κάνοντας `raise` την εξαίρεση `KeyboardInterrupt`.

```
>>> while True:
...     try:
...         x = int(input("Please enter a number: "))
...         break
...     except ValueError:
...         print("Oops! That was no valid number. Try again...")
... 
```

Η δήλωση `try` λειτουργεί ως εξής.

- Πρώτον, εκτελείται η *try clause* (η πρόταση(εις) μεταξύ των λέξεων-κλειδιών `try` and `except`).
- Εάν δεν προκύψει εξαίρεση, η *except clause* παραλείπεται και η εκτέλεση της πρότασης `try` ολοκληρώνεται.
- Εάν παρουσιαστεί μια εξαίρεση κατά την εκτέλεση της πρότασης `try`, η υπόλοιπη πρόταση παραλείπεται. Στη συνέχεια, εάν ο τύπος της ταιριάζει με την εξαίρεση που ονομάζεται από τη λέξη-κλειδί `except`, η *except clause* εκτελείται, και στη συνέχεια η εκτέλεση συνεχίζεται μετά το μπλοκ `try/except`.
- If an exception occurs which does not match the exception named in the *except clause*, it is passed on to outer `try` statements; if no handler is found, it is an *unhandled exception* and execution stops with a message as shown above.

Μια πρόταση `try` μπορεί να έχει περισσότερες από μία *except clause*, για να καθορίσει χειριστές για διαφορετικές εξαιρέσεις. Το πολύ ένας χειριστής θα εκτελεστεί. Οι χειριστές χειρίζονται μόνο εξαιρέσεις που εμφανίζονται στην αντίστοιχη *try clause*, όχι σε άλλους χειριστές της ίδιας πρότασης `try`. Μια *except clause* μπορεί να ονομάσει πολλαπλές εξαιρέσεις ως πλειάδα (tuple) σε παρένθεση, για παράδειγμα:

```
... except (RuntimeError, TypeError, NameError):
...     pass
```

A class in an `except` clause is compatible with an exception if it is the same class or a base class thereof (but not the other way around — an *except clause* listing a derived class is not compatible with a base class). For example, the following code will print B, C, D in that order:

```
class B(Exception):
    pass

class C(B):
    pass

class D(C):
    pass

for cls in [B, C, D]:
    try:
        raise cls()
    except D:
        print("D")
    except C:
        print("C")
    except B:
        print("B")
```

Σημειώστε ότι εάν οι *except clauses* είχαν αντιστραφεί (με το `except B` πρώτα), θα είχε εκτυπωθεί B, B, B — ενεργοποιείται η πρώτη αντιστοίχιση *except clause*.

All exceptions inherit from `BaseException`, and so it can be used to serve as a wildcard. Use this with extreme caution, since it is easy to mask a real programming error in this way! It can also be used to print an error message and then re-raise the exception (allowing a caller to handle the exception as well):

```
import sys

try:
    f = open('myfile.txt')
    s = f.readline()
    i = int(s.strip())
except OSError as err:
    print("OS error: {0}".format(err))
except ValueError:
    print("Could not convert data to an integer.")
except BaseException as err:
    print(f"Unexpected {err=}, {type(err)=}")
    raise
```

Alternatively the last `except` clause may omit the exception name(s), however the exception value must then be retrieved from `sys.exc_info()[1]`.

Η πρόταση `try ... except` έχει ένα προαιρετικό *else clause*, το οποίο, όταν υπάρχει, πρέπει να ακολουθεί όλες τις *except clauses*. Είναι χρήσιμο για κώδικα που πρέπει να εκτελεστεί εάν το *try clause* δεν κάνει `raise` μια εξαίρεση. Για παράδειγμα:

```
for arg in sys.argv[1:]:
    try:
        f = open(arg, 'r')
    except OSError:
        print('cannot open', arg)
    else:
        print(arg, 'has', len(f.readlines()), 'lines')
        f.close()
```

Η χρήση της πρότασης `else` είναι καλύτερη από την προσθήκη πρόσθετου κώδικα στην πρόταση `try`, επειδή αποφεύγει την κατά λάθος σύλληψη μιας εξαίρεσης που δεν προέκυψε από τον κώδικα που προστατεύεται από την πρόταση `try ... except`.

When an exception occurs, it may have an associated value, also known as the exception's *argument*. The presence and type of the argument depend on the exception type.

The *except clause* may specify a variable after the exception name. The variable is bound to an exception instance with the arguments stored in `instance.args`. For convenience, the exception instance defines `__str__()` so the arguments can be printed directly without having to reference `.args`. One may also instantiate an exception first before raising it and add any attributes to it as desired.

```
>>> try:
...     raise Exception('spam', 'eggs')
... except Exception as inst:
...     print(type(inst))    # the exception instance
...     print(inst.args)     # arguments stored in .args
...     print(inst)         # __str__ allows args to be printed directly,
...                         # but may be overridden in exception subclasses
...     x, y = inst.args     # unpack args
...     print('x =', x)
...     print('y =', y)
...
<class 'Exception'>
('spam', 'eggs')
('spam', 'eggs')
x = spam
y = eggs
```

If an exception has arguments, they are printed as the last part (“detail”) of the message for unhandled exceptions.

Exception handlers don't just handle exceptions if they occur immediately in the *try clause*, but also if they occur inside functions that are called (even indirectly) in the *try clause*. For example:

```
>>> def this_fails():
...     x = 1/0
...
>>> try:
...     this_fails()
... except ZeroDivisionError as err:
...     print('Handling run-time error:', err)
...
Handling run-time error: division by zero
```

## 8.4 Raising Εξαιρέσεων

Η δήλωση `raise` επιτρέπει στον προγραμματιστή να αναγκάσει να εμφανιστεί μια καθορισμένη εξαίρεση. Για παράδειγμα:

```
>>> raise NameError('HiThere')
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: HiThere
```

The sole argument to `raise` indicates the exception to be raised. This must be either an exception instance or an exception class (a class that derives from `Exception`). If an exception class is passed, it will be implicitly instantiated by calling its constructor with no arguments:

```
raise ValueError # shorthand for 'raise ValueError()'
```

Εάν πρέπει να προσδιορίσετε εάν έχει εγγραφεί μια εξαίρεση, αλλά δεν σκοπεύετε να τη χειριστείτε, μια απλούστερη μορφή της δήλωσης `raise` σας επιτρέπει να κάνετε ξανά `raise` την εξαίρεση:

```
>>> try:
...     raise NameError('HiThere')
... except NameError:
...     print('An exception flew by!')
...     raise
...
An exception flew by!
Traceback (most recent call last):
  File "<stdin>", line 2, in <module>
NameError: HiThere
```

## 8.5 Αλυσιδωτές Εξαιρέσεις

Εάν παρουσιαστεί μια μη χειριζόμενη (unhandled) εξαίρεση μέσα σε μια ενότητα `except`, θα επισυνάψει την εξαίρεση που θα χειριστεί και θα συμπεριληφθεί στο μήνυμα σφάλματος:

```
>>> try:
...     open("database.sqlite")
... except OSError:
...     raise RuntimeError("unable to handle error")
...
Traceback (most recent call last):
  File "<stdin>", line 2, in <module>
FileNotFoundError: [Errno 2] No such file or directory: 'database.sqlite'

During handling of the above exception, another exception occurred:

Traceback (most recent call last):
  File "<stdin>", line 4, in <module>
RuntimeError: unable to handle error
```

Για να υποδείξετε ότι μια εξαίρεση είναι άμεση συνέπεια μιας άλλης, η πρόταση `raise` επιτρέπει μια προαιρετική πρόταση `from`:

```
# exc must be exception instance or None.
raise RuntimeError from exc
```

Αυτό μπορεί να είναι χρήσιμο όταν μετασχηματίζεται εξαιρέσεις. Για παράδειγμα:

```
>>> def func():
...     raise ConnectionError
...
>>> try:
...     func()
... except ConnectionError as exc:
...     raise RuntimeError('Failed to open database') from exc
...
Traceback (most recent call last):
  File "<stdin>", line 2, in <module>
  File "<stdin>", line 2, in func
ConnectionError

The above exception was the direct cause of the following exception:

Traceback (most recent call last):
  File "<stdin>", line 4, in <module>
RuntimeError: Failed to open database
```

Επιτρέπει επίσης την απενεργοποίηση της αυτόματης αλυσίδας εξαιρέσεων χρησιμοποιώντας `from None` idiom:

```
>>> try:
...     open('database.sqlite')
... except OSError:
...     raise RuntimeError from None
...
Traceback (most recent call last):
  File "<stdin>", line 4, in <module>
RuntimeError
```

Για περισσότερες πληροφορίες σχετικά με την μηχανική αλυσίδων, δείτε `bltin-exceptions`.

## 8.6 Εξαιρέσεις που καθορίζονται από το χρήστη

Τα προγράμματα μπορούν να ονομάσουν τις δικές τους εξαιρέσεις δημιουργώντας μια νέα κλάση εξαιρέσεων (δείτε *Κλάσεις* για περισσότερα σχετικά με τις κλάσεις Python). Οι εξαιρέσεις θα πρέπει συνήθως να προέρχονται από την κλάση `Exception`, είτε άμεσα είτε έμμεσα.

Μπορούν να οριστούν κλάσεις εξαίρεσης που κάνουν οτιδήποτε μπορεί να κάνει οποιαδήποτε άλλη κλάση, αλλά συνήθως διατηρούνται απλές, συχνά προσφέρουν μόνο έναν αριθμό χαρακτηριστικών που επιτρέπουν την εξαγωγή πληροφοριών σχετικά με το σφάλμα από τους χειριστές για την εξαίρεση.

Οι περισσότερες εξαιρέσεις ορίζονται με ονόματα που τελειώνουν σε «Error», παρόμοια με την ονομασία των τυπικών εξαιρέσεων.

Many standard modules define their own exceptions to report errors that may occur in functions they define. More information on classes is presented in chapter *Κλάσεις*.

## 8.7 Καθορισμός ενεργειών καθαρισμού

Η δήλωση `try` έχει μια άλλη προαιρετική πρόταση που προορίζεται να ορίσει ενέργειες καθαρισμού που πρέπει να εκτελεστούν υπό οποιεσδήποτε συνθήκες. Για παράδειγμα:

```
>>> try:
...     raise KeyboardInterrupt
... finally:
...     print('Goodbye, world!')
...
Goodbye, world!
Traceback (most recent call last):
  File "<stdin>", line 2, in <module>
KeyboardInterrupt
```

Εάν υπάρχει μια πρόταση `finally`, η πρόταση `finally` θα εκτελεστεί ως η τελευταία εργασία πριν από την ολοκλήρωση της πρότασης `try`. Η πρόταση `finally` εκτελείται είτε όχι η πρόταση `try` παράγει μια εξαίρεση. Τα ακόλουθα σημεία συζητούν πιο περίπλοκες περιπτώσεις όταν εμφανίζεται μια εξαίρεση:

- Εάν παρουσιαστεί μια εξαίρεση κατά την εκτέλεση της πρότασης `try`, η εξαίρεση μπορεί να αντιμετωπιστεί από μια πρόταση `except`. Εάν η εξαίρεση δεν αντιμετωπίζεται από μια πρόταση `except`, η εξαίρεση γίνεται ξανά `raise` μετά την εκτέλεση της πρότασης `finally`.
- Μια εξαίρεση θα μπορούσε να προκύψει κατά την εκτέλεση μιας πρότασης `except` ή `else`. Και πάλι, η εξαίρεση τίθεται ξανά μετά την εκτέλεση της πρότασης `finally`.
- Εάν η πρόταση `finally` εκτελέσει μια πρόταση `break`, `continue` ή `return`, οι εξαιρέσεις δεν αυξάνονται εκ νέου.

- Εάν η πρόταση `try` φτάσει σε μια δήλωση `break`, `continue` ή `return`, η πρόταση `finally` θα εκτελεστεί ακριβώς πριν από τα `break`, `continue` or `return` της εκτέλεσης της δήλωσης.
- Εάν μια πρόταση `finally` περιλαμβάνει μια δήλωση `return`, η τιμή που επιστρέφεται θα είναι αυτή από την πρόταση `finally` της δήλωσης της `return`, και όχι η τιμή από τη δήλωση `try` της πρότασης `return`.

Για παράδειγμα:

```
>>> def bool_return():
...     try:
...         return True
...     finally:
...         return False
...
>>> bool_return()
False
```

Ένα πιο περίπλοκο παράδειγμα:

```
>>> def divide(x, y):
...     try:
...         result = x / y
...     except ZeroDivisionError:
...         print("division by zero!")
...     else:
...         print("result is", result)
...     finally:
...         print("executing finally clause")
...
>>> divide(2, 1)
result is 2.0
executing finally clause
>>> divide(2, 0)
division by zero!
executing finally clause
>>> divide("2", "1")
executing finally clause
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "<stdin>", line 3, in divide
TypeError: unsupported operand type(s) for /: 'str' and 'str'
```

Όπως μπορείτε να δείτε, η πρόταση `finally` εκτελείται σε οποιαδήποτε περίπτωση. Το `TypeError` που δημιουργείται με τη διαίρεση δύο συμβολοσειρών δεν χειρίζεται από την πρόταση `except` και επομένως γίνεται ξανά `raise` μετά την εκτέλεση του όρου `finally`.

Στις εφαρμογές του πραγματικού κόσμου, η πρόταση `finally` είναι χρήσιμη για την απελευθέρωση εξωτερικών πόρων (όπως αρχεία ή συνδέσεις δικτύου), ανεξάρτητα από το εάν η χρήση του πόρου ήταν επιτυχής.

## 8.8 Προκαθορισμένες ενέργειες καθαρισμού

Μερικά αντικείμενα ορίζουν τις τυπικές ενέργειες καθαρισμού που πρέπει να αναλαμβάνονται όταν το αντικείμενο δεν χρειάζεται πλέον, ανεξάρτητα από το εάν η λειτουργία που χρησιμοποιεί το αντικείμενο πέτυχε ή απέτυχε. Κοιτάξτε το ακόλουθο αντικείμενο, το οποίο προσπαθεί να ανοίξει ένα αρχείο και να εκτυπώσει τα περιεχόμενα του στην οθόνη.

```
for line in open("myfile.txt"):
    print(line, end="")
```

Το πρόβλημα με αυτόν τον κώδικα είναι ότι αφήνει το αρχείο ανοιχτό για απροσδιόριστο χρονικό διάστημα μετά την ολοκλήρωση της εκτέλεσης αυτού του τμήματος του κώδικα. Αυτό δεν είναι πρόβλημα σε απλά σενάρια, αλλά μπορεί να είναι πρόβλημα για μεγαλύτερες εφαρμογές. Η δήλωση `with` επιτρέπει σε αντικείμενα όπως αρχεία να χρησιμοποιούνται με τρόπο που διασφαλίζει ότι καθαρίζονται πάντα άμεσα και σωστά.

```
with open("myfile.txt") as f:
    for line in f:
        print(line, end=" ")
```

Μετά την εκτέλεση της πρότασης, το αρχείο `f` είναι πάντα κλειστό, ακόμα και αν παρουσιάστηκε πρόβλημα κατά την επεξεργασία των γραμμών. Τα αντικείμενα που, όπως τα αρχεία παρέχουν προκαθορισμένες ενέργειες καθαρισμού θα το υποδεικνύουν στην τεκμηρίωσή τους.



---

Κλάσεις

---

Οι κλάσεις παρέχουν ένα μέσο ομαδοποίησης δεδομένων και λειτουργικότητας. Η δημιουργία μιας νέας κλάσης δημιουργεί έναν νέο *τύπο* αντικειμένου, επιτρέποντας νέα *στιγμιότυπα* αυτού του τύπου που πρόκειται να γίνουν. Κάθε στιγμιότυπο κλάσης μπορεί να έχει χαρακτηριστικά που συνδέονται με αυτό για τη διατήρηση της κατάστασής του. Τα στιγμιότυπα κλάσης μπορούν να έχουν επίσης μεθόδους (που ορίζονται από την κλάση του) για την τροποποίηση της κατάστασής του.

Σε σύγκριση με άλλες γλώσσες προγραμματισμού, ο μηχανισμός κλάσης της Python προσθέτει κλάσεις με ελάχιστο νέο συντακτικό και σημασιολογία. Είναι ένα μείγμα των μηχανισμών κλάσεων που βρέθηκαν στη C++ και στο Modula-3. Οι κλάσεις της Python παρέχουν όλα τα standard χαρακτηριστικά του Αντικειμενοστραφούς Προγραμματισμού: ο μηχανισμός της κληρονομικότητας της κλάσης επιτρέπει την ύπαρξη πολλαπλών βασικών κλάσεων, μια παραγόμενη κλάση να μπορεί να παρακάμψει οποιεσδήποτε μεθόδους της βασικής κλάσης ή κλάσεων, και μια μέθοδος να μπορεί να καλέσει τη μέθοδο μίας βασικής κλάσης με το ίδιο όνομα. Τα αντικείμενα μπορούν να περιέχουν αυθαίρετα ποσά και είδη δεδομένων. Όπως ισχύει για τα modules, οι κλάσεις συμμετέχουν στη δυναμική φύση της Python: δημιουργούνται κατά το χρόνο εκτέλεσης και μπορούν να τροποποιηθούν περαιτέρω μετά τη δημιουργία.

Στην ορολογία της C++, συνήθως τα μέλη της κλάσης (συμπεριλαμβανομένων των μελών δεδομένων) είναι *δημόσια* (εκτός από βλέπε παρακάτω *Ιδιωτικές Μεταβλητές*), και όλες οι συμμετέχουσες συναρτήσεις είναι *εικονικές*. Όπως και στο Modula-3, δεν υπάρχουν συντομογραφίες για την αναφορά στα μέλη του αντικειμένου από τις μεθόδους του: η μέθοδος δηλώνεται με ρητό πρώτο όρισμα που αντιπροσωπεύει το αντικείμενο, το οποίο παρέχεται έμμεσα από την κλήση. Όπως και στο Smalltalk, οι ίδιες οι κλάσεις είναι αντικείμενα. Αυτό παρέχει σημασιολογία για εισαγωγή και μετονομασία. Σε αντίθεση με τις γλώσσες C++ και Modula-3, οι built-in τύποι μπορούν να χρησιμοποιηθούν ως βασικές κλάσεις για επέκταση από τον χρήστη. Επίσης, όπως στην C++, οι περισσότεροι built-in τελεστές με ειδική σύνταξη (αριθμητικοί τελεστές, εγγραφή κ.λπ.) μπορούν να επαναπροσδιοριστούν για τα στιγμιότυπα κλάσης.

(Ελλείπει καθολικής αποδεκτής ορολογίας για να μιλήσω για τις κλάσεις, θα κάνω περιστασιακή χρήση όρων από τη Smalltalk και τη C++. Θα χρησιμοποιούσα όρους από τη Modula-3, καθώς η αντικειμενοστραφής σημασιολογία του είναι πιο κοντά σε αυτήν της Python από ότι της C++, Αλλά πιστεύω ότι λίγοι αναγνώστες το έχουν ακούσει.)

## 9.1 Λίγα λόγια για Ονόματα και Αντικείμενα

Τα αντικείμενα έχουν μοναδικότητα και πολλά ονόματα (σε πολλαπλά πεδία) μπορούν να συνδεθούν στο ίδιο αντικείμενο. Αυτό είναι γνωστό ως ψευδώνυμο σε άλλες γλώσσες. Αυτό συνήθως δεν εκτιμάται με μια πρώτη ματιά στην Python και μπορεί να αγνοείται με ασφάλεια όταν ασχολείται με αμετάβλητους βασικούς τύπους (αριθμοί, συμβολοσειρές, πλειάδες (tuples)). Ωστόσο, το ψευδώνυμο έχει μια πιθανώς εκπληκτική επίδραση στη σημασιολογία του κώδικα της Python που περιλαμβάνει ευμετάβλητα αντικείμενα όπως λίστες, λεξικά, και τους περισσότερους άλλους τύπους. Αυτό χρησιμοποιείται συνήθως προς όφελος του προγράμματος, δεδομένου ότι τα ψευδώνυμα συμπεριφέρονται σαν δείκτες από ορισμένες απόψεις. Για παράδειγμα, η μετάδοση ενός αντικειμένου είναι ανέξοδη αφού μόνο ένας δείκτης περνά από την υλοποίηση, και αν μια συνάρτηση τροποποιεί ένα αντικείμενο που έχει περάσει ως όρισμα, ο καλών θα δει την αλλαγή — αυτό εξαλείφει την ανάγκη για δύο διαφορετικούς μηχανισμούς μετάδοσης ορισμάτων όπως στην Pascal.

## 9.2 Εμβέλεια και Πεδία Ονομάτων στην Python

Πριν από την εισαγωγή των κλάσεων, πρέπει πρώτα να σας πω κάτι για τους κανόνες εμβέλειας της Python. Οι ορισμοί των κλάσεων παίζουν μερικά ξεκάθαρα κόλπα με τα πεδία ονομάτων και πρέπει να γνωρίζετε πώς λειτουργούν πλήρως τα πεδία ονομάτων και η εμβέλεια για να κατανοήσετε πλήρως τι συμβαίνει. Παρεμπιπτόντως, η γνώση για αυτό το θέμα είναι χρήσιμη για κάθε προχωρημένο προγραμματιστή της Python.

Ας ξεκινήσουμε με ορισμένους ορισμούς.

Ένας *πεδίο ονομάτων* (*namespace*) είναι μια αντιστοίχιση από ονόματα σε αντικείμενα. Τα περισσότερα πεδία ονομάτων υλοποιούνται επί του παρόντος ως λεξικά Python, αλλά αυτό συνήθως δεν γίνεται αντιληπτό με κανέναν τρόπο (εκτός από την απόδοση) και μπορεί να αλλάξει στο μέλλον. Παραδείγματα πεδίων ονομάτων είναι: το σύνολο των ενσωματωμένων ονομάτων (που περιέχει συναρτήσεις όπως `abs()` και ενσωματωμένα ονόματα εξαιρέσεων)• τα καθολικά ονόματα σε ένα module και τα τοπικά ονόματα σε μια επίκληση συνάρτησης. Κατά μία έννοια το σύνολο των χαρακτηριστικών ενός αντικειμένου σχηματίζει επίσης ένα πεδίο ονομάτων. Το σημαντικό πράγμα που πρέπει να γνωρίζετε για τα πεδία ονομάτων είναι ότι δεν υπάρχει καμία απολύτως σχέση μεταξύ ονομάτων σε διαφορετικά πεδία ονομάτων, για παράδειγμα, δύο διαφορετικά modules μπορεί και τα δύο να ορίσουν μια συνάρτηση `maximize` χωρίς σύγχυση — χρήστες των modules πρέπει να την προσθέσουν με το όνομα του module.

Παρεμπιπτόντως, χρησιμοποιώ τη λέξη *attribute* για οποιοδήποτε όνομα που ακολουθεί μια τελεία — για παράδειγμα, στην έκφραση `z.real`, το `real` είναι ένα attribute του αντικειμένου `z`. Αυστηρά μιλώντας, οι αναφορές σε ονόματα των modules είναι αναφορές σε attributes: στην έκφραση `modname.funcname`, το `modname` είναι ένα module αντικείμενο και το `funcname` είναι ένα attribute του αντικειμένου. Σε αυτήν την περίπτωση συμβαίνει να υπάρχει μια απλή αντιστοίχιση μεταξύ των attributes των modules και των καθολικών ονομάτων που ορίζονται στο module: μοιράζονται τον ίδιο χώρο ονομάτων!<sup>1</sup>

Attributes may be read-only or writable. In the latter case, assignment to attributes is possible. Module attributes are writable: you can write `modname.the_answer = 42`. Writable attributes may also be deleted with the `del` statement. For example, `del modname.the_answer` will remove the attribute `the_answer` from the object named by `modname`.

Οι χώροι ονομάτων δημιουργούνται σε διαφορετικές στιγμές και έχουν διαφορετική διάρκεια ζωής. Ο χώρος ονομάτων που περιέχει τα built-in ονόματα δημιουργείται κατά την εκκίνηση του διερμηνέα της Python και δεν διαγράφεται ποτέ. Ο καθολικός χώρος ονομάτων για ένα module δημιουργείται όταν διαβάζεται ο ορισμός του module. Κανονικά, οι χώροι ονομάτων των modules διαρκούν επίσης μέχρι να τερματιστεί ο διερμηνέας. Οι δηλώσεις που εκτελούνται από την επίκληση ανώτατου επιπέδου του διερμηνέα, είτε διαβάζονται από ένα script είτε διαδραστικά, θεωρούνται μέρος ενός module που ονομάζεται `__main__`, επομένως έχουν τον δικό τους καθολικό χώρο ονομάτων. (Τα ενσωματωμένα ονόματα στην πραγματικότητα υπάρχουν επίσης σε ένα module, αυτό ονομάζεται `builtins`.)

Ο τοπικός χώρος ονομάτων για μια συνάρτηση δημιουργείται όταν καλείται η συνάρτηση, και διαγράφεται όταν η συνάρτηση επιστρέφει ή δημιουργεί μια εξαίρεση που δεν αντιμετωπίζεται στην συνάρτηση. (Στην

<sup>1</sup> Except for one thing. Module objects have a secret read-only attribute called `__dict__` which returns the dictionary used to implement the module's namespace; the name `__dict__` is an attribute but not a global name. Obviously, using this violates the abstraction of namespace implementation, and should be restricted to things like post-mortem debuggers.

πραγματικότητα, η λήθη θα ήταν καλύτερος τρόπος για να περιγράψουμε τι πραγματικά συμβαίνει.) Φυσικά, οι επαναλαμβανόμενες επικλήσεις έχουν το δικό τους τοπικό χώρο ονομάτων.

Η εμβέλεια είναι μια περιοχή κειμένου ενός προγράμματος Python όπου ένας χώρος ονομάτων είναι άμεσα προσβάσιμος. Το «Άμεση πρόσβαση» εδώ σημαίνει ότι μια ανεπιφύλακτη αναφορά σε ένα όνομα προσπαθεί να βρει το όνομα στον χώρο ονομάτων.

Παρόλο που τα πεδία προσδιορίζονται στατικά, χρησιμοποιούνται δυναμικά. Σε οποιοδήποτε χρόνο κατά την διάρκεια της εκτέλεσης, υπάρχουν 3 ή 4 ένθετα πεδία των οποίων οι χώροι ονομάτων είναι άμεσα προσβάσιμοι:

- η ενδότερη εμβέλεια, η οποία αναζητείται πρώτα, περιέχει τα τοπικά ονόματα
- η εμβέλεια οποιωνδήποτε εσωκλειόμενων συναρτήσεων, τα οποία αναζητούνται ξεκινώντας από την πλησιέστερη εσωκλειόμενη εμβέλεια, περιέχει μη τοπικά, αλλά και μη καθολικά ονόματα
- η επόμενη προς την τελευταία εμβέλεια περιέχει τα τρέχοντα καθολικά ονόματα του module
- η πιο απομακρυσμένη εμβέλεια (που έγινε τελευταία αναζήτηση) είναι ο χώρος ονομάτων που περιέχει built-in ονόματα

Εάν ένα όνομα έχει δηλωθεί ως καθολικό, τότε όλες οι αναφορές και οι εκχωρήσεις πηγαίνουν απευθείας στο επόμενο προς το τελευταίο πεδίο που περιέχει τα καθολικά ονόματα του module. Για την επανασύνδεση μεταβλητών που βρίσκονται εκτός της ενδότερης εμβέλειας, μπορεί να χρησιμοποιηθεί η δήλωση του `nonlocal`. Εάν δεν δηλωθούν ως μη τοπικές, αυτές οι μεταβλητές είναι μόνο για ανάγνωση (μια προσπάθεια εγγραφής σε μια τέτοια μεταβλητή θα δημιουργήσει απλώς μια νέα τοπική μεταβλητή στην ενδότερη εμβέλεια, αφήνοντας αμετάβλητη την εξωτερική μεταβλητή με το ίδιο όνομα).

Συνήθως, η τοπική εμβέλεια παραπέμπει στα τοπικά ονόματα της (κείμενης) τρέχουσας συνάρτησης. Εκτός συναρτήσεων, η τοπική εμβέλεια αναφέρεται στον ίδιο χώρο ονομάτων με την καθολική εμβέλεια: τον χώρο ονομάτων του module. Οι ορισμοί κλάσεων τοποθετούν έναν ακόμη χώρο ονομάτων στην τοπική εμβέλεια.

Είναι σημαντικό να συνειδητοποιήσουμε ότι οι εμβέλειες καθορίζονται με κείμενο: η καθολική εμβέλεια μιας συνάρτησης που ορίζεται σε ένα module είναι ο χώρος ονομάτων αυτού του module ανεξάρτητα από το πού ή με ποιο ψευδώνυμο καλείται η συνάρτηση. Από την άλλη πλευρά, η πραγματική αναζήτηση ονομάτων γίνεται δυναμικά, κατά το χρόνο εκτέλεσης — ωστόσο, ο ορισμός της γλώσσας εξελίσσεται προς τη στατική ανάλυση ονομάτων, την ώρα της «μεταγλώττισης», επομένως μην βασίζεστε σε δυναμική ανάλυση ονόματος! (Στην πραγματικότητα, οι τοπικές μεταβλητές έχουν ήδη καθοριστεί στατικά.)

Μια ιδιαίτερη ιδιορρυθμία της Python είναι ότι – αν οι δηλώσεις `global` ή `nonlocal` δεν είναι σε ισχύ – οι εκχωρήσεις στα ονόματα πηγαίνουν πάντα στην ενδότερη εμβέλεια. Οι εκχωρήσεις δεν αντιγράφουν δεδομένα — απλώς δεσμεύουν ονόματα σε αντικείμενα. Το ίδιο ισχύει και για τις διαγραφές: η δήλωση `del x` αφαιρεί την σύνδεση του `x` από τον χώρο ονομάτων που αναφέρεται από την τοπική εμβέλεια. Στην πραγματικότητα, όλες οι λειτουργίες που εισάγουν νέα ονόματα χρησιμοποιούν την τοπική εμβέλεια: συγκεκριμένα οι δηλώσεις, `import` και οι ορισμοί συναρτήσεων δεσμεύουν το όνομα του module ή της συνάρτησης στην τοπική εμβέλεια.

Η δήλωση `global` μπορεί να χρησιμοποιηθεί για να υποδείξει ότι συγκεκριμένες μεταβλητές ζουν στην καθολική εμβέλεια και θα πρέπει να ανακάμψουν εκεί. Η δήλωση `nonlocal` υποδηλώνει ότι συγκεκριμένες μεταβλητές ζουν σε μια εσωκλειστή εμβέλεια και θα πρέπει να ανακάμψουν εκεί.

### 9.2.1 Παράδειγμα Εμβέλειας και Χώρων Ονομάτων

Αυτό είναι ένα παράδειγμα που δείχνει τον τρόπο αναφοράς στα διαφορετικά πεδία και χώρους ονομάτων και πώς τα `global` και `nonlocal` επηρεάζουν τα variable binding:

```
def scope_test():
    def do_local():
        spam = "local spam"

    def do_nonlocal():
        nonlocal spam
        spam = "nonlocal spam"
```

(συνέχεια στην επόμενη σελίδα)

(συνεχίζεται από την προηγούμενη σελίδα)

```
def do_global():
    global spam
    spam = "global spam"

spam = "test spam"
do_local()
print("After local assignment:", spam)
do_nonlocal()
print("After nonlocal assignment:", spam)
do_global()
print("After global assignment:", spam)

scope_test()
print("In global scope:", spam)
```

Το αποτέλεσμα του κώδικα στο παράδειγμα είναι:

```
After local assignment: test spam
After nonlocal assignment: nonlocal spam
After global assignment: nonlocal spam
In global scope: global spam
```

Σημειώστε πώς η εκχώρηση *τοπική* (η οποία είναι προεπιλεγμένη) δεν άλλαξε την δέσμευση *scope\_test*'s του *spam*. Η εκχώρηση *nonlocal* άλλαξε την δέσμευση του *scope\_test*'s του *spam* και η εκχώρηση του *global* άλλαξε τη δέσμευση σε επίπεδο *module*.

Μπορείτε επίσης να δείτε ότι δεν υπήρχε προηγούμενη δέσμευση για *spam* πριν από την εκχώρηση *global*..

## 9.3 Μια πρώτη ματιά στις Κλάσεις

Οι Κλάσεις εισάγουν λίγη νέα σύνταξη, τρεις νέους τύπους αντικειμένων και κάποια νέα σημασιολογία.

### 9.3.1 Σύνταξη Ορισμού Κλάσης

Η απλούστερη μορφή ορισμού κλάσης μοιάζει με αυτό:

```
class ClassName:
    <statement-1>
    .
    .
    .
    <statement-N>
```

Ορισμοί κλάσεων, όπως ορισμοί συναρτήσεων (*def* δηλώσεις) πρέπει να εκτελεστούν προτού έχουν οποιοδήποτε αποτέλεσμα. (Θα μπορούσατε να τοποθετήσετε έναν ορισμό κλάσης σε έναν κλάδο μιας δήλωσης *if* ή μέσα σε μια συνάρτηση.)

Στην πράξη, οι δηλώσεις μέσα σε έναν ορισμό κλάσης συνήθως θα είναι ορισμοί συναρτήσεων, αλλά επιτρέπονται άλλες δηλώσεις και μερικές φορές χρήσιμες — θα επανέλθουμε σε αυτό αργότερα. Οι ορισμοί συναρτήσεων μέσα σε μια κλάση συνήθως έχουν μια περίεργη μορφή λίστας ορισμάτων, που υπαγορεύεται από τις συμβάσεις κλήσης για μεθόδους — και πάλι, αυτό εξηγείται αργότερα.

Όταν εισάγεται ένας ορισμός κλάσης, δημιουργείται ένας νέος χώρος ονομάτων και χρησιμοποιείται ως τοπική εμβέλεια — επομένως, όλες οι εκχωρήσεις σε τοπικές μεταβλητές πηγαίνουν σε αυτόν τον νέο χώρο ονομάτων. Συγκεκριμένα, οι ορισμοί συναρτήσεων δεσμεύουν το όνομα της νέας συνάρτησης εδώ.

When a class definition is left normally (via the end), a *class object* is created. This is basically a wrapper around the contents of the namespace created by the class definition; we'll learn more about class objects in the next section. The

original local scope (the one in effect just before the class definition was entered) is reinstated, and the class object is bound here to the class name given in the class definition header (ClassName in the example).

### 9.3.2 Αντικείμενα Κλάσης

Τα αντικείμενα κλάσης υποστηρίζουν δύο είδη πράξεων: αναφορές χαρακτηριστικών και στιγμίοτυπο.

Οι Αναφορές χαρακτηριστικών χρησιμοποιούν την τυπική σύνταξη που χρησιμοποιείται για όλα τις αναφορές χαρακτηριστικών στην Python: `obj.name`. Τα έγκυρα ονόματα χαρακτηριστικών είναι όλα τα ονόματα που βρίσκονταν στον χώρο ονομάτων της κλάσης όταν δημιουργήθηκε το αντικείμενο της κλάσης. Έτσι, αν ο ορισμός της κλάσης έμοιαζε ως εξής:

```
class MyClass:
    """A simple example class"""
    i = 12345

    def f(self):
        return 'hello world'
```

then `MyClass.i` and `MyClass.f` are valid attribute references, returning an integer and a function object, respectively. Class attributes can also be assigned to, so you can change the value of `MyClass.i` by assignment. `__doc__` is also a valid attribute, returning the docstring belonging to the class: "A simple example class".

Η κλάση *στιγμίοτυπο* χρησιμοποιεί σημειογραφία συνάρτησης. Απλώς προσποιηθείτε ότι το αντικείμενο της κλάσης είναι μια συνάρτηση χωρίς παραμέτρους που επιστρέφει ένα νέο στιγμίοτυπο της κλάσης. Για παράδειγμα (υποθέτοντας την παραπάνω κλάση):

```
x = MyClass()
```

δημιουργεί ένα νέο *στιγμίοτυπο* της κλάσης και εκχωρεί αυτό το αντικείμενο στην τοπική μεταβλητή `x`.

The instantiation operation («calling» a class object) creates an empty object. Many classes like to create objects with instances customized to a specific initial state. Therefore a class may define a special method named `__init__()`, like this:

```
def __init__(self):
    self.data = []
```

When a class defines an `__init__()` method, class instantiation automatically invokes `__init__()` for the newly created class instance. So in this example, a new, initialized instance can be obtained by:

```
x = MyClass()
```

Of course, the `__init__()` method may have arguments for greater flexibility. In that case, arguments given to the class instantiation operator are passed on to `__init__()`. For example,

```
>>> class Complex:
...     def __init__(self, realpart, imagpart):
...         self.r = realpart
...         self.i = imagpart
...
>>> x = Complex(3.0, -4.5)
>>> x.r, x.i
(3.0, -4.5)
```

### 9.3.3 Αντικείμενα Στιγμιότυπων

Τώρα τι μπορούμε να κάνουμε με τα αντικείμενα στιγμιότυπων; Οι μόνες λειτουργίες που γίνονται κατανοητές από τα αντικείμενα στιγμιότυπων είναι οι αναφορές χαρακτηριστικών. Υπάρχουν δύο είδη έγκυρων ονομάτων attributes: attributes και μέθοδοι δεδομένων.

*Data attributes* correspond to «instance variables» in Smalltalk, and to «data members» in C++. Data attributes need not be declared; like local variables, they spring into existence when they are first assigned to. For example, if `x` is the instance of `MyClass` created above, the following piece of code will print the value 16, without leaving a trace:

```
x.counter = 1
while x.counter < 10:
    x.counter = x.counter * 2
print(x.counter)
del x.counter
```

The other kind of instance attribute reference is a *method*. A method is a function that «belongs to» an object. (In Python, the term method is not unique to class instances: other object types can have methods as well. For example, list objects have methods called `append`, `insert`, `remove`, `sort`, and so on. However, in the following discussion, we'll use the term method exclusively to mean methods of class instance objects, unless explicitly stated otherwise.)

Τα έγκυρα ονόματα μεθόδων ενός αντικειμένου στιγμιότυπου εξαρτώνται από την κλάση του. Εξ ορισμού, όλα τα χαρακτηριστικά μιας κλάσης που είναι αντικείμενα συνάρτησης ορίζουν τις αντίστοιχες μεθόδους των στιγμιότυπων της. Έτσι στο παράδειγμά μας, το `x.f` είναι μια έγκυρη αναφορά μεθόδου, αφού το `MyClass.f` είναι συνάρτηση, αλλά το `x.i` δεν είναι αφού το `MyClass.i` δεν είναι. Αλλά το `x.f` δεν είναι το ίδιο πράγμα με το `MyClass.f` — είναι ένα αντικείμενο μεθόδου, όχι ένα αντικείμενο συνάρτησης.

### 9.3.4 Αντικείμενα Μεθόδου

Συνήθως, μια μέθοδος καλείται αμέσως μετά τη δέσμευσή της:

```
x.f()
```

In the `MyClass` example, this will return the string `'hello world'`. However, it is not necessary to call a method right away: `x.f` is a method object, and can be stored away and called at a later time. For example:

```
xf = x.f
while True:
    print(xf())
```

Θα συνεχίσει να εκτυπώνει το `hello world` μέχρι το τέλος του χρόνου.

What exactly happens when a method is called? You may have noticed that `x.f()` was called without an argument above, even though the function definition for `f()` specified an argument. What happened to the argument? Surely Python raises an exception when a function that requires an argument is called without any — even if the argument isn't actually used...

Στην πραγματικότητα, μπορεί να έχετε μαντέψει την απάντηση: το ιδιαίτερο με τις μεθόδους είναι ότι το αντικείμενο του στιγμιότυπου μεταβιβάζεται ως το πρώτο όρισμα της συνάρτησης. Στο παράδειγμά μας, η κλήση `x.f()` είναι ακριβώς ισοδύναμη με το `MyClass.f(x)`. Γενικά, η κλήση μιας μεθόδου με μια λίστα από  $n$  ορίσματα ισοδυναμεί με την κλήση της αντίστοιχης συνάρτησης με μια λίστα ορισμάτων που δημιουργείται με την εισαγωγή του αντικειμένου στιγμιότυπου της μεθόδου πριν από το πρώτο όρισμα.

If you still don't understand how methods work, a look at the implementation can perhaps clarify matters. When a non-data attribute of an instance is referenced, the instance's class is searched. If the name denotes a valid class attribute that is a function object, a method object is created by packing (pointers to) the instance object and the function object just found together in an abstract object: this is the method object. When the method object is called with an argument list, a new argument list is constructed from the instance object and the argument list, and the function object is called with this new argument list.

### 9.3.5 Μεταβλητές Κλάσης και Στιγμιότυπου

Σε γενικές γραμμές, οι μεταβλητές στιγμιότυπου προορίζονται για δεδομένα μοναδικά για κάθε στιγμιότυπο και οι μεταβλητές κλάσης είναι για χαρακτηριστικά και μεθόδους που μοιράζονται όλα τα στιγμιότυπα της κλάσης:

```
class Dog:

    kind = 'canine'           # class variable shared by all instances

    def __init__(self, name):
        self.name = name     # instance variable unique to each instance

>>> d = Dog('Fido')
>>> e = Dog('Buddy')
>>> d.kind                # shared by all dogs
'canine'
>>> e.kind                # shared by all dogs
'canine'
>>> d.name                # unique to d
'Fido'
>>> e.name                # unique to e
'Buddy'
```

Όπως συζητήθηκε στο *Λίγα λόγια για Ονόματα και Αντικείμενα*, τα κοινά δεδομένα μπορεί να έχουν πιθανώς εκπληκτικά αποτελέσματα με τη συμμετοχή αντικειμένων *mutable* όπως λίστες και λεξικά. Για παράδειγμα, η λίστα *tricks* στον παρακάτω κώδικα δεν θα πρέπει να χρησιμοποιείται ως μεταβλητή κλάσης επειδή μόνο μία λίστα θα μπορούσε να είναι κοινή σε όλα τα στιγμιότυπα *Dog*:

```
class Dog:

    tricks = []             # mistaken use of a class variable

    def __init__(self, name):
        self.name = name

    def add_trick(self, trick):
        self.tricks.append(trick)

>>> d = Dog('Fido')
>>> e = Dog('Buddy')
>>> d.add_trick('roll over')
>>> e.add_trick('play dead')
>>> d.tricks                # unexpectedly shared by all dogs
['roll over', 'play dead']
```

Ο σωστός σχεδιασμός της κλάσης θα πρέπει να χρησιμοποιεί μια μεταβλητή στιγμιότυπου αντί:

```
class Dog:

    def __init__(self, name):
        self.name = name
        self.tricks = []    # creates a new empty list for each dog

    def add_trick(self, trick):
        self.tricks.append(trick)

>>> d = Dog('Fido')
>>> e = Dog('Buddy')
>>> d.add_trick('roll over')
>>> e.add_trick('play dead')
>>> d.tricks
```

(συνέχεια στην επόμενη σελίδα)



(συνεχίζεται από την προηγούμενη σελίδα)

```
['roll over']
>>> e.tricks
['play dead']
```

## 9.4 Τυχαίες Παρατηρήσεις

Αν το ίδιο όνομα χαρακτηριστικού εμφανίζεται και σε ένα στιγμιότυπο και σε μια κλάση, τότε η αναζήτηση χαρακτηριστικών δίνει προτεραιότητα στο στιγμιότυπο:

```
>>> class Warehouse:
...     purpose = 'storage'
...     region = 'west'
...
>>> w1 = Warehouse()
>>> print(w1.purpose, w1.region)
storage west
>>> w2 = Warehouse()
>>> w2.region = 'east'
>>> print(w2.purpose, w2.region)
storage east
```

Τα χαρακτηριστικά δεδομένων μπορούν να αναφέρονται με μεθόδους καθώς και από απλούς χρήστες («πελάτες») ενός αντικειμένου. Με άλλα λόγια, οι κλάσεις δεν μπορούν να χρησιμοποιηθούν για την υλοποίηση καθαρών αφηρημένων τύπων δεδομένων. Στην πραγματικότητα, τίποτα στην Python δεν καθιστά δυνατή την επιβολή της απόκρυψης δεδομένων — όλα βασίζονται σε σύμβαση. (από την άλλη πλευρά, η εφαρμογή Python, γραμμένη σε C, μπορεί να αποκρύψει εντελώς τις λεπτομέρειες υλοποίησης και να ελέγξει την πρόσβαση σε ένα αντικείμενο εάν είναι απαραίτητο αυτό μπορεί να χρησιμοποιηθεί από επεκτάσεις στην Python γραμμένες σε C.)

Οι χρήστες θα πρέπει να χρησιμοποιούν τα χαρακτηριστικά δεδομένων με προσοχή — οι χρήστες ενδέχεται να μπερδέψουν τα αμετάβλητα που διατηρούνται από τις μεθόδους σφραγίζοντας τα χαρακτηριστικά των δεδομένων τους. Λάβετε υπόψη ότι οι χρήστες μπορούν να προσθέσουν δικά τους χαρακτηριστικά δεδομένων σε ένα αντικείμενο στιγμιότυπου χωρίς να επηρεάσουν την εγκυρότητα των μεθόδων, εφόσον αποφεύγονται οι συγκρούσεις ονομάτων — και πάλι, μια σύμβαση ονομασίας μπορεί να σώσει πολλούς πονοκεφάλους εδώ.

Δεν υπάρχει συντομογραφία για την αναφορά χαρακτηριστικών δεδομένων (ή άλλων μεθόδων!) μέσα από τις μεθόδους. Διαπιστώνω ότι αυτό στην πραγματικότητα αυξάνει την αναγνωσιμότητα των μεθόδων: δεν υπάρχει καμία πιθανότητα σύγχυσης τοπικών μεταβλητών και των μεταβλητών παραδείγματος όταν εξετάζουμε μια μέθοδο.

Συχνά, το πρώτο όρισμα μιας μεθόδου ονομάζεται *self*. Αυτό δεν είναι τίποτα περισσότερο από μια σύμβαση: το όνομα *self* δεν έχει καμία απολύτως ιδιαίτερη σημασία για την Python. Σημειώστε, ωστόσο, ότι αν δεν ακολουθήσετε τη σύμβαση ο κώδικάς σας μπορεί να είναι λιγότερο ευανάγνωστος σε άλλους προγραμματιστές Python, και είναι επίσης κατανοητό ότι μπορεί να γραφτεί ένα πρόγραμμα *class browser* που να βασίζεται σε μια τέτοια σύμβαση.

Κάθε αντικείμενο συνάρτησης που είναι χαρακτηριστικό κλάσης ορίζει μια μέθοδο για στιγμιότυπα αυτής της κλάσης. Δεν είναι απαραίτητο ο ορισμός της συνάρτησης να περικλείεται με κείμενο στον ορισμό της κλάσης: η αντιστοίχιση ενός αντικειμένου συνάρτησης σε μια τοπική μεταβλητή της κλάσης είναι επίσης εντάξει. Για παράδειγμα:

```
# Function defined outside the class
def f1(self, x, y):
    return min(x, x+y)

class C:
    f = f1
```

(συνέχεια στην επόμενη σελίδα)



(συνεχίζεται από την προηγούμενη σελίδα)

```
def g(self):
    return 'hello world'

h = g
```

Now `f`, `g` and `h` are all attributes of class `C` that refer to function objects, and consequently they are all methods of instances of `C` — `h` being exactly equivalent to `g`. Note that this practice usually only serves to confuse the reader of a program.

Οι μέθοδοι μπορούν να καλούν άλλες μεθόδους χρησιμοποιώντας χαρακτηριστικά μεθόδου του argument `self`:

```
class Bag:
    def __init__(self):
        self.data = []

    def add(self, x):
        self.data.append(x)

    def addtwice(self, x):
        self.add(x)
        self.add(x)
```

Οι μέθοδοι μπορεί να αναφέρονται σε καθολικά ονόματα με τον ίδιο τρόπο όπως οι συνηθισμένες συναρτήσεις. Η καθολική εμβέλεια που σχετίζεται με μια μέθοδο είναι το module που περιέχει τον ορισμό της. (Μια κλάση δεν χρησιμοποιείται ποτέ ως καθολική εμβέλεια.) Αν και σπάνια συναντά κανείς έναν καλό λόγο για τη χρήση καθολικών δεδομένων σε μια μέθοδο, υπάρχουν πολλές νόμιμες χρήσεις της καθολικής εμβέλειας: για ένα πράγμα, οι λειτουργίες και οι λειτουργικές μονάδες που εισάγονται στην καθολική εμβέλεια μπορούν να χρησιμοποιηθούν από μεθόδους, καθώς και συναρτήσεις και κλάσεις που ορίζονται σε αυτό. Συνήθως, η κλάση που περιέχει τη μέθοδο ορίζεται από μόνη της σε αυτή την καθολική εμβέλεια, και στην επόμενη ενότητα θα βρούμε μερικούς καλούς λόγους για τους οποίους μια μέθοδος θα ήθελε να αναφέρει τη δική της κλάση.

Κάθε τιμή είναι ένα αντικείμενο και επομένως έχει μια κλάση (ονομάζεται επίσης *τύπος* της). Αποθηκεύεται ως `object.__class__`.

## 9.5 Κληρονομικότητα

Φυσικά, ένα χαρακτηριστικό γλώσσας δεν θα ήταν αντάξιο του ονόματος «class» χωρίς την υποστήριξη της κληρονομικότητας. Η σύνταξη για έναν παραγόμενο ορισμό κλάσης μοιάζει με αυτό:

```
class DerivedClassName(BaseClassName):
    <statement-1>
    .
    .
    .
    <statement-N>
```

The name `BaseClassName` must be defined in a scope containing the derived class definition. In place of a base class name, other arbitrary expressions are also allowed. This can be useful, for example, when the base class is defined in another module:

```
class DerivedClassName(modname.BaseClassName):
```

Η εκτέλεση ενός παραγόμενου ορισμού κλάσης προχωρά το ίδιο όπως για μια βασική κλάση. Όταν το αντικείμενο της κλάσης κατασκευάζεται, η βασική κλάση απομνημονεύεται. Αυτό χρησιμοποιείται για την επίλυση αναφορών χαρακτηριστικών: εάν ένα ζητούμενο χαρακτηριστικό δεν βρεθεί στην κλάση, η αναζήτηση

προχωρά στην αναζήτηση στη βασική κλάση. Αυτός ο κανόνας εφαρμόζεται αναδρομικά εάν η ίδια η βασική κλάση προέρχεται από κάποια άλλη κλάση.

Δεν υπάρχει τίποτα το ιδιαίτερο σχετικά με την δημιουργία στιγμιότυπου παραγόμενων κλάσεων: `DerivedClassName()` δημιουργεί ένα νέο στιγμιότυπο της κλάσης. Οι αναφορές μεθόδων επιλύονται ως εξής: γίνεται αναζήτηση του αντίστοιχου χαρακτηριστικού κλάσης, κατεβαίνοντας προς τα κάτω στην αλυσίδα των βασικών κλάσεων εάν είναι απαραίτητο, και η αναφορά της μεθόδου είναι έγκυρη εάν αυτό αποδίδει ένα αντικείμενο συνάρτησης.

Οι παράγωγες κλάσεις ενδέχεται να παρακάμπτουν τις μεθόδους των βασικών τους κλάσεων. Επειδή οι μέθοδοι δεν έχουν ειδικά προνόμια όταν καλούν άλλες μεθόδους του ίδιου αντικειμένου, μια μέθοδος μιας βασικής κλάσης που καλεί μια άλλη μέθοδο που ορίζεται στην ίδια βασική κλάση μπορεί να καταλήξει να καλεί μια μέθοδο μιας παραγόμενης κλάσης που την αντικαθιστά. (Για προγραμματιστές C++: όλες οι μέθοδοι στην Python είναι ουσιαστικά «εικονικές».)

Μια υπερισχύουσα μέθοδος σε μια παραγόμενη κλάση μπορεί στην πραγματικότητα να θέλει να επεκτείνει αντί να αντικαταστήσει απλώς τη μέθοδο βασικής κλάσης με το ίδιο όνομα. Υπάρχει ένας απλός τρόπος για να καλέσετε τη μέθοδο βασικής κλάσης απευθείας: απλώς καλέστε το `BaseClassName.methodname(self, arguments)`. Αυτό είναι περιστασιακά χρήσιμο στους χρήστες (Λάβετε υπόψη ότι αυτό λειτουργεί μόνο εάν η βασική κλάση είναι προσβάσιμη ως `BaseClassName` στην καθολική εμβέλεια.)

Η Python έχει δύο (ενσωματωμένες) built-in συναρτήσεις που λειτουργούν με κληρονομικότητα:

- Χρησιμοποιήστε το `isinstance()` για να ελέγξετε τον τύπο ενός στιγμιότυπου: το `isinstance(obj, int)` θα είναι `True` μόνο εάν το `obj.__class__` είναι `int` ή προέρχεται από κάποια κλάση από `int`.
- Χρησιμοποιήστε το `issubclass()` για να ελέγξετε την κληρονομικότητα κλάσης: Το `issubclass(bool, int)` είναι `True` αφού το `bool` είναι υποκλάση του `int`. Ωστόσο, το `issubclass(float, int)` είναι `False` αφού το `float` δεν είναι υποκλάση του `int`.

## 9.5.1 Πολλαπλή Κληρονομικότητα

Η Python υποστηρίζει επίσης μια μορφή πολλαπλής κληρονομικότητας. Ένας ορισμός κλάσης με πολλαπλές βασικές κλάσεις μοιάζει με αυτό:

```
class DerivedClassName(Base1, Base2, Base3):
    <statement-1>
    .
    .
    .
    <statement-N>
```

For most purposes, in the simplest cases, you can think of the search for attributes inherited from a parent class as depth-first, left-to-right, not searching twice in the same class where there is an overlap in the hierarchy. Thus, if an attribute is not found in `DerivedClassName`, it is searched for in `Base1`, then (recursively) in the base classes of `Base1`, and if it was not found there, it was searched for in `Base2`, and so on.

Στην πραγματικότητα, είναι λίγο πιο περίπλοκο από αυτό. Η σειρά ανάλυσης της μεθόδου αλλάζει δυναμικά για να υποστηρίξει συνεργατικές κλήσεις σε `super()`. Αυτή η προσέγγιση είναι γνωστή σε ορισμένες άλλες γλώσσες πολλαπλής κληρονομικότητας ως `call-next-method` και είναι πιο ισχυρή από τη σούπερ κλήση που βρίσκεται σε γλώσσες μεμονωμένης κληρονομικότητας.

Dynamic ordering is necessary because all cases of multiple inheritance exhibit one or more diamond relationships (where at least one of the parent classes can be accessed through multiple paths from the bottommost class). For example, all classes inherit from `object`, so any case of multiple inheritance provides more than one path to reach `object`. To keep the base classes from being accessed more than once, the dynamic algorithm linearizes the search order in a way that preserves the left-to-right ordering specified in each class, that calls each parent only once, and that is monotonic (meaning that a class can be subclassed without affecting the precedence order of its parents). Taken together, these properties make it possible to design reliable and extensible classes with multiple inheritance. For more detail, see <https://www.python.org/download/releases/2.3/mro/>.

## 9.6 Ιδιωτικές Μεταβλητές

Οι μεταβλητές στιγμιότυπου «Private» στις οποίες δεν είναι δυνατή η πρόσβαση εκτός από το εσωτερικό ενός αντικειμένου, δεν υπάρχουν στην Python. Ωστόσο, υπάρχει μια σύμβαση που ακολουθείται από τον περισσότερο Python κώδικα: ένα όνομα με πρόθεμα κάτω παύλα (π.χ. `_spam`) θα πρέπει να αντιμετωπίζεται ως μη δημόσιο μέρος του API (είτε πρόκειται για συνάρτηση, μέθοδο ή μέλος δεδομένων). Θα πρέπει να θεωρείται ως λεπτομέρεια υλοποίησης και υπόκειται σε αλλαγές χωρίς προειδοποίηση.

Δεδομένου ότι υπάρχει μια έγκυρη περίπτωση χρήσης για ιδιωτικά μέλη της κλάσης (δηλαδή για να αποφευχθούν συγκρούσεις ονομάτων με ονόματα που ορίζονται από υποκλάσεις), υπάρχει περιορισμένη υποστήριξη για έναν τέτοιο μηχανισμό, που ονομάζεται *name mangling*. Οποιοδήποτε αναγνωριστικό της φόρμας `__spam` (τουλάχιστον δύο προπορευόμενες κάτω παύλες, το πολύ μια στη συνέχεια κάτω παύλα) αντικαθίσταται με κείμενο με το `__classname__spam`, όπου το `classname` είναι το όνομα της τρέχουσας τάξης με την πρώτη υπογράμμιση *stripped*. Αυτό το mangling γίνεται χωρίς να λαμβάνεται υπόψη η συντακτική θέση του του αναγνωριστικού, αρκεί να εμφανίζεται εντός του ορισμού μιας κλάσης.

Η παραβίαση ονομάτων είναι χρήσιμη για να επιτρέπεται στις υποκλάσεις να παρακάμπτουν μεθόδους χωρίς να διακόπτουν τις κλήσεις μεθόδων ενδοκλάσεων. Για παράδειγμα:

```
class Mapping:
    def __init__(self, iterable):
        self.items_list = []
        self.__update(iterable)

    def update(self, iterable):
        for item in iterable:
            self.items_list.append(item)

    __update = update    # private copy of original update() method

class MappingSubclass(Mapping):

    def update(self, keys, values):
        # provides new signature for update()
        # but does not break __init__()
        for item in zip(keys, values):
            self.items_list.append(item)
```

Το παραπάνω παράδειγμα θα λειτουργούσε ακόμα και αν το `MappingSubclass` εισήγαγε ένα αναγνωριστικό `__update` αφού αντικαταστάθηκε με το `__Mapping__update` στην κλάση `Mapping` και με το `__MappingSubclass__update` στη κλάση `MappingSubclass` αντίστοιχα.

Λάβετε υπόψη ότι οι κανόνες παραβίασης έχουν σχεδιαστεί κυρίως για την αποφυγή ατυχημάτων. Εξακολουθεί να είναι δυνατή η πρόσβαση ή η τροποποίηση μιας μεταβλητής που θεωρείται ιδιωτική. Αυτό μπορεί να είναι χρήσιμο ακόμη και σε ειδικές περιπτώσεις, όπως στο πρόγραμμα εντοπισμού σφαλμάτων (debugger).

Σημειώστε ότι ο κώδικας που μεταβιβάστηκε στο `exec()` ή στο `eval()` δεν θεωρεί ότι το το όνομα κλάσης της κλάσης επίκλησης να είναι η τρέχουσα κλάση. Αυτό είναι παρόμοιο με το αποτέλεσμα της καθολικής δήλωσης, το αποτέλεσμα της οποίας επίσης περιορίζεται στον κώδικα που έχει μεταγλωττιστεί μαζί (byte-compiled). Ο ίδιος περιορισμός ισχύει για τα `getattr()`, `setattr()` και `delattr()`, καθώς και όταν γίνεται αναφορά απευθείας στο `__dict__`.

## 9.7 Μικροπράγματα

Μερικές φορές είναι χρήσιμο να έχετε έναν τύπο δεδομένων παρόμοιο με τον Pascal «record» ή C «struct», ομαδοποιώντας μερικά επώνυμα στοιχεία δεδομένων. Η ιδιωματική προσέγγιση είναι η χρήση `dataclasses` για αυτόν τον σκοπό:

```
from dataclasses import dataclass

@dataclass
class Employee:
    name: str
    dept: str
    salary: int
```

```
>>> john = Employee('john', 'computer lab', 1000)
>>> john.dept
'computer lab'
>>> john.salary
1000
```

A piece of Python code that expects a particular abstract data type can often be passed a class that emulates the methods of that data type instead. For instance, if you have a function that formats some data from a file object, you can define a class with methods `read()` and `readline()` that get the data from a string buffer instead, and pass it as an argument.

Instance method objects have attributes, too: `m.__self__` is the instance object with the method `m()`, and `m.__func__` is the function object corresponding to the method.

## 9.8 Επαναλήπτες

Μέχρι τώρα πιθανότατα έχετε παρατηρήσει ότι τα περισσότερα αντικείμενα container μπορούν να επαναληφθούν χρησιμοποιώντας μια δήλωση `for`:

```
for element in [1, 2, 3]:
    print(element)
for element in (1, 2, 3):
    print(element)
for key in {'one':1, 'two':2}:
    print(key)
for char in "123":
    print(char)
for line in open("myfile.txt"):
    print(line, end='')
```

Αυτό το στυλ πρόσβασης είναι σαφές, συνοπτικό και βολικό. Η χρήση των Iterators διαπερνά και ενοποιεί την Python. Στο παρασκήνιο, η δήλωση `for` καλεί `iter()` στο αντικείμενο container. Η συνάρτηση επιστρέφει ένα αντικείμενο iterator που ορίζει τη μέθοδο `__next__()` η οποία έχει πρόσβαση σε στοιχεία στο container ένα κάθε φορά. Όταν δεν υπάρχουν άλλα στοιχεία, το `__next__()` δημιουργεί μια `StopIteration` εξαίρεση που λέει τον βρόχο `for` να τερματιστεί. Μπορείτε να καλέσετε τη μέθοδο `__next__()` χρησιμοποιώντας την ενσωματωμένη συνάρτηση `next()`. Αυτό το παράδειγμα δείχνει πώς λειτουργούν όλα:

```
>>> s = 'abc'
>>> it = iter(s)
>>> it
<str_iterator object at 0x10c90e650>
>>> next(it)
'a'
>>> next(it)
```

(συνέχεια στην επόμενη σελίδα)

(συνεχίζεται από την προηγούμενη σελίδα)

```
'b'
>>> next(it)
'c'
>>> next(it)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
    next(it)
StopIteration
```

Having seen the mechanics behind the iterator protocol, it is easy to add iterator behavior to your classes. Define an `__iter__()` method which returns an object with a `__next__()` method. If the class defines `__next__()`, then `__iter__()` can just return `self`:

```
class Reverse:
    """Iterator for looping over a sequence backwards."""
    def __init__(self, data):
        self.data = data
        self.index = len(data)

    def __iter__(self):
        return self

    def __next__(self):
        if self.index == 0:
            raise StopIteration
        self.index = self.index - 1
        return self.data[self.index]
```

```
>>> rev = Reverse('spam')
>>> iter(rev)
<__main__.Reverse object at 0x00A1DB50>
>>> for char in rev:
...     print(char)
...
m
a
p
s
```

## 9.9 Γεννήτορες (Generators)

*Generators* είναι ένα απλό και ισχυρό εργαλείο για τη δημιουργία iterators. Είναι γραμμένες σαν κανονικές συναρτήσεις αλλά χρησιμοποιούν τη `yield` όποτε θέλουν να επιστρέψουν δεδομένα. Κάθε φορά που καλείται `next()` σε αυτό, ο generator συνεχίζει από εκεί που σταμάτησε (θυμάται όλες τις τιμές δεδομένων και ποια δήλωση εκτελέστηκε τελευταία). Ένα παράδειγμα δείχνει ότι οι generators μπορεί να είναι ασήμαντα εύκολο να δημιουργηθούν:

```
def reverse(data):
    for index in range(len(data)-1, -1, -1):
        yield data[index]
```

```
>>> for char in reverse('golf'):
...     print(char)
...
f
l
o
g
```

Anything that can be done with generators can also be done with class-based iterators as described in the previous section. What makes generators so compact is that the `__iter__()` and `__next__()` methods are created automatically.

Ένα άλλο βασικό χαρακτηριστικό είναι ότι οι τοπικές μεταβλητές και η κατάσταση εκτέλεσης αποθηκεύονται αυτόματα μεταξύ των κλήσεων. Αυτό έκανε τη συνάρτηση πιο εύκολη στην γραφή και πολύ πιο ξεκάθαρη από μια προσέγγιση που χρησιμοποιεί μεταβλητές παράδειγμα όπως `self.index` και `self.data`.

Εκτός από την αυτόματη δημιουργία μεθόδου και την αποθήκευση της κατάστασης του προγράμματος, όταν οι generators τερματίζονται, εγείρουν αυτόματα την εξαίρεση `StopIteration`. Σε συνδυασμό, αυτά τα χαρακτηριστικά καθιστούν εύκολη τη δημιουργία επαναλήψεων χωρίς περισσότερη προσπάθεια από τη σύνταξη μιας κανονικής συνάρτησης.

## 9.10 Εκφράσεις Γεννητόρων

Ορισμένοι απλοί generators μπορούν να κωδικοποιηθούν συνοπτικά ως εκφράσεις χρησιμοποιώντας μια σύνταξη παρόμοια με τις `list comprehensions`, αλλά με παρενθέσεις αντί για αγκύλες. Αυτές οι εκφράσεις έχουν σχεδιαστεί για καταστάσεις όπου ο generator χρησιμοποιείται αμέσως από μια περικλείουσα συνάρτηση. Οι εκφράσεις generator είναι πιο συμπαγείς αλλά λιγότερο ευέλικτες από τους ορισμούς πλήρους generator και τείνουν να είναι περισσότερο φιλικό προς τη μνήμη από αντίστοιχα `list comprehensions`.

Παραδείγματα:

```
>>> sum(i*i for i in range(10))           # sum of squares
285

>>> xvec = [10, 20, 30]
>>> yvec = [7, 5, 3]
>>> sum(x*y for x,y in zip(xvec, yvec))   # dot product
260

>>> unique_words = set(word for line in page for word in line.split())

>>> valedictorian = max((student.gpa, student.name) for student in graduates)

>>> data = 'golf'
>>> list(data[i] for i in range(len(data)-1, -1, -1))
['f', 'l', 'o', 'g']
```

### Υποσημειώσεις

---

## Σύντομη ξενάγηση στην Standard Βιβλιοθήκη

---

### 10.1 Διεπαφή Λειτουργικού Συστήματος

Το module `os` παρέχει δεκάδες λειτουργίες για αλληλεπίδραση με το λειτουργικό σύστημα:

```
>>> import os
>>> os.getcwd()           # Return the current working directory
'C:\\Python310'
>>> os.chdir('/server/accesslogs') # Change current working directory
>>> os.system('mkdir today')      # Run the command mkdir in the system shell
0
```

Βεβαιωθείτε ότι χρησιμοποιείτε το `import os` αντί για το `from os import *`. Αυτό θα κρατήσει το `os.open()` υπό τη σκίαση της ενσωματωμένης συνάρτησης `open()` που λειτουργεί πολύ διαφορετικά.

Οι ενσωματωμένες συναρτήσεις `dir()` και `help()` είναι χρήσιμες ως διαδραστικά βοηθήματα για εργασία με μεγάλα modules όπως `os`:

```
>>> import os
>>> dir(os)
<returns a list of all module functions>
>>> help(os)
<returns an extensive manual page created from the module's docstrings>
```

Για καθημερινές διαχειριστικές εργασίες σε αρχεία και καταλόγους, το module `shutil` παρέχει μια διεπαφή υψηλότερου επιπέδου που είναι πιο εύκολη στην χρήση:

```
>>> import shutil
>>> shutil.copyfile('data.db', 'archive.db')
'archive.db'
>>> shutil.move('/build/executables', 'installdir')
'installdir'
```

## 10.2 Wildcard Αρχεία

Το module `glob` παρέχει μια λειτουργία για τη δημιουργία λιστών αρχείων από αναζητήσεις με χαρακτήρες μπαλαντέρ καταλόγου:

```
>>> import glob
>>> glob.glob('*.py')
['primes.py', 'random.py', 'quote.py']
```

## 10.3 Ορίσματα γραμμής εντολών

Common utility scripts often need to process command line arguments. These arguments are stored in the `sys` module's `argv` attribute as a list. For instance the following output results from running `python demo.py one two three` at the command line:

```
>>> import sys
>>> print(sys.argv)
['demo.py', 'one', 'two', 'three']
```

Το module `argparse` παρέχει έναν πιο εξελιγμένο μηχανισμό για την επεξεργασία ορισμάτων γραμμής εντολών. Το ακόλουθο script εξάγει ένα ή περισσότερα ονόματα αρχείων και έναν προαιρετικό αριθμό γραμμών που θα εμφανιστούν:

```
import argparse

parser = argparse.ArgumentParser(
    prog='top',
    description='Show top lines from each file')
parser.add_argument('filenames', nargs='+')
parser.add_argument('-l', '--lines', type=int, default=10)
args = parser.parse_args()
print(args)
```

Όταν εκτελείται στη γραμμή εντολών το `python top.py --lines=5 alpha.txt beta.txt`, το script ορίζει το `args.lines` σε 5 και το `args.filenames` σε `['alpha.txt', 'beta.txt']`.

## 10.4 Ανακατεύθυνση εξόδου σφάλματος και τερματισμός προγράμματος

Το module `sys` έχει επίσης χαρακτηριστικά για `stdin`, `stdout`, και `stderr`. Το τελευταίο είναι χρήσιμο για την εκπομπή προειδοποιήσεων και μηνυμάτων σφαλμάτων ώστε να είναι ορατά ακόμα και όταν το `stdout` έχει ανακατευθυνθεί:

```
>>> sys.stderr.write('Warning, log file not found starting a new one\n')
Warning, log file not found starting a new one
```

Ο πιο άμεσος τρόπος για να τερματίσετε ένα script είναι να χρησιμοποιήσετε το `sys.exit()`.



## 10.5 Ταίριασμα μοτίβων συμβολοσειρών

Το module `re` παρέχει εργαλεία κανονική έκφρασης για προηγμένη επεξεργασία συμβολοσειρών. Για πολύπλοκη αντιστοίχιση και χειρισμό, οι τυπικές εκφράσεις προσφέρουν συνοπτικές, βελτιστοποιημένες λύσεις:

```
>>> import re
>>> re.findall(r'\b[a-z]*', 'which foot or hand fell fastest')
['foot', 'fell', 'fastest']
>>> re.sub(r'(\b[a-z]+) \1', r'\1', 'cat in the the hat')
'cat in the hat'
```

Όταν χρειάζονται μόνο απλές δυνατότητες, προτιμώνται οι μέθοδοι συμβολοσειρών, επειδή είναι ευκολότερες στην ανάγνωση και τον εντοπισμό σφαλμάτων:

```
>>> 'tea for too'.replace('too', 'two')
'tea for two'
```

## 10.6 Μαθηματικά

The `math` module gives access to the underlying C library functions for floating point math:

```
>>> import math
>>> math.cos(math.pi / 4)
0.70710678118654757
>>> math.log(1024, 2)
10.0
```

Το module `random` παρέχει εργαλεία για να κάνουμε τυχαίες επιλογές:

```
>>> import random
>>> random.choice(['apple', 'pear', 'banana'])
'apple'
>>> random.sample(range(100), 10) # sampling without replacement
[30, 83, 16, 4, 8, 81, 41, 50, 18, 33]
>>> random.random() # random float
0.17970987693706186
>>> random.randrange(6) # random integer chosen from range(6)
4
```

Το module `statistics` υπολογίζει βασικές στατιστικές ιδιότητες (μέσος όρος, διάμεσος, διακύμανση, κ.λπ.) αριθμητικών δεδομένων:

```
>>> import statistics
>>> data = [2.75, 1.75, 1.25, 0.25, 0.5, 1.25, 3.5]
>>> statistics.mean(data)
1.6071428571428572
>>> statistics.median(data)
1.25
>>> statistics.variance(data)
1.3720238095238095
```

Το έργο SciPy <<https://scipy.org>> έχει πολλές άλλες ενότητες για αριθμητικούς υπολογισμούς.

## 10.7 Πρόσβαση στο Διαδίκτυο

Υπάρχει ένας αριθμός modules για πρόσβαση στο διαδίκτυο και επεξεργασία πρωτοκόλλων διαδικτύου. Δύο από τα πιο απλά είναι το `urllib.request` για να την ανάκτηση δεδομένων από διευθύνσεις URL και το `smtplib` για την αποστολή αλληλογραφίας:

```
>>> from urllib.request import urlopen
>>> with urlopen('http://worldtimeapi.org/api/timezone/etc/UTC.txt') as response:
...     for line in response:
...         line = line.decode()           # Convert bytes to a str
...         if line.startswith('datetime'):
...             print(line.rstrip())       # Remove trailing newline
...
datetime: 2022-01-01T01:36:47.689215+00:00

>>> import smtplib
>>> server = smtplib.SMTP('localhost')
>>> server.sendmail('soothsayer@example.org', 'jcaesar@example.org',
...     """To: jcaesar@example.org
...     From: soothsayer@example.org
...
...     Beware the Ides of March.
...     """)
>>> server.quit()
```

(Σημειώστε ότι το δεύτερο παράδειγμα χρειάζεται διακομιστή αλληλογραφίας που εκτελείται σε localhost.)

## 10.8 Ημερομηνίες και ώρες

Το module `datetime` παρέχει κλάσεις για χειρισμό ημερομηνιών και ωρών με απλούς και σύνθετους τρόπους. Ενώ υποστηρίζεται η αριθμητική ημερομηνία και ώρα, η υλοποίηση εστιάζεται στην αποτελεσματική εξαγωγή μελών για μορφοποίηση και χειρισμό εξόδου. Το module επίσης υποστηρίζει αντικείμενα που έχουν επίγνωση ζώνης ώρας.

```
>>> # dates are easily constructed and formatted
>>> from datetime import date
>>> now = date.today()
>>> now
datetime.date(2003, 12, 2)
>>> now.strftime("%m-%d-%y. %d %b %Y is a %A on the %d day of %B.")
'12-02-03. 02 Dec 2003 is a Tuesday on the 02 day of December.'

>>> # dates support calendar arithmetic
>>> birthday = date(1964, 7, 31)
>>> age = now - birthday
>>> age.days
14368
```

## 10.9 Συμπίεση Δεδομένων

Οι συνήθεις μορφές αρχειοθέτησης και συμπίεσης δεδομένων υποστηρίζονται άμεσα από modules όπως: `zlib`, `gzip`, `bz2`, `lzma`, `zipfile` και `tarfile`.

```
>>> import zlib
>>> s = b'witch which has which witches wrist watch'
>>> len(s)
41
>>> t = zlib.compress(s)
>>> len(t)
37
>>> zlib.decompress(t)
b'witch which has which witches wrist watch'
>>> zlib.crc32(s)
226805979
```

## 10.10 Μέτρηση επίδοσης

Ορισμένοι χρήστες Python αναπτύσσουν βαθύ ενδιαφέρον να γνωρίζουν τη σχετική απόδοση διαφορετικών προσεγγίσεων στο ίδιο πρόβλημα. Η Python παρέχει ένα εργαλείο μέτρησης που απαντά σε αυτές τις ερωτήσεις αμέσως.

Για παράδειγμα, μπορεί να είναι δελεαστικό να χρησιμοποιήσετε τη δυνατότητα tuple packing και unpacking αντί της παραδοσιακής προσέγγισης για την εναλλαγή ορισμάτων. Το module `timeit` δείχνει γρήγορα ένα ταπεινό πλεονέκτημα απόδοσης:

```
>>> from timeit import Timer
>>> Timer('t=a; a=b; b=t', 'a=1; b=2').timeit()
0.57535828626024577
>>> Timer('a,b = b,a', 'a=1; b=2').timeit()
0.54962537085770791
```

Σε αντίθεση με το λεπτό επίπεδο ευκρίνειας του `timeit`, τα modules `profile` και `pstats` παρέχουν εργαλεία για τον εντοπισμό κρίσιμων χρονικών τμημάτων σε μεγαλύτερα μπλοκ κώδικα.

## 10.11 Έλεγχος ποιότητας

Μια καλή προσέγγιση για την ανάπτυξη λογισμικού υψηλής ποιότητας είναι να γράφονται tests για κάθε λειτουργία καθώς αναπτύσσεται και να εκτελούνται συχνά αυτά τα tests κατά τη διαδικασία ανάπτυξης.

Το module `doctest` παρέχει ένα εργαλείο για τη σάρωση ενός module και την επικύρωση tests που είναι ενσωματωμένες στις συμβολοσειρές εγγράφων ενός προγράμματος. Η κατασκευή του test είναι τόσο απλή όσο η αποκοπή και επικόλληση μιας τυπικής κλήσης μαζί με τα αποτελέσματα της στη συμβολοσειρά εγγράφων. Αυτό βελτιώνει την τεκμηρίωση παρέχοντας στον χρήστη ένα παράδειγμα και επιτρέπει στην ενότητα `doctest` να βεβαιωθεί ότι ο κώδικας παραμένει πιστός στην τεκμηρίωση:

```
def average(values):
    """Computes the arithmetic mean of a list of numbers.

    >>> print(average([20, 30, 70]))
    40.0
    """
    return sum(values) / len(values)

import doctest
doctest.testmod() # automatically validate the embedded tests
```

Το module `unittest` δεν είναι τόσο εύκολο όσο το module `doctest`, αλλά επιτρέπει τη διατήρηση ενός πιο ολοκληρωμένου συνόλου tests σε ξεχωριστό αρχείο:

```
import unittest

class TestStatisticalFunctions(unittest.TestCase):

    def test_average(self):
        self.assertEqual(average([20, 30, 70]), 40.0)
        self.assertEqual(round(average([1, 5, 7]), 1), 4.3)
        with self.assertRaises(ZeroDivisionError):
            average([])
        with self.assertRaises(TypeError):
            average(20, 30, 70)

unittest.main() # Calling from the command line invokes all tests
```

## 10.12 Batteries Included

Η Python έχει μια φιλοσοφία «συμπεριλαμβάνονται μπαταρίες». Αυτό φαίνεται καλύτερα μέσα από τις εξε-  
λιγμένες και ισχυρές δυνατότητες των μεγαλύτερων πακέτων της. Για παράδειγμα:

- Τα modules `xmlrpc.client` και `xmlrpc.server` καθιστούν την υλοποίηση κλήσεων απομακρυ-  
σμένων διαδικασιών σε μια σχεδόν ασήμαντη εργασία. Παρά τα ονόματα των modules, δεν απαιτείται  
άμεση γνώση ή χειρισμός της XML.
- Το πακέτο `email` είναι μια βιβλιοθήκη για τη διαχείριση μηνυμάτων ηλεκτρονικού ταχυδρομείου,  
συμπεριλαμβανομένων MIME και άλλων μηνυμάτων εγγράφων που βασίζονται σε [RFC 2822](#). Σε αντί-  
θεση με τα `smtplib` και `poplib` που στην πραγματικότητα στέλνουν και λαμβάνουν μηνύματα, το  
πακέτο `email` έχει ένα πλήρες σύνολο εργαλείων για τη δημιουργία ή την αποκωδικοποίηση πολύπλο-  
κων δομών μηνυμάτων (συμπεριλαμβανομένων των συνημμένων) και για την εφαρμογή κωδικοποίηση  
και πρωτόκολλο κεφαλίδων στο διαδίκτυο.
- Το πακέτο `json` παρέχει ισχυρή υποστήριξη για την ανάλυση αυτής της δημοφιλούς μορφής ανταλ-  
λαγής δεδομένων. Το module `csv` υποστηρίζει την άμεση ανάγνωση και εγγραφή αρχείων σε μορφή τι-  
μής διαχωρισμένου με κόμματα, που συνήθως υποστηρίζεται από βάσεις δεδομένων και υπολογιστικά  
φύλλα. Η XML επεξεργασία υποστηρίζεται από τα πακέτα `xml.etree.ElementTree`, `xml.dom`  
και `xml.sax`. Μαζί, αυτές οι μονάδες και τα πακέτα απλοποιούν σημαντικά την ανταλλαγή δεδομέ-  
νων μεταξύ εφαρμογών Python και άλλων εργαλείων.
- Το module `sqlite3` αποτελεί έναν wrapper για τη βιβλιοθήκη της βάσης δεδομένων SQLite, παρέχο-  
ντας μια συνεχής βάση δεδομένων που μπορεί να ενημερωθεί και να προσπελαστεί χρησιμοποιώντας  
ελαφρώς μη τυπική σύνταξη SQL.
- Η διεθνοποίηση υποστηρίζεται από έναν αριθμό modules, συμπεριλαμβανομένων των `gettext`,  
`locale`, και το πακέτο `codecs`.

---

## Σύντομη περιήγηση στην Πρότυπη Βιβλιοθήκη — Μέρος II

---

Αυτή η δεύτερη περιήγηση καλύπτει τα πιο προηγμένα modules που υποστηρίζουν επαγγελματικές ανάγκες προγραμματισμού. Αυτά τα modules σπάνια εμφανίζονται σε μικρά scripts.

### 11.1 Μορφοποίηση εξόδου

Το module `reprlib` παρέχει μια έκδοση του `repr()` προσαρμοσμένη για συντομευμένες εμφανίσεις μεγάλων ή βαθιά ένθετων containers:

```
>>> import reprlib
>>> reprlib.repr(set('supercalifragilisticexpialidocious'))
"{'a', 'c', 'd', 'e', 'f', 'g', ...}"
```

Το module `pprint` προσφέρει πιο εξελιγμένο έλεγχο της εκτύπωσης τόσο των ενσωματωμένων (built-in) και των καθορισμένων από τον χρήστη αντικειμένων με τρόπο που είναι ευανάγνωστο από τον διερμηνέα. Όταν το αποτέλεσμα είναι μεγαλύτερο από μία γραμμή, ο «pretty printer» προσθέτει αλλαγές γραμμής και εσοχές για να εμφανιστεί πιο ξεκάθαρα η δομή δεδομένων:

```
>>> import pprint
>>> t = [[['black', 'cyan'], 'white', ['green', 'red']], [['magenta',
...     'yellow'], 'blue']]
...
>>> pprint.pprint(t, width=30)
[[['black', 'cyan'],
   'white',
   ['green', 'red']],
 [['magenta', 'yellow'],
  'blue']]
```

Το module `textwrap` μορφοποιεί τις παραγράφους του κειμένου ώστε να ταιριάζει σε ένα δεδομένο πλάτος οθόνης:

```
>>> import textwrap
>>> doc = """The wrap() method is just like fill() except that it returns
... a list of strings instead of one big string with newlines to separate
... the wrapped lines."""
...

```

(συνέχεια στην επόμενη σελίδα)

(συνεχίζεται από την προηγούμενη σελίδα)

```
>>> print(textwrap.fill(doc, width=40))
The wrap() method is just like fill()
except that it returns a list of strings
instead of one big string with newlines
to separate the wrapped lines.
```

Το module `locale` έχει πρόσβαση σε μια βάση δεδομένων μορφών δεδομένων συγκεκριμένης κουλτούρας. Το χαρακτηριστικό ομαδοποίησης της συνάρτησης μορφοποίησης της τοπικής ρύθμισης παρέχει έναν άμεσο τρόπο μορφοποίησης αριθμών με διαχωριστικά ομάδων:

```
>>> import locale
>>> locale.setlocale(locale.LC_ALL, 'English_United States.1252')
'English_United States.1252'
>>> conv = locale.localeconv()           # get a mapping of conventions
>>> x = 1234567.8
>>> locale.format("%d", x, grouping=True)
'1,234,567'
>>> locale.format_string("%s%.*f", (conv['currency_symbol'],
...                               conv['frac_digits'], x), grouping=True)
'$1,234,567.80'
```

## 11.2 Templating

Το module `string` περιλαμβάνει μια ευέλικτη κλάση `Template` με απλοποιημένη σύνταξη κατάλληλη για επεξεργασία από τελικούς χρήστες. Αυτό επιτρέπει στους χρήστες να προσαρμόζουν τις εφαρμογές τους χωρίς να χρειάζεται να αλλάξουν την εφαρμογή.

Η μορφή χρησιμοποιεί ονόματα κράτησης θέσης που σχηματίζονται από `$` με έγκυρα αναγνωριστικά Python (αλφαριθμητικούς χαρακτήρες και κάτω παύλες). Περιβάλλοντας το placeholder με αγκύλες επιτρέπει να ακολουθείται από περισσότερα αλφαριθμητικά γράμματα χωρίς ενδιάμεσα κενά. Γράφοντας `$$` δημιουργεί ένα ενιαίο `$`:

```
>>> from string import Template
>>> t = Template('${village}folk send $$10 to $cause.')
>>> t.substitute(village='Nottingham', cause='the ditch fund')
'Nottinghamfolk send $10 to the ditch fund.'
```

Η μέθοδος `substitute()` κάνει `raise` ένα `KeyError` όταν ένα placeholder δεν παρέχεται σε ένα λεξικό ή ένα όρισμα λέξης-κλειδιού. Για εφαρμογές στυλ συγχώνευσης `mail`, τα δεδομένα που παρέχονται από τον χρήστη ενδέχεται να είναι ελλιπή και η μέθοδος `safe_substitute()` μπορεί να είναι πιο κατάλληλη — θα αφήσει αμετάβλητα τα placeholders εάν λείπουν δεδομένα:

```
>>> t = Template('Return the $item to $owner.')
>>> d = dict(item='unladen swallow')
>>> t.substitute(d)
Traceback (most recent call last):
...
KeyError: 'owner'
>>> t.safe_substitute(d)
'Return the unladen swallow to $owner.'
```

Οι υποκατηγορίες προτύπων μπορούν να καθορίσουν έναν προσαρμοσμένο οριοθέτη. Για παράδειγμα, ένα batch πρόγραμμα μετονομασίας για ένα πρόγραμμα περιήγησης φωτογραφιών μπορεί να επιλέξει να χρησιμοποιεί σύμβολα ποσοστού για placeholders όπως η τρέχουσα ημερομηνία, ο αριθμός ακολουθίας εικόνων ή η μορφή αρχείου:

```
>>> import time, os.path
>>> photofiles = ['img_1074.jpg', 'img_1076.jpg', 'img_1077.jpg']
>>> class BatchRename(Template):
...     delimiter = '%'
...
>>> fmt = input('Enter rename style (%d-date %n-seqnum %f-format): ')
Enter rename style (%d-date %n-seqnum %f-format): Ashley_%n%f

>>> t = BatchRename(fmt)
>>> date = time.strftime('%d%b%y')
>>> for i, filename in enumerate(photofiles):
...     base, ext = os.path.splitext(filename)
...     newname = t.substitute(d=date, n=i, f=ext)
...     print('{0} --> {1}'.format(filename, newname))

img_1074.jpg --> Ashley_0.jpg
img_1076.jpg --> Ashley_1.jpg
img_1077.jpg --> Ashley_2.jpg
```

Μια άλλη εφαρμογή για templating είναι ο διαχωρισμός της λογικής του προγράμματος από τις λεπτομέρειες πολλαπλών μορφών εξόδου. Αυτό καθιστά δυνατή την αντικατάσταση προσαρμοσμένων προτύπων για αρχεία XML, αναφορές απλού κειμένου και αναφορές ιστού HTML.

## 11.3 Εργασία με δυαδικές διατάξεις εγγραφής δεδομένων

Το module `struct` παρέχει τις συναρτήσεις `pack()` και `unpack()` για εργασία με μορφές δυαδικών (binary) εγγραφών μεταβλητού μήκους. Το ακόλουθο παράδειγμα δείχνει πως να κάνετε μια λούπα μέσω των πληροφοριών κεφαλίδας στο ένα αρχείο ZIP χωρίς τη χρήση του module `zipfile`. Οι κωδικοί πακέτου "H" και "I" αντιπροσωπεύουν αριθμούς χωρίς υπογραφή δύο και τεσσάρων byte αντίστοιχα. Το "<" υποδηλώνει ότι είναι τυπικού μεγέθους και σε σειρά byte λίγο endian:

```
import struct

with open('myfile.zip', 'rb') as f:
    data = f.read()

start = 0
for i in range(3):
    # show the first 3 file headers
    start += 14
    fields = struct.unpack('<IIIHH', data[start:start+16])
    crc32, comp_size, uncomp_size, filenamesize, extra_size = fields

    start += 16
    filename = data[start:start+filenamesize]
    start += filenamesize
    extra = data[start:start+extra_size]
    print(filename, hex(crc32), comp_size, uncomp_size)

    start += extra_size + comp_size
    # skip to the next header
```

## 11.4 Multi-threading

Το threading είναι μια τεχνική για την αποσύνδεση εργασιών που είναι διαδοχικά εξαρτώμενες. Τα νήματα μπορούν να χρησιμοποιηθούν για την βελτίωση της ανταπόκρισης των εφαρμογών που δέχονται είσοδο από τον χρήστη ενώ άλλες εργασίες εκτελούνται στο παρασκήνιο. Μια σχετική περίπτωση χρήσης εκτελεί I/O παράλληλα με υπολογισμούς στο άλλο νήμα.

Ο ακόλουθος κώδικας δείχνει πως το module υψηλού επιπέδου threading μπορεί να εκτελεί εργασίες στο παρασκήνιο ενώ το κύριο πρόγραμμα συνεχίζει να εκτελείται:

```
import threading, zipfile

class AsyncZip(threading.Thread):
    def __init__(self, infile, outfile):
        threading.Thread.__init__(self)
        self.infile = infile
        self.outfile = outfile

    def run(self):
        f = zipfile.ZipFile(self.outfile, 'w', zipfile.ZIP_DEFLATED)
        f.write(self.infile)
        f.close()
        print('Finished background zip of:', self.infile)

background = AsyncZip('mydata.txt', 'myarchive.zip')
background.start()
print('The main program continues to run in foreground.')

background.join()    # Wait for the background task to finish
print('Main program waited until background was done.')
```

Η κύρια πρόκληση των εφαρμογών πολλαπλών νημάτων είναι ο συντονισμός νημάτων που μοιράζονται δεδομένα ή άλλους πόρους. Για το σκοπό αυτό, το thread module παρέχει έναν αριθμό πρωτόγονων συγχρονισμού, συμπεριλαμβανομένων locks, events, μεταβλητών συνθηκών, και semaphores.

Ενώ αυτά τα εργαλεία είναι ισχυρά, μικρά σφάλματα σχεδιασμού μπορεί να οδηγήσουν σε προβλήματα που είναι δύσκολο να αναπαραχθούν. Επομένως, η προτιμώμενη προσέγγιση στον συντονισμό εργασιών είναι να συγκεντρωθεί όλη η πρόσβαση σε έναν πόρο σε ένα μόνο νήμα και στη συνέχεια να χρησιμοποιηθεί το module queue, για τροφοδοτήσει αυτό το νήμα με αιτήματα από άλλα νήματα. Οι εφαρμογές που χρησιμοποιούν αντικείμενα Queue για επικοινωνία και συντονισμό μεταξύ νημάτων είναι πιο εύκολο να σχεδιαστούν, είναι πιο ευανάγνωστες και πιο αξιόπιστες.

## 11.5 Logging

Το module logging προσφέρει ένα πλήρως εξοπλισμένο και ευέλικτο σύστημα καταγραφής. Στην απλούστερη μορφή του, τα μηνύματα καταγραφής αποστέλλονται σε ένα αρχείο ή στο `sys.stderr`:

```
import logging
logging.debug('Debugging information')
logging.info('Informational message')
logging.warning('Warning:config file %s not found', 'server.conf')
logging.error('Error occurred')
logging.critical('Critical error -- shutting down')
```

Αυτό παράγει την ακόλουθη έξοδο:

```
WARNING:root:Warning:config file server.conf not found
ERROR:root:Error occurred
CRITICAL:root:Critical error -- shutting down
```



Από default (προεπιλογή), τα ενημερωτικά μηνύματα και τα μηνύματα εντοπισμού σφαλμάτων αποκρύπτονται και η έξοδος αποστέλλεται σε τυπικό σφάλμα. Άλλες επιλογές εξόδου περιλαμβάνουν δρομολόγηση μηνυμάτων μέσω email, datagrams, υποδοχών (sockets), ή σε HTTP διακομιστή (server). Τα νέα φίλτρα μπορούν να επιλέξουν διαφορετική δρομολόγηση με βάση την προτεραιότητα του μηνύματος: DEBUG, INFO, WARNING, ERROR, και CRITICAL.

Το σύστημα logging μπορεί να διαμορφωθεί απευθείας από την Python ή μπορεί να φορτωθεί από ένα επεξεργάσιμο αρχείο διαμόρφωσης για προσαρμοσμένη καταγραφή χωρίς την τροποποίηση της εφαρμογής.

## 11.6 Αδύναμες αναφορές

Η Python κάνει αυτόματη διαχείριση μνήμης (καταμέτρηση αναφορών για τα περισσότερα αντικείμενα και *garbage collection* για την εξάλειψη των κύκλων). Η μνήμη ελευθερώνεται λίγο μετά την κατάργηση της τελευταίας αναφοράς σε αυτήν.

Αυτή η προσέγγιση λειτουργεί καλά για τις περισσότερες εφαρμογές, αλλά περιστασιακά υπάρχει ανάγκη παρακολούθησης αντικειμένων μόνο εφόσον χρησιμοποιούνται από κάτι άλλο. Δυστυχώς, η παρακολούθηση τους δημιουργεί μια αναφορά που τα κάνει μόνιμα. Το module `weakref` παρέχει εργαλεία για την παρακολούθηση αντικειμένων χωρίς τη δημιουργία αναφοράς. Όταν το αντικείμενο δεν χρειάζεται πλέον, αφαιρείται αυτόματα από έναν πίνακα ασθενούς αναφοράς και ενεργοποιείται μια επιστροφή κλήσης για αντικείμενα ασθενούς αναφοράς:

```
>>> import weakref, gc
>>> class A:
...     def __init__(self, value):
...         self.value = value
...     def __repr__(self):
...         return str(self.value)
...
>>> a = A(10)                                # create a reference
>>> d = weakref.WeakValueDictionary()
>>> d['primary'] = a                          # does not create a reference
>>> d['primary']                              # fetch the object if it is still alive
10
>>> del a                                    # remove the one reference
>>> gc.collect()                             # run garbage collection right away
0
>>> d['primary']                             # entry was automatically removed
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
    d['primary']                              # entry was automatically removed
  File "C:/python310/lib/weakref.py", line 46, in __getitem__
    o = self.data[key]()
KeyError: 'primary'
```

## 11.7 Εργαλεία για εργασία με λίστες

Πολλές ανάγκες δομών δεδομένων μπορούν να καλυφθούν με τον ενσωματωμένο τύπο λίστας. Ωστόσο, μερικές φορές υπάρχει ανάγκη για εναλλακτικές υλοποιήσεις με διαφορετικούς συμβιβασμούς απόδοσης.

The `array` module provides an `array()` object that is like a list that stores only homogeneous data and stores it more compactly. The following example shows an array of numbers stored as two byte unsigned binary numbers (typecode "H") rather than the usual 16 bytes per entry for regular lists of Python int objects:

```
>>> from array import array
>>> a = array('H', [4000, 10, 700, 22222])
>>> sum(a)
```

(συνέχεια στην επόμενη σελίδα)

(συνεχίζεται από την προηγούμενη σελίδα)

```
26932
>>> a[1:3]
array('H', [10, 700])
```

The `collections` module provides a `deque()` object that is like a list with faster appends and pops from the left side but slower lookups in the middle. These objects are well suited for implementing queues and breadth first tree searches:

```
>>> from collections import deque
>>> d = deque(["task1", "task2", "task3"])
>>> d.append("task4")
>>> print("Handling", d.popleft())
Handling task1
```

```
unsearched = deque([starting_node])
def breadth_first_search(unsearched):
    node = unsearched.popleft()
    for m in gen_moves(node):
        if is_goal(m):
            return m
    unsearched.append(m)
```

Εκτός από τις εναλλακτικές υλοποιήσεις λιστών, η βιβλιοθήκη προσφέρει επίσης και άλλα εργαλεία όπως το module `bisect` με συναρτήσεις για τον χειρισμό ταξινομημένων λιστών:

```
>>> import bisect
>>> scores = [(100, 'perl'), (200, 'tcl'), (400, 'lua'), (500, 'python')]
>>> bisect.insort(scores, (300, 'ruby'))
>>> scores
[(100, 'perl'), (200, 'tcl'), (300, 'ruby'), (400, 'lua'), (500, 'python')]
```

Το module `heapq` παρέχει λειτουργίες για την υλοποίηση σωρών (heaps) που βασίζονται σε κανονικές λίστες. Η καταχώριση με την χαμηλότερη τιμή διατηρείται πάντα στη θέση μηδέν. Αυτό είναι χρήσιμο για εφαρμογές που έχουν επανειλημμένα πρόσβαση στο μικρότερο στοιχείο αλλά δεν θέλουν να εκτελέσουν μια πλήρη ταξινόμηση λίστας:

```
>>> from heapq import heapify, heappop, heappush
>>> data = [1, 3, 5, 7, 9, 2, 4, 6, 8, 0]
>>> heapify(data) # rearrange the list into heap order
>>> heappush(data, -5) # add a new entry
>>> [heappop(data) for i in range(3)] # fetch the three smallest entries
[-5, 0, 1]
```

## 11.8 Decimal Floating Point Arithmetic

The `decimal` module offers a `Decimal` datatype for decimal floating point arithmetic. Compared to the built-in `float` implementation of binary floating point, the class is especially helpful for

- οικονομικές εφαρμογές και άλλες χρήσεις που απαιτούν ακριβή δεκαδική αναπαράσταση,
- έλεγχο για την ακρίβεια,
- έλεγχος της στρογγυλοποίησης για την εκπλήρωση νομικών ή κανονιστικών απαιτήσεων,
- παρακολούθηση σημαντικών δεκαδικών ψηφίων, ή
- εφαρμογές όπου ο χρήστης αναμένει ότι τα αποτελέσματα ταιριάζουν με υπολογισμούς που έγιναν με το χέρι.

Για παράδειγμα, ο υπολογισμός ενός φόρου 5% σε χρέωση τηλεφώνου 70 λεπτών δίνει διαφορετικά αποτελέσματα σε δεκαδική κινητή υποδιαστολή και σε δυαδική (binary) κινητή υποδιαστολή. Η διαφορά γίνεται σημαντική εάν τα αποτελέσματα στρογγυλοποιηθούν στο πλησιέστερο λεπτό:

```
>>> from decimal import *
>>> round(Decimal('0.70') * Decimal('1.05'), 2)
Decimal('0.74')
>>> round(.70 * 1.05, 2)
0.73
```

Το αποτέλεσμα `Decimal` διατηρεί ένα μηδέν στο τέλος, συνάγοντας αυτόματα τη σημασία τεσσάρων θέσεων από πολλαπλάσια με σημασία δύο θέσεων. Το δεκαδικό αναπαράγει τα μαθηματικά όπως γίνονται με το χέρι και αποφεύγει ζητήματα που μπορεί να προκύψουν όταν η δυαδική κινητή υποδιαστολή δεν μπορεί να αντιπροσωπεύει ακριβώς τις δεκαδικές ποσότητες.

Η ακριβής αναπαράσταση επιτρέπει στην κλάση `Decimal` να εκτελεί υπολογισμούς modulo και δοκιμές ισότητας που είναι ακατάλληλες για δυαδική κινητή υποδιαστολή:

```
>>> Decimal('1.00') % Decimal('.10')
Decimal('0.00')
>>> 1.00 % 0.10
0.099999999999999995

>>> sum([Decimal('0.1')] * 10) == Decimal('1.0')
True
>>> sum([0.1] * 10) == 1.0
False
```

Το module `decimal` παρέχει αριθμητική με όση ακρίβεια χρειάζεται:

```
>>> getcontext().prec = 36
>>> Decimal(1) / Decimal(7)
Decimal('0.142857142857142857142857142857142857')
```



---

## Εικονικά Περιβάλλοντα και πακέτα

---

### 12.1 Εισαγωγή

Οι εφαρμογές Python συχνά χρησιμοποιούν πακέτα και modules που δεν αποτελούν μέρος της τυπικής βιβλιοθήκης. Οι εφαρμογές μερικές φορές χρειάζονται μια συγκεκριμένη έκδοση μιας βιβλιοθήκης, επειδή η εφαρμογή μπορεί να απαιτεί την επιδιόρθωση ενός συγκεκριμένου σφάλματος ή επειδή η εφαρμογή έχει γραφεί χρησιμοποιώντας μια ξεπερασμένη έκδοση της διεπαφής της βιβλιοθήκης.

Αυτό σημαίνει ότι μπορεί να μην είναι δυνατό για μία εγκατάσταση Python να πληροί τις απαιτήσεις κάθε εφαρμογής. Εάν η εφαρμογή A χρειάζεται την έκδοση 1.0 ενός συγκεκριμένου module, αλλά η εφαρμογή B χρειάζεται την έκδοση 2.0, τότε οι απαιτήσεις βρίσκονται σε σύγκρουση και η εγκατάσταση της έκδοσης 1.0 ή 2.0 θα αφήσει μια εφαρμογή ανίκανη να εκτελεστεί.

Η λύση για αυτό το πρόβλημα είναι να δημιουργήσετε ένα *virtual environment*, ένα αυτόνομο δέντρο καταλόγου που περιέχει μια εγκατάσταση Python για μια συγκεκριμένη έκδοση της Python, καθώς και έναν αριθμό πρόσθετων πακέτων.

Διαφορετικές εφαρμογές μπορούν στη συνέχεια να χρησιμοποιούν διαφορετικά εικονικά περιβάλλοντα. Για να επιλύσετε το προηγούμενο παράδειγμα αντικρουόμενων απαιτήσεων, η εφαρμογή A μπορεί να έχει το δικό της εικονικό περιβάλλον με εγκατεστημένη την έκδοση 1.0 ενώ η εφαρμογή B έχει άλλο εικονικό περιβάλλον με την έκδοση 2.0. Εάν η εφαρμογή B απαιτεί αναβάθμιση βιβλιοθήκης στην έκδοση 3.0, αυτό δεν θα επηρεάσει το περιβάλλον της εφαρμογής A.

### 12.2 Δημιουργία εικονικών περιβάλλοντων

The module used to create and manage virtual environments is called `venv`. `venv` will usually install the most recent version of Python that you have available. If you have multiple versions of Python on your system, you can select a specific Python version by running `python3` or whichever version you want.

Για να δημιουργήσετε ένα εικονικό περιβάλλον, αποφασίστε έναν φάκελο όπου θέλετε να το τοποθετήσετε και εκτελέστε το module `venv` ως ένα script με τη διαδρομή καταλόγου:

```
python -m venv tutorial-env
```

Αυτό θα δημιουργήσει τον κατάλογο `tutorial-env` εάν δεν υπάρχει, και επίσης θα δημιουργήσει καταλόγους μέσα σε αυτόν που περιέχουν ένα αντίγραφο του interpreter της Python και διάφορα υποστηρικτικά αρχεία.

Μια κοινή τοποθεσία καταλόγου για ένα εικονικό περιβάλλον είναι `.venv`. Αυτό το όνομα κρατά τον κατάλογο συνήθως κρυμμένο στο shell σας και συνεπώς μακριά από τη διαδρομή, ενώ του δίνει ένα όνομα που εξηγεί γιατί υπάρχει ο κατάλογος. Αποτρέπει επίσης τη σύγκρουση με αρχεία ορισμού μεταβλητών περιβάλλοντος `.env` που υποστηρίζουν ορισμένα εργαλεία.

Μόλις δημιουργήσετε ένα εικονικό περιβάλλον, μπορεί να το ενεργοποιήσετε.

Σε Windows, εκτελέστε:

```
tutorial-env\Scripts\activate.bat
```

Σε Unix ή MacOS, εκτελέστε:

```
source tutorial-env/bin/activate
```

(Αυτό το script είναι γραμμένο για το bash shell. Εάν χρησιμοποιείτε τα shells **csh** ή **fish**, υπάρχουν εναλλακτικά scripts που θα πρέπει να χρησιμοποιούνται αντί αυτών, όπως `activate.csh` και `activate.fish`.)

Η ενεργοποίηση του εικονικού περιβάλλοντος θα αλλάξει το prompt του shell σας για να δείξει ποιο εικονικό περιβάλλον χρησιμοποιείτε και θα τροποποιήσει το περιβάλλον έτσι ώστε η εκτέλεση της python να σας δώσει τη συγκεκριμένη έκδοση και εγκατάσταση της Python. Για παράδειγμα:

```
$ source ~/envs/tutorial-env/bin/activate
(tutorial-env) $ python
Python 3.5.1 (default, May 6 2016, 10:59:36)
...
>>> import sys
>>> sys.path
['', '/usr/local/lib/python3.5.zip', ...,
'~/envs/tutorial-env/lib/python3.5/site-packages']
>>>
```

## 12.3 Διαχείριση Πακέτων με το pip

Μπορείτε να εγκαταστήσετε, να αναβαθμίσετε και να αφαιρέσετε πακέτα χρησιμοποιώντας ένα πρόγραμμα που ονομάζεται **pip**. Από προεπιλογή το pip θα εγκαταστήσει πακέτα από το [Python Package Index](#). Εσείς μπορείτε να περιηγηθείτε στο Python Package Index μεταβαίνοντας σε αυτό στο πρόγραμμα περιήγησής σας.

Το pip έχει έναν αριθμό υποεντολών: «install», «uninstall», «freeze», κ.λπ. (Συμβουλευτείτε τον οδηγό `installing-index` για πλήρη τεκμηρίωση για το pip.)

Μπορείτε να εγκαταστήσετε την τελευταία έκδοση ενός πακέτου προσδιορίζοντας ένα όνομα ενός πακέτου:

```
(tutorial-env) $ python -m pip install novas
Collecting novas
  Downloading novas-3.1.1.3.tar.gz (136kB)
Installing collected packages: novas
  Running setup.py install for novas
Successfully installed novas-3.1.1.3
```

Μπορείτε επίσης να εγκαταστήσετε μια συγκεκριμένη έκδοση ενός πακέτου δίνοντας το όνομα του πακέτου ακολουθούμενο από `==` και τον αριθμό έκδοσης:

```
(tutorial-env) $ python -m pip install requests==2.6.0
Collecting requests==2.6.0
  Using cached requests-2.6.0-py2.py3-none-any.whl
Installing collected packages: requests
Successfully installed requests-2.6.0
```

If you re-run this command, `pip` will notice that the requested version is already installed and do nothing. You can supply a different version number to get that version, or you can run `pip install --upgrade` to upgrade the package to the latest version:

```
(tutorial-env) $ python -m pip install --upgrade requests
Collecting requests
Installing collected packages: requests
  Found existing installation: requests 2.6.0
    Uninstalling requests-2.6.0:
      Successfully uninstalled requests-2.6.0
Successfully installed requests-2.7.0
```

`pip uninstall` followed by one or more package names will remove the packages from the virtual environment.

`pip show` will display information about a particular package:

```
(tutorial-env) $ pip show requests
---
Metadata-Version: 2.0
Name: requests
Version: 2.7.0
Summary: Python HTTP for Humans.
Home-page: http://python-requests.org
Author: Kenneth Reitz
Author-email: me@kennethreitz.com
License: Apache 2.0
Location: /Users/akuchling/envs/tutorial-env/lib/python3.4/site-packages
Requires:
```

`pip list` will display all of the packages installed in the virtual environment:

```
(tutorial-env) $ pip list
novas (3.1.1.3)
numpy (1.9.2)
pip (7.0.3)
requests (2.7.0)
setuptools (16.0)
```

`pip freeze` will produce a similar list of the installed packages, but the output uses the format that `pip install` expects. A common convention is to put this list in a `requirements.txt` file:

```
(tutorial-env) $ pip freeze > requirements.txt
(tutorial-env) $ cat requirements.txt
novas==3.1.1.3
numpy==1.9.2
requests==2.7.0
```

To `requirements.txt` μπορεί στη συνέχεια να δημοσιευθεί στον έλεγχο έκδοσης και να διατεθεί ως μέρος μιας εφαρμογής. Οι χρήστες μπορούν στη συνέχεια να εγκαταστήσουν όλα τα απαραίτητα πακέτα με το `install -r`:

```
(tutorial-env) $ python -m pip install -r requirements.txt
Collecting novas==3.1.1.3 (from -r requirements.txt (line 1))
...
Collecting numpy==1.9.2 (from -r requirements.txt (line 2))
...
Collecting requests==2.7.0 (from -r requirements.txt (line 3))
...
Installing collected packages: novas, numpy, requests
  Running setup.py install for novas
Successfully installed novas-3.1.1.3 numpy-1.9.2 requests-2.7.0
```

`pip` has many more options. Consult the `installing-index` guide for complete documentation for `pip`. When you've written a package and want to make it available on the Python Package Index, consult the `distributing-index` guide.



---

### Και τώρα τι;

---

Η ανάγνωση αυτού του διδακτικού υλικού πιθανότατα ενίσχυσε το ενδιαφέρον σας για τη χρήση της Python — θα πρέπει να είστε πρόθυμοι να εφαρμόσετε την Python για την επίλυση των προβλημάτων του πραγματικού σας κόσμου. Που πρέπει να πάτε για να μάθετε περισσότερα;

Αυτό το διδακτικό υλικό είναι μέρος του συνόλου τεκμηρίωσης της Python. Μερικά άλλα έγγραφα του συνόλου είναι:

- `library-index`:

You should browse through this manual, which gives complete (though terse) reference material about types, functions, and the modules in the standard library. The standard Python distribution includes a *lot* of additional code. There are modules to read Unix mailboxes, retrieve documents via HTTP, generate random numbers, parse command-line options, write CGI programs, compress data, and many other tasks. Skimming through the Library Reference will give you an idea of what's available.

- `installing-index` εξηγεί πως να εγκαταστήσετε πρόσθετα modules γραμμένα από άλλους χρήστες Python.
- `reference-index`: Μια λεπτομερής εξήγηση της σύνταξης και της σημασιολογίας. Είναι βαριά ανάγνωση, αλλά είναι χρήσιμο ως πλήρης οδηγός για την ίδια τη γλώσσα.

Περισσότερα Βοηθήματα για Python:

- <https://www.python.org>: Ο κύριος ιστότοπος της Python. Περιέχει κώδικα, τεκμηρίωση, και δείκτες που σχετίζονται με την Python στο web.
- <https://docs.python.org>: Γρήγορη πρόσβαση στην τεκμηρίωση της Python.
- <https://pypi.org>: Το ευρετήριο Πακέτων Python, που προηγουμένως ονομαζόταν επίσης Τυροκομείο<sup>1</sup>, είναι ένα ευρετήριο Python modules που έχουν δημιουργηθεί από χρήστες που είναι διαθέσιμα για λήψη. Μόλις ξεκινήσετε την κυκλοφορία του κώδικα, μπορείτε να τον καταχωρήσετε εδώ για το βρουν άλλοι.
- <https://code.activestate.com/recipes/langs/python/>: Το Python Cookbook είναι μια αρκετά μεγάλη συλλογή παραδειγμάτων κώδικα, μεγαλύτερα modules και χρήσιμα scripts. Ιδιαίτερα αξιοσημείωτες συνεισφορές συλλέγονται σε ένα βιβλίο με τίτλο Python Cookbook (O'Reilly & Associates, ISBN 0-596-00797-3.)
- <https://pyvideo.org> συλλέγει συνδέσμους προς βίντεο που σχετίζονται με την Python από συνέδρια και συναντήσεις ομάδων χρηστών.

---

<sup>1</sup> Το «Cheese Shop» είναι ένα σκίτσο των Monty Python: ένα πελάτης μπαίνει σε ένα τυροκομείο αλλά ό,τι τυρί ζητήσει, ο υπάλληλος λέει ότι δεν υπάρχει.

- <https://scipy.org>: Το έργο Scientific Python περιλαμβάνει modules για γρήγορους υπολογισμούς και χειρισμούς πινάκων συν μια σειρά από πακέτα για πράγματα όπως η γραμμική άλγεβρα, οι μετασχηματισμοί Fourier, οι μη γραμμικοί λύτες, οι κατανομές τυχαίων αριθμών, η στατιστική ανάλυση και τα λοιπά παρόμοια.

Για ερωτήσεις και αναφορές προβλημάτων που σχετίζονται με Python, μπορείτε να δημοσιεύσετε στην ομάδα συζήτησης `comp.lang.python`, ή να τις στείλετε στη λίστα αλληλογραφίας στην διεύθυνση [python-list@python.org](mailto:python-list@python.org). Η ομάδα συζήτησης και η λίστα αλληλογραφίας είναι gatewayed, έτσι ώστε τα μηνύματα που δημοσιεύονται στο ένα θα προωθούνται αυτόματα στο άλλο. Υπάρχουν εκατοντάδες αναρτήσεις την ημέρα, που κάνουν (και απαντούν) ερωτήσεις, προτείνουν νέες δυνατότητες και ανακοινώνουν νέα modules. Τα αρχεία της λίστα αλληλογραφίας είναι διαθέσιμα στη διεύθυνση <https://mail.python.org/pipermail/>.

Πριν από τη δημοσίευση, φροντίστε να ελέγξετε τη λίστα με Frequently Asked Questions (σε συντομογραφία FAQ). Οι FAQ ερωτήσεις απαντούν σε πολλές από τις ερωτήσεις που εμφανίζονται ξανά και ξανά και μπορεί να περιέχουν ήδη τη λύση για το πρόβλημά σας.

### Υποσημειώσεις

---

## Διαδραστική Επεξεργασία Input και Αντικατάσταση Ιστορικού

---

Ορισμένες εκδόσεις του interpreter της Python υποστηρίζουν την επεξεργασία του τρέχοντος input και την αντικατάσταση του ιστορικού, παρόμοια με τις λειτουργίες που βρίσκονται στο κέλυφος Korn και στο κέλυφος GNU Bash. Αυτό υλοποιείται χρησιμοποιώντας τη βιβλιοθήκη [GNU Readline](#), η οποία υποστηρίζει διάφορα στυλ επεξεργασίας. Αυτή η βιβλιοθήκη έχει τη δική της τεκμηρίωση που δεν θα αντιγράψουμε εδώ.

### 14.1 Συμπλήρωση Tab και Επεξεργασία Ιστορικού

Completion of variable and module names is automatically enabled at interpreter startup so that the Tab key invokes the completion function; it looks at Python statement names, the current local variables, and the available module names. For dotted expressions such as `string.a`, it will evaluate the expression up to the final `'.'` and then suggest completions from the attributes of the resulting object. Note that this may execute application-defined code if an object with a `__getattr__()` method is part of the expression. The default configuration also saves your history into a file named `.python_history` in your user directory. The history will be available again during the next interactive interpreter session.

### 14.2 Εναλλακτικές λύσεις για τον Διαδραστικό Interpreter

This facility is an enormous step forward compared to earlier versions of the interpreter; however, some wishes are left: It would be nice if the proper indentation were suggested on continuation lines (the parser knows if an indent token is required next). The completion mechanism might use the interpreter's symbol table. A command to check (or even suggest) matching parentheses, quotes, etc., would also be useful.

Ένας εναλλακτικός, βελτιωμένος, διαδραστικός interpreter που υπάρχει εδώ και αρκετό καιρό είναι το [IPython](#), το οποίο διαθέτει συμπλήρωση tab, εξερεύνηση αντικειμένων και προηγμένη διαχείριση ιστορικού. Μπορεί επίσης να προσαρμοστεί πλήρως και να ενσωματωθεί σε άλλες εφαρμογές. Ένα άλλο παρόμοιο βελτιωμένο διαδραστικό περιβάλλον είναι το [bpython](#).



## Floating Point Arithmetic: Issues and Limitations

Floating-point numbers are represented in computer hardware as base 2 (binary) fractions. For example, the decimal fraction

0.125

has value  $1/10 + 2/100 + 5/1000$ , and in the same way the binary fraction

0.001

has value  $0/2 + 0/4 + 1/8$ . These two fractions have identical values, the only real difference being that the first is written in base 10 fractional notation, and the second in base 2.

Δυστυχώς, τα περισσότερα δεκαδικά κλάσματα δεν μπορούν να αναπαρασταθούν ακριβώς ως κλάσματα. Η συνέπεια είναι ότι, γενικά, οι δεκαδικοί αριθμοί κινητής υποδιαστολής που εισάγετε προσεγγίζονται μόνο από τους δυαδικούς αριθμούς κινητής υποδιαστολής που είναι πράγματι αποθηκευμένοι στο μηχάνημα.

Το πρόβλημα είναι πιο κατανοητό στην αρχή με βάση το 10. Θεωρήστε το κλάσμα  $1/3$ . Μπορεί να το προσεγγίσετε ως κλάσμα βάσης 10:

0.3

ή, καλύτερα,

0.33

ή, καλύτερα,

0.333

και ούτω καθεξής. Όσα ψηφία και αν είστε διατεθειμένοι να γράψετε, το αποτέλεσμα δεν θα είναι ποτέ ακριβώς το  $1/3$ , αλλά θα είναι μια ολοένα και καλύτερη προσέγγιση του  $1/3$ .

Με τον ίδιο τρόπο, ανεξάρτητα από το πόσα ψηφία βάσης 2 είστε διατεθειμένοι να χρησιμοποιήσετε, η δεκαδική τιμή 0,1 δεν μπορεί να αναπαρασταθεί ακριβώς ως κλάσμα βάσης 2. Στη βάση 2, το  $1/10$  είναι το κλάσμα που επαναλαμβάνεται

0.0001100110011001100110011001100110011001100110011001100110011...

Σταματήστε σε οποιονδήποτε πεπερασμένο αριθμό bit και λαμβάνετε μια προσέγγιση. Στις περισσότερες μηχανές σήμερα, οι floats προσεγγίζονται χρησιμοποιώντας τα πρώτα 53 bit ξεκινώντας από το πιο σημαντικό bit και με τον παρανομαστή ως δύναμη του δύο. Στην περίπτωση του  $1/10$ , το δυαδικό κλάσμα είναι  $3602879701896397 / 2^{55}$  που είναι κοντά αλλά όχι ακριβώς ίσο με την πραγματική τιμή του  $1/10$ .

Many users are not aware of the approximation because of the way values are displayed. Python only prints a decimal approximation to the true decimal value of the binary approximation stored by the machine. On most machines, if Python were to print the true decimal value of the binary approximation stored for 0.1, it would have to display

```
>>> 0.1
0.1000000000000000055511151231257827021181583404541015625
```

That is more digits than most people find useful, so Python keeps the number of digits manageable by displaying a rounded value instead

```
>>> 1 / 10
0.1
```

Απλώς θυμηθείτε, παρόλο που το εκτυπωμένο αποτέλεσμα μοιάζει με την ακριβή τιμή του  $1/10$ , η πραγματική αποθηκευμένη τιμή είναι το πλησιέστερο αναπαράστασιμο δυαδικό κλάσμα.

Είναι ενδιαφέρον ότι υπάρχουν πολλοί διαφορετικοί δεκαδικοί αριθμοί που μοιράζονται το ίδιο πλησιέστερο κατά προσέγγιση δυαδικό κλάσμα. Για παράδειγμα, οι αριθμοί 0.1 και 0.1000000000000000055511151231257827021181583404541015625 είναι όλα κατά προσέγγιση με  $3602879701896397 / 2^{55}$ . Δεδομένου ότι όλες αυτές οι δεκαδικές τιμές μοιράζονται την ίδια προσέγγιση, οποιαδήποτε από αυτές θα μπορούσε να εμφανιστεί διατηρώντας παράλληλα το αμετάβλητο `eval(repr(x)) == x`.

Ιστορικά, το prompt της Python και η ενσωματωμένη συνάρτηση `repr()` θα επέλεγε αυτό με 17 σημαντικά ψηφία, 0.100000000000000001. Ξεκινώντας με την Python 3.1, η Python (στα περισσότερα συστήματα) είναι πλέον σε θέση να επιλέξει το συντομότερο από αυτά και απλά εμφανίζει το 0.1.

Note that this is in the very nature of binary floating-point: this is not a bug in Python, and it is not a bug in your code either. You'll see the same kind of thing in all languages that support your hardware's floating-point arithmetic (although some languages may not *display* the difference by default, or in all output modes).

For more pleasant output, you may wish to use string formatting to produce a limited number of significant digits:

```
>>> format(math.pi, '.12g') # give 12 significant digits
'3.14159265359'

>>> format(math.pi, '.2f')   # give 2 digits after the point
'3.14'

>>> repr(math.pi)
'3.141592653589793'
```

Είναι σημαντικό να συνειδητοποιήσουμε ότι αυτό είναι, με την πραγματική έννοια, μια ψευδαίσθηση: απλά στρογγυλεύετε την *παρουσίαση* της πραγματικής αξίας του μηχανήματος.

One illusion may beget another. For example, since 0.1 is not exactly  $1/10$ , summing three values of 0.1 may not yield exactly 0.3, either:

```
>>> .1 + .1 + .1 == .3
False
```

Also, since the 0.1 cannot get any closer to the exact value of  $1/10$  and 0.3 cannot get any closer to the exact value of  $3/10$ , then pre-rounding with `round()` function cannot help:

```
>>> round(.1, 1) + round(.1, 1) + round(.1, 1) == round(.3, 1)
False
```

Though the numbers cannot be made closer to their intended exact values, the `round()` function can be useful for post-rounding so that results with inexact values become comparable to one another:

```
>>> round(.1 + .1 + .1, 10) == round(.3, 10)
True
```

Binary floating-point arithmetic holds many surprises like this. The problem with «0.1» is explained in precise detail below, in the «Representation Error» section. See [The Perils of Floating Point](#) for a more complete account of other common surprises.

As that says near the end, «there are no easy answers.» Still, don't be unduly wary of floating-point! The errors in Python float operations are inherited from the floating-point hardware, and on most machines are on the order of no more than 1 part in  $2^{53}$  per operation. That's more than adequate for most tasks, but you do need to keep in mind that it's not decimal arithmetic and that every float operation can suffer a new rounding error.

Ενώ υπάρχουν παθολογικές περιπτώσεις, για την πιο περιστασιακή χρήση της αριθμητικής κινητής υποδιαστολής, θα δείτε στο τέλος το αποτέλεσμα που περιμένετε εάν απλώς στρογγυλοποιήσετε την εμφάνιση των τελικών αποτελεσμάτων σας στον αριθμό των δεκαδικών ψηφίων που περιμένετε. Το `str()` συνήθως αρκεί, και για καλύτερο έλεγχο δείτε τους προσδιοριστές μορφής της μεθόδου `str.format()` σε `formatstrings`.

Για περιπτώσεις χρήσης που απαιτούν ακριβή δεκαδική αναπαράσταση, δοκιμάστε να χρησιμοποιήσετε το `module decimal` που εφαρμόζει δεκαδική αριθμητική κατάλληλη για λογιστικές εφαρμογές και εφαρμογές υψηλής ακριβείας.

Μια άλλη μορφή ακριβούς αριθμητικής υποστηρίζεται από το `module fractions`, η οποία υλοποιεί την αριθμητική με βάση τους ορθολογικούς αριθμούς (έτσι οι αριθμοί όπως το  $1/3$  μπορούν να αναπαρασταθούν ακριβώς).

If you are a heavy user of floating point operations you should take a look at the NumPy package and many other packages for mathematical and statistical operations supplied by the SciPy project. See <https://scipy.org>.

Python provides tools that may help on those rare occasions when you really *do* want to know the exact value of a float. The `float.as_integer_ratio()` method expresses the value of a float as a fraction:

```
>>> x = 3.14159
>>> x.as_integer_ratio()
(3537115888337719, 1125899906842624)
```

Since the ratio is exact, it can be used to losslessly recreate the original value:

```
>>> x == 3537115888337719 / 1125899906842624
True
```

The `float.hex()` method expresses a float in hexadecimal (base 16), again giving the exact value stored by your computer:

```
>>> x.hex()
'0x1.921f9f01b866ep+1'
```

This precise hexadecimal representation can be used to reconstruct the float value exactly:

```
>>> x == float.fromhex('0x1.921f9f01b866ep+1')
True
```

Δεδομένου ότι η αναπαράσταση είναι ακριβής, είναι χρήσιμη για την αξιόπιστη μεταφορά τιμών σε διαφορετικές εκδόσεις της Python (ανεξαρτησία πλατφόρμας) και την ανταλλαγή δεδομένων με άλλες γλώσσες που υποστηρίζουν την ίδια μορφή (όπως Java και C99).

Another helpful tool is the `math.fsum()` function which helps mitigate loss-of-precision during summation. It tracks «lost digits» as values are added onto a running total. That can make a difference in overall accuracy so that the errors do not accumulate to the point where they affect the final total:

```
>>> sum([0.1] * 10) == 1.0
False
>>> math.fsum([0.1] * 10) == 1.0
True
```

## 15.1 Σφάλμα Αναπαράστασης

Αυτή η ενότητα εξηγεί το παράδειγμα «0.1» λεπτομερώς και δείχνει πώς μπορείτε να εκτελέσετε μια ακριβή ανάλυση περιπτώσεων όπως αυτή μόνοι σας. Υποτίθεται ότι έχετε βασική εξοικείωση με την αναπαράσταση δυαδικής κινητής υποδιαστολής.

Το *Σφάλμα αναπαράστασης (Representation error)* αναφέρεται στο γεγονός ότι ορισμένα (τα περισσότερα, στην πραγματικότητα) δεκαδικά κλάσματα δεν μπορούν να αναπαρασταθούν ακριβώς ως δυαδικά (βάση 2) κλάσματα. Αυτός είναι ο κύριος λόγος για τον οποίο η Python (ή Perl, C, C++, Java, Fortran, και πολλές άλλες) συχνά δεν εμφανίζουν τον ακριβή δεκαδικό αριθμό που περιμένετε.

Why is that? 1/10 is not exactly representable as a binary fraction. Almost all machines today (November 2000) use IEEE-754 floating point arithmetic, and almost all platforms map Python floats to IEEE-754 «double precision». 754 doubles contain 53 bits of precision, so on input the computer strives to convert 0.1 to the closest fraction it can of the form  $J/2^N$  where  $J$  is an integer containing exactly 53 bits. Rewriting

```
1 / 10 ~ J / (2**N)
```

ως

```
J ~ 2**N / 10
```

and recalling that  $J$  has exactly 53 bits (is  $\geq 2^{52}$  but  $< 2^{53}$ ), the best value for  $N$  is 56:

```
>>> 2**52 <= 2**56 // 10 < 2**53
True
```

That is, 56 is the only value for  $N$  that leaves  $J$  with exactly 53 bits. The best possible value for  $J$  is then that quotient rounded:

```
>>> q, r = divmod(2**56, 10)
>>> r
6
```

Since the remainder is more than half of 10, the best approximation is obtained by rounding up:

```
>>> q+1
7205759403792794
```

Therefore the best possible approximation to 1/10 in 754 double precision is:

```
7205759403792794 / 2 ** 56
```

Η διαίρεση του αριθμητή και του παρονομαστή με δύο μειώνει το κλάσμα σε:

```
3602879701896397 / 2 ** 55
```

Λάβετε υπόψη ότι από τη στιγμή που κάναμε στρογγυλοποίηση, αυτό είναι στην πραγματικότητα λίγο μεγαλύτερο από το 1/10· αν δεν είχαμε στρογγυλοποιήσει προς τα πάνω, το πηλίκο θα ήταν λίγο μικρότερο από το 1/10. Αλλά σε καμία περίπτωση δεν μπορεί να είναι *ακριβώς* 1/10!

So the computer never «sees» 1/10: what it sees is the exact fraction given above, the best 754 double approximation it can get:

```
>>> 0.1 * 2 ** 55
3602879701896397.0
```

If we multiply that fraction by  $10^{55}$ , we can see the value out to 55 decimal digits:

```
>>> 3602879701896397 * 10 ** 55 // 2 ** 55
1000000000000000055511151231257827021181583404541015625
```



meaning that the exact number stored in the computer is equal to the decimal value 0.1000000000000000055511151231257827021181583404541015625. Instead of displaying the full decimal value, many languages (including older versions of Python), round the result to 17 significant digits:

```
>>> format(0.1, '.17f')
'0.10000000000000001'
```

The fractions and decimal modules make these calculations easy:

```
>>> from decimal import Decimal
>>> from fractions import Fraction

>>> Fraction.from_float(0.1)
Fraction(3602879701896397, 36028797018963968)

>>> (0.1).as_integer_ratio()
(3602879701896397, 36028797018963968)

>>> Decimal.from_float(0.1)
Decimal('0.1000000000000000055511151231257827021181583404541015625')

>>> format(Decimal.from_float(0.1), '.17f')
'0.10000000000000001'
```



## 16.1 Διαδραστική Λειτουργία

### 16.1.1 Διαχείριση Σφαλμάτων

When an error occurs, the interpreter prints an error message and a stack trace. In interactive mode, it then returns to the primary prompt; when input came from a file, it exits with a nonzero exit status after printing the stack trace. (Exceptions handled by an `except` clause in a `try` statement are not errors in this context.) Some errors are unconditionally fatal and cause an exit with a nonzero exit; this applies to internal inconsistencies and some cases of running out of memory. All error messages are written to the standard error stream; normal output from executed commands is written to standard output.

Πληκτρολογώντας τον χαρακτήρα διακοπής (συνήθως `Control-C` ή `Delete`) στην κύρια ή δευτερεύουσα γραμμή εντολών ακυρώνει την είσοδο και επιστρέφει στην κύρια γραμμή εντολών.<sup>1</sup> Πληκτρολογώντας μια διακοπή ενώ εκτελείται μια εντολή δημιουργείται η εξαίρεση `KeyboardInterrupt`, η οποία μπορεί να αντιμετωπιστεί από μια πρόταση `try`.

### 16.1.2 Εκτελέσιμα Python Scripts

Στα συστήματα BSD-ish Unix systems, τα Python scripts μπορούν να γίνουν άμεσα εκτελέσιμα, όπως τα shell scripts, βάζοντας τη γραμμή:

```
#!/usr/bin/env python3.5
```

(υποθέτοντας ότι ο interpreter βρίσκεται στο `PATH` του χρήστη) στην αρχή του σεναρίου και δίνοντας στο αρχείο μια εκτελέσιμη λειτουργία. Το `#!` πρέπει να είναι οι δύο πρώτοι χαρακτήρες του αρχείου. Σε ορισμένες πλατφόρμες, αυτή η πρώτη γραμμή πρέπει να τελειώνει με μια γραμμή τύπου Unix που τελειώνει με μια γραμμή τύπου Unix που τελειώνει (`'\n'`), όχι με γραμμή Windows (`'\r\n'`). Σημειώστε ότι ο χαρακτήρας κατακερματισμού, `,` `'#'`, χρησιμοποιείται για την έναρξη ενός σχολίου στην Python.

Το script μπορεί να δοθεί μια εκτελέσιμη λειτουργία ή άδεια, χρησιμοποιώντας την εντολή `chmod`.

```
$ chmod +x myscript.py
```

<sup>1</sup> Ένα πρόβλημα με το πακέτο GNU Readline μπορεί να το αποτρέψει.

Στα συστήματα Windows, δεν υπάρχει η έννοια της «εκτελέσιμη λειτουργίας». Το πρόγραμμα εγκατάστασης της Python συσχετίζει αυτόματα τα αρχεία `.py` με το `python.exe` έτσι ώστε να εκτελείται ένα διπλό κλικ σε ένα αρχείο Python ως script. Η επέκταση μπορεί επίσης να είναι `.pyw`, σε αυτήν την περίπτωση, το παράθυρο της κονσόλας που εμφανίζεται συνήθως αποκρύπτεται.

### 16.1.3 Το διαδραστικό αρχείο εκκίνησης

Όταν χρησιμοποιείτε την Python διαδραστικά, είναι συχνά βολικό να εκτελούνται ορισμένες τυπικές εντολές κάθε φορά που ξεκινά ο interpreter. Μπορείτε να το κάνετε αυτό ορίζοντας μια μεταβλητή περιβάλλοντος με το όνομα `PYTHONSTARTUP` στο όνομα ενός αρχείου που περιέχει τις εντολές εκκίνησης σας. Αυτό είναι παρόμοιο με το χαρακτηριστικό `.profile` στα Unix shells.

Αυτό το αρχείο διαβάζεται μόνο σε διαδραστικές συνεδρίες, όχι όταν η Python διαβάζει εντολές από ένα script, και όχι όταν το `/dev/tty` δίνεται ως η ρητή πηγή εντολών (η οποία κατά τα άλλα συμπεριφέρεται σαν μια διαδραστική συνεδρία). Εκτελείται στον ίδιο χώρο ονομάτων όπου εκτελούνται αλληλεπιδραστικές εντολές, έτσι ώστε τα αντικείμενα που ορίζει ή εισάγει να μπορούν να χρησιμοποιηθούν χωρίς επιφύλαξη στη διαδραστική περίοδο λειτουργίας. Μπορείτε επίσης να αλλάξετε τις προτροπές `sys.ps1` και `sys.ps2` σε αυτό το αρχείο.

Εάν θέλετε να διαβάσετε ένα επιπλέον αρχείο εκκίνησης από τον τρέχοντα κατάλογο, μπορείτε να το προγραμματίσετε στο καθολικό αρχείο εκκίνησης χρησιμοποιώντας κώδικα όπως `if os.path.isfile('.pythonrc.py') : exec(open('.pythonrc.py').read())`. Εάν θέλετε να χρησιμοποιήσετε το αρχείο εκκίνησης σε ένα σενάριο, πρέπει να το κάνετε ρητά στο script:

```
import os
filename = os.environ.get('PYTHONSTARTUP')
if filename and os.path.isfile(filename):
    with open(filename) as fobj:
        startup_file = fobj.read()
    exec(startup_file)
```

### 16.1.4 Τα Modules Προσαρμογής

Python provides two hooks to let you customize it: `sitecustomize` and `usercustomize`. To see how it works, you need first to find the location of your user site-packages directory. Start Python and run this code:

```
>>> import site
>>> site.getusersitepackages()
'/home/user/.local/lib/python3.5/site-packages'
```

Τώρα μπορείτε να δημιουργήσετε ένα αρχείο με το όνομα `usercustomize.py` σε αυτόν τον κατάλογο και να βάλετε ό,τι θέλετε σε αυτόν. Θα επηρεάσει κάθε επίκληση της Python, εκτός εάν ξεκινήσει με την επιλογή `-s` απενεργοποιήστε την αυτόματη εισαγωγή.

`sitecustomize` works in the same way, but is typically created by an administrator of the computer in the global site-packages directory, and is imported before `usercustomize`. See the documentation of the `site` module for more details.

## Υποσημειώσεις



>>> Το προεπιλεγμένο Python prompt του διαδραστικού shell. Συχνά εμφανίζεται για παραδείγματα κώδικα που μπορούν να εκτελεστούν διαδραστικά στον interpreter.

... Μπορεί να αναφέρεται σε:

- Το προεπιλεγμένο Python prompt του διαδραστικού shell κατά την εισαγωγή του κώδικα για ένα μπλοκ κώδικα με εσοχή, όταν βρίσκεται μέσα σε ένα ζεύγος ταιριασμένων αριστερών και δεξιών delimiters (παρενθέσεις, αγκύλες, άγκιστρα ή τριπλά εισαγωγικά), ή μετά τον καθορισμό ενός decorator.
- Η ενσωματωμένη σταθερά Ellipsis.

**2to3** Ένα εργαλείο που προσπαθεί να μετατρέψει τον κώδικα Python 2.x σε κώδικα Python 3.x διαχειρίζοντας τις περισσότερες ασυμβατότητες που μπορούν να εντοπιστούν αναλύοντας την πηγή και διασχίζοντας το δέντρο ανάλυσης.

2to3 είναι διαθέσιμο στην στάνταρ βιβλιοθήκη ως `lib2to3`, παρέχεται ένα σημείο εισόδου ως `Tools/scripts/2to3`. Βλ. [2to3-reference](#).

**αφηρημένη βασική κλάση** Οι αφηρημένες βασικές κλάσεις συμπληρώνουν το *duck-typing* παρέχοντας έναν τρόπο ορισμού interfaces όταν άλλες τεχνικές όπως η `hasattr()` θα ήταν αδέξιες ή ανεπαίσθητα λανθασμένες (για παράδειγμα με magic methods). Τα ABC (abstract base class) εισάγουν εικονικές υποκλάσεις, οι οποίες είναι κλάσεις που δεν κληρονομούνται από μια κλάση, αλλά εξακολουθούν να αναγνωρίζονται από το `isinstance()` και από το `issubclass()`” βλ. την τεκμηρίωση του `module abc`. Η Python διαθέτει πολλά ενσωματωμένα ABC για δομές δεδομένων (στο `module collections.abc`), αριθμούς (στο `module numbers`), ροές (στο `module io`), εισαγωγή finders και loaders (στο `module importlib.abc`). Μπορείτε να δημιουργήσετε τα δικά σας ABC με το `module abc`.

**annotation** Μια ετικέτα που σχετίζεται με μια μεταβλητή, ένα χαρακτηριστικό κλάσης ή μια παράμετρος συνάρτησης ή τιμή που επιστρέφεται, που χρησιμοποιείται κατά σύμβαση ως *type hint*.

Δεν είναι δυνατή η πρόσβαση στα annotations των τοπικών μεταβλητών κατά το χρόνο εκτέλεσης, αλλά τα annotations των global μεταβλητών, των χαρακτηριστικών κλάσης και των συναρτήσεων αποθηκεύονται στο ειδικό χαρακτηριστικό `__annotations__` των modules, των κλάσεων και των συναρτήσεων, αντίστοιχα.

Βλ. *variable annotation*, *function annotation*, **PEP 484** και **PEP 526**, τα οποία περιγράφουν την λειτουργικότητα. Επίσης βλ. *annotations-howto* για τις βέλτιστες πρακτικές δουλεύοντας με annotations.

**όρισμα** Μια τιμή μεταβιβάζεται σε μία *function* (ή *method*) κατά την κλήση της συνάρτησης. Υπάρχουν δύο είδη ορισμάτων:

- *keyword argument*: ένα όρισμα πριν από ένα αναγνωριστικό (π.χ. `name=`) σε μια κλήση συνάρτησης ή περνώντας το ως τιμή σε ένα λεξικό πριν από `**`. Για παράδειγμα, το 3 και το 5 αποτελούν ορίσματα λέξεων-κλειδιών στις ακόλουθες κλήσεις προς `complex()`:

```
complex(real=3, imag=5)
complex(**{'real': 3, 'imag': 5})
```

- *positional argument*: ένα όρισμα που δεν είναι όρισμα keyword. Τα ορίσματα θέσης μπορούν να εμφανίζονται στην αρχή μιας λίστας ορισμάτων ή/και να μεταβιβάζονται ως στοιχεία ενός *iterable* πριν από `*`. Για παράδειγμα, το 3 και το 5 αποτελούν ορίσματα θέσης στις παρακάτω κλήσεις:

```
complex(3, 5)
complex(*(3, 5))
```

Τα ορίσματα εκχωρούνται στις ονομασμένες τοπικές μεταβλητές στο σώμα μια συνάρτησης. Βλ. την ενότητα *calls* για τους κανόνες που διέπουν αυτήν την εκχώρηση. Συντακτικά, οποιαδήποτε έκφραση μπορεί να χρησιμοποιηθεί για να αναπαραστήσει ένα όρισμα” η αξιολογούμενη τιμή εκχωρείται σε μια τοπική μεταβλητή.

Βλ. επίσης την εγγραφή του γλωσσarium για το *parameter*, την FAQ ερώτηση στο η διαφορά μεταξύ ορισμάτων και παραμέτρων, και **PEP 362**.

**ασύγχρονος διαχειριστής context** An object which controls the environment seen in an `async with` statement by defining `__aenter__()` and `__aexit__()` methods. Introduced by **PEP 492**.

**ασύγχρονος generator** Μια συνάρτηση που επιστρέφει έναν *asynchronous generator iterator*. Μοιάζει με μια συνάρτηση *coroutine* που ορίζεται με `async def` εκτός από ότι περιέχει εκφράσεις `yield` για την παραγωγή μιας σειράς τιμών που μπορούν να χρησιμοποιηθούν σε έναν `async for` βρόχο.

Συνήθως αναφέρεται σε μια συνάρτηση ασύγχρονου generator, αλλά μπορεί να αναφέρεται σε έναν *asynchronous generator iterator* σε ορισμένα contexts. Σε περιπτώσεις όπου το επιδιωκόμενο νόημα δεν είναι σαφές, με την χρήση των πλήρων όρων αποφεύγεται η ασάφεια.

Μια συνάρτηση ασύγχρονου generator μπορεί να περιέχει εκφράσεις `await`, καθώς και δηλώσεις `async for`, και `async with`.

**ασύγχρονος generator iterator** Ένα αντικείμενο που δημιουργήθηκε από μια συνάρτηση *asynchronous generator*.

This is an *asynchronous iterator* which when called using the `__anext__()` method returns an awaitable object which will execute the body of the asynchronous generator function until the next `yield` expression.

Each `yield` temporarily suspends processing, remembering the location execution state (including local variables and pending try-statements). When the *asynchronous generator iterator* effectively resumes with another awaitable returned by `__anext__()`, it picks up where it left off. See **PEP 492** and **PEP 525**.

**ασύγχρονος iterable** An object, that can be used in an `async for` statement. Must return an *asynchronous iterator* from its `__aiter__()` method. Introduced by **PEP 492**.

**ασύγχρονος iterator** An object that implements the `__aiter__()` and `__anext__()` methods. `__anext__()` must return an *awaitable* object. `async for` resolves the awaitables returned by an asynchronous iterator’s `__anext__()` method until it raises a `StopAsyncIteration` exception. Introduced by **PEP 492**.

**χαρακτηριστικό** Μια τιμή που σχετίζεται με ένα αντικείμενο που συνήθως αναφέρεται με όνομα χρησιμοποιώντας εκφράσεις με κουκκίδες. Για παράδειγμα, εάν ένα αντικείμενο *o* έχει ένα χαρακτηριστικό *a* θα αναφέρεται ως *o.a*.

Είναι δυνατό να δώσουμε σε ένα αντικείμενο ένα χαρακτηριστικό που το όνομα του δεν είναι αναγνωριστικό όπως ορίζεται από *identifiers*, για παράδειγμα χρησιμοποιώντας `setattr()`, αν επιτρέπεται από το αντικείμενο. Ένα τέτοιο χαρακτηριστικό δεν θα είναι προσβάσιμο χρησιμοποιώντας τις τελείες, και αντί αυτού θα πρέπει να ανακτηθεί χρησιμοποιώντας `getattr()`.

**awaitable** An object that can be used in an `await` expression. Can be a *coroutine* or an object with an `__await__()` method. See also **PEP 492**.



**BDFL** Ακρωνύμιο του *Benevolent Dictator For Life*, καλοκάγαθος δικτάτορας της ζωής, δηλαδή [Guido van Rossum](#), ο δημιουργός της Python.

**δυναδικό αρχείο** A *file object* able to read and write *bytes-like objects*. Examples of binary files are files opened in binary mode ('rb', 'wb' or 'rb+'), `sys.stdin.buffer`, `sys.stdout.buffer`, and instances of `io.BytesIO` and `gzip.GzipFile`.

Βλ. επίσης *text file* για ένα αντικείμενο τύπου αρχείο ικανό να διαβάσει και να γράψει `str` αντικείμενα.

**δανεική αναφορά** Στο C API της Python, μια δανεική αναφορά είναι μια αναφορά σε ένα αντικείμενο, όπου ο κώδικας που χρησιμοποιεί το αντικείμενο δεν κατέχει την αναφορά. Γίνεται ένας αχρησιμοποίητος δείκτης εάν το αντικείμενο καταστραφεί. Για παράδειγμα, μια διαδικασία garbage collection μπορεί να αφαιρέσει το τελευταίο *strong reference* από το αντικείμενο και έτσι να το καταστρέψει.

Συνίσταται η κλήση του `Py_INCREF()` στο *δανεική αναφορά* με σκοπό να μετατραπεί σε ένα *ισχυρή αναφορά* επιτόπου, εκτός όταν το αντικείμενο δεν μπορεί να καταστραφεί πριν από την τελευταία χρήση της δανεικής αναφοράς. Η συνάρτηση `Py_NewRef()` μπορεί να χρησιμοποιηθεί ώστε να δημιουργηθεί ένα *ισχυρή αναφορά*.

**bytes-like αντικείμενα** Ένα αντικείμενο που υποστηρίζει το `bufferobjects` και μπορεί να εξάγει ένα *C-contiguous* buffer. Αυτό περιλαμβάνει όλα τα αντικείμενα `bytes`, `bytearray`, και `array.array`, καθώς και πολλά κοινά `memoryview` αντικείμενα. Τα δυναδικού τύπου (bytes-like) αντικείμενα μπορούν να χρησιμοποιηθούν για διάφορες λειτουργίες που διαχειρίζονται δυναδικά δεδομένα" αυτά περιλαμβάνουν συμπίεση αποθήκευση σε δυναδικό αρχείο και αποστολή μέσω socket.

Ορισμένες λειτουργίες χρειάζονται τα δυναδικά δεδομένα να είναι μεταβλητά. Η τεκμηρίωση συχνά αναφέρεται σε αυτά ως «δυναδικά αντικείμενα ανάγνωσης-εγγραφής» (read-write bytes-like objects). Παραδείγματα μεταβλητών αντικειμένων προσωρινής αποθήκευσης περιέχουν `bytearray` και ένα `memoryview` ενός `bytearray`. Άλλες λειτουργίες απαιτούν την αποθήκευσης των δυναδικών δεδομένα σε αμετάβλητα αντικείμενα («δυναδικά αντικείμενα μόνο ανάγνωσης» (read-only bytes-like objects) παραδείγματα αυτών περιέχουν `bytes` και ένα `memoryview` ενός `bytes` αντικειμένου.

**bytecode** Ο πηγαίος κώδικας της Python μεταγλωττίζεται σε *bytecode*, η εσωτερική αναπαράσταση ενός προγράμματος Python στον διερμηνέα CPython. Το *bytecode* αποθηκεύεται επίσης προσωρινά ως `.pyc` αρχεία ώστε η εκτέλεση του ίδιου αρχείου να είναι γρηγορότερη την δεύτερη φορά εκτέλεσης (μπορεί να αποφευχθεί η εκ νέου μεταγλώττιση από τον πηγαίο κώδικα σε *bytecode*). Αυτή η «ενδιάμεση γλώσσα» λέγεται ότι τρέχει σε μια *virtual machine* που εκτελεί τον κώδικα μηχανής που αντιστοιχεί σε κάθε *bytecode*. Λάβετε υπόψη ότι τα *bytecode* δεν αναμένεται να λειτουργούν μεταξύ διαφορετικών εικονικών μηχανών Python, ούτε να είναι σταθερά μεταξύ των εκδόσεων της Python.

Μια λίστα από οδηγίες σχετικά με τα *bytecode* μπορεί να βρεθεί στην τεκμηρίωση για το module `dis`.

**callable** Ένα callable είναι ένα αντικείμενο που μπορεί να καλεστεί, πιθανά με ένα σύνολο ορισμάτων (βλ. *argument*), με την παρακάτω σύνταξη:

```
callable(argument1, argument2, ...)
```

Μια *function*, και κατ'επέκταση μια *method* είναι callable. Ένα στιγμιότυπο μια κλάσης που υλοποιεί τη μέθοδο `__call__()` είναι επίσης callable.

**callback** Μια subroutine συνάρτηση η οποία μεταβιβάζεται ως όρισμα που θα εκτελεστεί κάποια στιγμή στο μέλλον.

**κλάση** Ένα πρότυπο για τη δημιουργία αντικειμένων που ορίζονται από το χρήστη. Οι ορισμοί κλάσεων συνήθως περιέχουν ορισμούς μεθόδων που λειτουργούν σε στιγμιότυπα της κλάσης.

**μεταβλητή κλάσης** Μια μεταβλητή που ορίζεται σε μια κλάση και προορίζεται να τροποποιηθεί μόνο σε επίπεδο κλάσης (δηλ. όχι σε ένα στιγμιότυπο μιας κλάσης).

**coercion** The implicit conversion of an instance of one type to another during an operation which involves two arguments of the same type. For example, `int(3.15)` converts the floating point number to the integer 3, but in `3+4.5`, each argument is of a different type (one int, one float), and both must be converted to the same type before they can be added or it will raise a `TypeError`. Without coercion, all arguments of even compatible types would have to be normalized to the same value by the programmer, e.g., `float(3)+4.5` rather than just `3+4.5`.

**μυγαδικός αριθμός** Μια επέκταση του γνωστού συστήματος πραγματικών αριθμών στο οποίο όλοι οι αριθμοί εκφράζονται ως άθροισμα ενός πραγματικού μέρους και ενός φανταστικού μέρους. Οι φανταστικοί αριθμοί είναι πραγματικά πολλαπλάσια της φανταστικής μονάδα (η τετραγωνική ρίζα του  $-1$ ), που συχνά γράφονται  $i$  στα μαθηματικά ή  $j$  στη μηχανική. Η Python έχει ενσωματωμένη υποστήριξη για μυγαδικούς αριθμούς, οι οποίοι γράφονται με αυτόν τον τελευταίο συμβολισμό” το φανταστικό μέρος γράφεται με το επίθημα  $j$ , π.χ.,  $3+1j$ . Για να αποκτήσετε πρόσβαση σε σύνθετα ισοδύναμα το module `math`, χρησιμοποιήστε το `cmath`. Η χρήση μυγαδικών αριθμών είναι ένα αρκετά προηγμένο μαθηματικό χαρακτηριστικό. εάν δεν γνωρίζετε την ανάγκη τους, είναι σχεδόν σίγουρο ότι μπορείτε να τα αγνοήσετε με ασφάλεια.

**διαχειριστής context** An object which controls the environment seen in a `with` statement by defining `__enter__()` and `__exit__()` methods. See [PEP 343](#).

**context μεταβλητή** Μια μεταβλητή που μπορεί να έχει πολλές διαφορετικές τιμές ανάλογα με το context. Αυτό είναι κοινό στο Thread-Local Storage όπου κάθε εκτέλεση του νήματος μπορεί να έχει διαφορετική τιμή για μια μεταβλητή. Παρόλα αυτά, με τις context μεταβλητές, μπορεί να υπάρχουν πολλά περιβάλλοντα σε ένα νήμα εκτέλεσης και η κύρια χρήση για τις context μεταβλητές είναι η παρακολούθηση των μεταβλητών σε ταυτόχρονες διεργασίες. Βλ. `contextvars`.

**contiguous** Ένα buffer θεωρείται contiguous ακριβώς εάν είναι είτε *C-contiguous* είτε *Fortran contiguous*. Το buffer μηδενικών διαστάσεων είναι C και Fortran contiguous. Σε μονοδιάστατους πίνακες, τα στοιχεία πρέπει να τοποθετούνται στη μνήμη το ένα δίπλα στο άλλο, με σειρά αύξησης των δεικτών ξεκινώντας από το μηδέν. Σε πολυδιάστατους C-contiguous πίνακες, ο τελευταίος δείκτης μεταβάλλεται ταχύτερα όταν επισκέπτονται τα στοιχεία σε σειρά διεύθυνσης μνήμης. Ωστόσο, σε Fortran contiguous πίνακες, ο πρώτος δείκτης μεταβάλλεται πιο γρήγορα.

**coroutine** Οι coroutines είναι μια πιο γενικευμένη μορφή subroutines. Οι subroutines εισάγονται σε ένα σημείο και εξάγονται σε άλλο σημείο. Οι coroutines μπορεί να εισαχθούν, να εξαχθούν και να συνεχιστούν σε πολλά διαφορετικά σημεία. Μπορούν να υλοποιηθούν με την δήλωση `async def`. Βλ. επίσης [PEP 492](#).

**coroutine συνάρτηση** Μια συνάρτηση που επιστρέφει ένα *coroutine* αντικείμενο. Μια συνάρτηση coroutine μπορεί να ορίζεται από τη δήλωση `async def`, και μπορεί να περιέχει `await`, `async for`, και `async with` λέξεις κλειδιά. Αυτές εισήχθησαν από το [PEP 492](#).

**CPython** Η κανονική υλοποίηση της γλώσσας προγραμματισμού Python, όπως διανέμεται στο [python.org](#). Ο όρος «CPython» χρησιμοποιείται όταν είναι απαραίτητο για την διάκριση αυτής της υλοποίησης από άλλες όπως η *Jython* ή η *IronPython*.

**decorator** Μια συνάρτηση που επιστρέφει μια άλλη συνάρτηση, συνήθως εφαρμόζεται ως μετασχηματισμός συνάρτησης χρησιμοποιώντας την `@wrapper` σύνταξη. Συνηθισμένα παραδείγματα για τους decorators είναι `classmethod()` και `staticmethod()`.

Η σύνταξη του decorator είναι απλώς καλλωπιστική, οι ακόλουθοι δύο ορισμοί συναρτήσεων είναι σημασιολογικά ισοδύναμοι:

```
def f(arg):
    ...
f = staticmethod(f)

@staticmethod
def f(arg):
    ...
```

Η ίδια έννοια υπάρχει για τις κλάσεις, αλλά χρησιμοποιείται λιγότερο συχνά εκεί. Βλ. την τεκμηρίωση για function definitions και class definitions για περισσότερα σχετικά με τους decorators.

**descriptor** Any object which defines the methods `__get__()`, `__set__()`, or `__delete__()`. When a class attribute is a descriptor, its special binding behavior is triggered upon attribute lookup. Normally, using `a.b` to get, set or delete an attribute looks up the object named `b` in the class dictionary for `a`, but if `b` is a descriptor, the respective descriptor method gets called. Understanding descriptors is a key to a deep understanding of Python because they are the basis for many features including functions, methods, properties, class methods, static methods, and reference to super classes.

Για περισσότερες πληροφορίες αναφορικά με τις μεθόδους των descriptors, βλ. [see descriptors](#) ή το Πρακτικός οδηγός για τη χρήση του Descriptor.

**λεξικό** An associative array, where arbitrary keys are mapped to values. The keys can be any object with `__hash__()` and `__eq__()` methods. Called a hash in Perl.

**κατανόηση λεξικού** Ένα συμπαγής τρόπος για να επεξεργαστείτε όλα ή μέρος των στοιχείων σε ένα επαναληπτικό και να επιστραφεί ένα με λεξικό με τα αποτελέσματα. `results = {n: n ** 2 for n in range(10)}` δημιουργεί ένα λεξικό που περιέχει το κλειδί `n` που αντιστοιχίζεται με την τιμή `n ** 2`. Βλ. [comprehensions](#).

**όψη λεξικού** Τα αντικείμενα που επιστρέφονται από `dict.keys()`, `dict.values()`, και `dict.items()` καλούνται όψεις λεξικού. Αυτές παρέχουν μια δυναμική όψη των των εγγραφών του λεξικού, που σημαίνει ότι όταν το λεξικό μεταβάλλεται, η όψη αντικατοπτρίζει αυτές τις αλλαγές. Για να αναγκάσετε την όψη λεξικού να γίνει μια πλήρης λίστα χρησιμοποιήστε το `list(dictview)`. Βλ. [dict-views](#).

**docstring** A string literal which appears as the first expression in a class, function or module. While ignored when the suite is executed, it is recognized by the compiler and put into the `__doc__` attribute of the enclosing class, function or module. Since it is available via introspection, it is the canonical place for documentation of the object.

**duck-typing** Ένα στυλ προγραμματισμού που δεν εξετάζει τον τύπο ενός αντικειμένου για να προσδιορίσει αν έχει τη σωστή διεπαφή αντίθετα, η μέθοδος ή το χαρακτηριστικό καλείται απλώς ή χρησιμοποιείται («If it looks like a duck and quacks like a duck, it must be a duck.») Δίνοντας έμφαση στις διεπαφές και όχι σε συγκεκριμένους τύπους, ο καλά σχεδιασμένος κώδικας βελτιώνει την ευελιξία του επιτρέποντας την πολυμορφική υποκατάσταση. Ο τύπος duck-typing αποφεύγει δοκιμές χρησιμοποιώντας `type()` ή `isinstance()`. (Σημείωση, ωστόσο, ότι ο τύπος πάπιας *duck-typing* μπορεί να συμπληρωθεί με [abstract base classes](#).) Αντί αυτού, συνήθως χρησιμοποιεί δοκιμές `hasattr()` ή προγραμματισμό [EAFP](#).

**EAFP** Πιο εύκολο να ζητήσεις συγχώρεση παρά άδεια. Αυτό το κοινό στυλ προγραμματισμού σε Python προϋποθέτει την ύπαρξη έγκυρων κλειδιών ή χαρακτηριστικών και συλλαμβάνει εξαιρέσεις εάν η υπόθεση αποδεχθεί εσφαλμένη. Αυτό το καθαρό και γρήγορο στυλ χαρακτηρίζεται από την παρουσία πολλών δηλώσεων `try` και `except`. Η τεχνική έρχεται σε αντίθεση με το στυλ που είναι [LBYL](#) κοινό σε πολλές άλλες γλώσσες, όπως η C.

**έκφραση** Ένα κομμάτι σύνταξης που μπορεί να αξιολογηθεί σε κάποια τιμή. Με άλλα λόγια, μια έκφραση είναι μια συσσώρευση στοιχείων έκφρασης όπως κυριολεξία, ονόματα, πρόσβαση χαρακτηριστικών, τελεστές ή κλήσεις συναρτήσεων που όλες επιστρέφουν μια τιμή. Σε αντίθεση με πολλές άλλες γλώσσες, δεν είναι όλες οι γλωσσικές δομές εκφράσεις. Υπάρχουν επίσης [statements](#) που δεν μπορούν να χρησιμοποιηθούν ως εκφράσεις, όπως το `while`. Οι αναθέσεις τιμών είναι επίσης δηλώσεις όχι εκφράσεις.

**module επέκτασης** Ένα module γραμμένο σε C ή C++, που χρησιμοποιείται από το C API της Python για να αλληλεπιδράσουν με τον πυρήνα και με τον κώδικα του χρήστη.

**f-string** Οι κυριολεκτικές συμβολοσειρές χρησιμοποιούν με πρόθεμα `'f'` ή `'F'` ονομάζονται συνήθως «f-strings» που είναι συντομογραφία του formatted string literals. Βλ. επίσης [PEP 498](#).

**αντικείμενο αρχείου** An object exposing a file-oriented API (with methods such as `read()` or `write()`) to an underlying resource. Depending on the way it was created, a file object can mediate access to a real on-disk file or to another type of storage or communication device (for example standard input/output, in-memory buffers, sockets, pipes, etc.). File objects are also called *file-like objects* or *streams*.

Στην πραγματικότητα υπάρχουν τρεις κατηγορίες αντικειμένων αρχείου *raw δυαδικά αρχεία*, *buffered δυαδικά αρχεία* και *αρχεία κειμένου*. Οι διεπαφές τους ορίζονται στην ενότητα [io](#). Ο κανονικός τρόπος για να δημιουργήσετε ένα αντικείμενο αρχείου είναι χρησιμοποιώντας την συνάρτηση `open()`.

**αντικείμενο που μοιάζει με αρχείο** Ένα συνώνυμο με το [file object](#).

**κωδικοποίηση συστήματος αρχείων και χειριστής σφαλμάτων** Η κωδικοποίηση και ο χειριστής σφαλμάτων χρησιμοποιείται από την Python για την αποκωδικοποίηση των bytes από το λειτουργικό σύστημα και την κωδικοποίηση σε Unicode για το λειτουργικό σύστημα.

Η κωδικοποίηση συστήματος αρχείων μπορεί να εγγραφεί την επιτυχημένη αποκωδικοποίηση όλων των bytes κάτω από 128. Εάν η κωδικοποίηση συστήματος αρχείων δεν παρέχει αυτήν την εγγύηση, οι συναρτήσεις API μπορούν να εγείρουν ένα `UnicodeError`.

Οι συναρτήσεις `sys.getfilesystemencoding()` και `sys.getfilesystemencodeerrors()` μπορούν να χρησιμοποιηθούν για να λάβετε την κωδικοποίηση του συστήματος αρχείων και του χειριστή σφαλμάτων.

Ο *filesystem encoding and error handler* διαμορφώνονται κατά την εκκίνηση της Python από τη συνάρτηση `PyConfig_Read()` βλ. `filesystem_encoding` και `filesystem_errors` μέλη του `PyConfig`.

Βλ. επίσης το *locale encoding*.

**finder** Ένα αντικείμενο που προσπαθεί να βρει το *loader* για ένα module που εισήχθη.

Since Python 3.3, there are two types of finder: *meta path finders* for use with `sys.meta_path`, and *path entry finders* for use with `sys.path_hooks`.

See [PEP 302](#), [PEP 420](#) and [PEP 451](#) for much more detail.

**ακέραια διαίρεση** Η μαθηματική διαίρεση που στρογγυλοποιεί προς τα κάτω στον κοντινότερο ακέραιο. Ο τελεστής ακέραιας διαίρεσης είναι `//`. Για παράδειγμα, η έκφραση `11 // 4` αξιολογείται σε 2 σε αντίθεση με την τιμή `2.75` που επιστρέφεται από την διαίρεση με υποδιαστολή. Σημείωση ότι `(-11) // 4` κάνει `-3` επειδή αυτή είναι η στρογγυλοποίηση προς τα κάτω του `-2.75`. Βλ. [PEP 238](#).

**συνάρτηση** Μια σειρά από δηλώσεις που επιστρέφουν κάποια τιμή σε αυτόν που την κάλεσε. Σε αυτές μπορούν να περαστούν κανένα ή περισσότερα *ορίσματα* που μπορεί να χρησιμοποιηθεί για την εκτέλεση. Βλ. επίσης τις ενότητες *parameter*, *method*, και *the function*.

**συνάρτηση annotation** Ένας *annotation* μιας παραμέτρου συνάρτησης ή μιας τιμής επιστροφής.

Οι συναρτήσεις annotations συχνά χρησιμοποιούνται για *υποδείξεις τύπου*: για παράδειγμα, αυτή η συνάρτηση αναμένεται να πάρει δύο ορίσματα `int` και επίσης αναμένεται να έχει μία επιστρεφόμενη τιμή `int`:

```
def sum_two_numbers(a: int, b: int) -> int:
    return a + b
```

Η σύνταξη συνάρτησης annotation αναλύεται στην ενότητα *function*.

Βλ. *variable annotation* και [PEP 484](#), που περιγράφει αυτή την λειτουργικότητα. Επίσης βλ. *annotations-howto* για τις καλύτερες πρακτικές δουλεύοντας με annotations.

**\_\_future\_\_** Ένα future statement, `from __future__ import <feature>`, καθοδηγεί τον μεταγλωττιστή να μεταγλωττίσει το τρέχον module χρησιμοποιώντας σύνταξη ή σημασιολογία που θα γίνει η τυπική σε μελλοντική έκδοση της Python. Το module `__future__` τεκμηριώνει τις πιθανές τιμές του *feature*. Με την εισαγωγή αυτής της λειτουργικής μονάδας και την αξιολόγηση των μεταβλητών της, μπορείτε να δείτε πότε μια νέα δυνατότητα προστέθηκε για πρώτη φορά στην γλώσσα και πότε θα γίνει (ή έγινε) η προεπιλογή:

```
>>> import __future__
>>> __future__.division
_Feature((2, 2, 0, 'alpha', 2), (3, 0, 0, 'alpha', 0), 8192)
```

**συλλογή απορριμάτων** Η διαδικασία απελευθέρωσης της μνήμης όταν δεν χρησιμοποιείται άλλο. Η Python εκτελεί συλλογή απορριμάτων μέσω καταμέτρησης αναφορών και ενός κυκλικού συλλέκτη σκουπιδιών που είναι σε θέση να ανιχνεύει και να σπάει τους κύκλους αναφοράς. Ο συλλέκτης απορριμάτων μπορεί να ελεγχθεί χρησιμοποιώντας το module `gc`.

**generator** Μια συνάρτηση που επιστρέφει ένα *generator iterator*. Μοιάζει με μια κανονική συνάρτηση εκτός από το ότι περιέχει εκφράσεις `yield` για την παραγωγή μιας σειράς τιμών που μπορούν να χρησιμοποιηθούν σε έναν βρόχο *for* ή που μπορούν να ανακτηθούν μία τη φορά με την συνάρτηση `next()` function.

Συνήθως αναφέρεται σε μια συνάρτηση *generator*, αλλά μπορεί να αναφέρεται σε έναν *generator iterator* σε μερικά contexts. Σε περιπτώσεις όπου το επιδιωκόμενο νόημα δεν είναι σαφές, η χρήση των πλήρων όρων αποφεύγει την ασάφεια.

**generator iterator** Ένα αντικείμενο που δημιουργείται από μια συνάρτηση *generator*.

Each `yield` temporarily suspends processing, remembering the location execution state (including local variables and pending try-statements). When the *generator iterator* resumes, it picks up where it left off (in contrast to functions which start fresh on every invocation).

**generator έκφραση** Μια έκφραση που επιστρέφει έναν iterator. Μοιάζει με κανονική έκφραση που ακολουθείται από μια πρόταση `for` που ορίζει μια μεταβλητή βρόχου, ένα εύρος και μια προαιρετική πρόταση `if`. Η συνδυασμένη έκφραση δημιουργεί τιμές για μια συνάρτηση εγκλεισμού:

```
>>> sum(i*i for i in range(10))           # sum of squares 0, 1, 4, ... 81
285
```

**γενική συνάρτηση** Μια συνάρτηση που αποτελείται από πολλαπλές συναρτήσεις που υλοποιούν την ίδια λειτουργία για διαφορετικούς τύπους. Ποια υλοποίηση πρέπει να χρησιμοποιηθεί κατά τη διάρκεια μια κλήσης καθορίζεται από τον αλγόριθμο αποστολής.

Βλ. επίσης την καταχώρηση του *single dispatch*, τον decorator `functools singledispatch()` και [PEP 443](#).

**γενικός τύπος** Ένας *type* που μπορεί να παραμετροποιηθεί” συνήθως μια container class, όπως `list` ή `dict`. Χρησιμοποιείται για *type hints* και *annotations*.

Για περισσότερες λεπτομέρειες, βλ. generic alias types [PEP 483](#), [PEP 484](#), [PEP 585](#), και το module `typing`.

**GIL** Βλ. *global interpreter lock*.

**global interpreter lock** Ο μηχανισμός που χρησιμοποιείται από τον διερμηνέα *CPython* για να διασφαλίσει ότι μόνο ένα νήμα εκτελεί Python *bytecode* κάθε φορά. Αυτό απλοποιεί την υλοποίηση *CPython* δημιουργώντας το μοντέλο αντικειμένου (συμπεριλαμβανομένων κρίσιμων ενσωματωμένων τύπων όπως π.χ. `dict`) έμμεσα ασφαλές έναντι ταυτόχρονης πρόσβασης. Το κλείδωμα ολόκληρου του διερμηνέα διευκολύνει τον διερμηνέα να είναι πολλαπλών νημάτων, εις βάρος του μεγάλου μέρους του παραλληλισμού που παρέχουν οι μηχανές πολλαπλών επεξεργαστών.

Ωστόσο, ορισμένες λειτουργικές μονάδες επέκτασης, είτε τυπικές είτε τρίτων, έχουν σχεδιαστεί έτσι ώστε να απελευθερώνουν το GIL όταν εκτελούν εργασίες εντατικών υπολογισμών όπως συμπίεση ή κατακερματισμός. Επίσης, το GIL απελευθερώνεται πάντα όταν εκτελείτε I/O.

Προηγούμενες προσπάθειες να δημιουργηθεί ένας διερμηνέας «ελεύθερων-νημάτων» (αυτός που κλειδώνει τα κοινόχρηστα δεδομένα με πολύ πιο λεπτομερή ευαισθησία) δεν ήταν επιτυχείς επειδή η απόδοση υποχώρησε στην κοινή περίπτωση ενός επεξεργαστή. Πιστεύεται ότι η υπέρβαση αυτού του προβλήματος απόδοσης θα κάνουν πολύ πιο περίπλοκη και επομένως πιο δαπανηρή στην συντήρηση.

**hash-based pyc** Ένα αρχείο κρυφής μνήμης *bytecode* που χρησιμοποιεί τον κατακερματισμό και όχι τον χρόνο τροποποίησης του αντίστοιχου αρχείου προέλευσης για να προσδιορίσει την εγκυρότητα του. Βλ. `pyc-invalidation`.

**hashable** An object is *hashable* if it has a hash value which never changes during its lifetime (it needs a `__hash__()` method), and can be compared to other objects (it needs an `__eq__()` method). Hashable objects which compare equal must have the same hash value.

Η ύπαρξη *hashable* κάνει ένα αντικείμενο να μπορεί να χρησιμοποιηθεί ως κλειδί λεξικού και ως μέλος ενός συνόλου, επειδή αυτές οι δομές δεδομένων χρησιμοποιούν τιμές κατακερματισμού.

Τα περισσότερα από τα αμετάβλητα ενσωματωμένα αντικείμενα της Python μπορούν να κατακερματιστούν” τα μεταβλητά κοντέινερ (όπως οι λίστες ή τα λεξικά) δεν είναι” τα αμετάβλητα κοντέινερ (όπως πλειάδες και τα frozensets) μπορούν να κατακερματιστούν μόνο εάν τα στοιχεία τους είναι κατακερματισμένα. Τα αντικείμενα που είναι στιγμιότυπα κλάσεων που ορίζονται από το χρήστη μπορούν να κατακερματιστούν από προεπιλογή. Όλα συγκρίνονται άνισα εκτός από τον εαυτό τους) και η τιμή κατακερματισμού τους προέρχεται από το `id()`.



**IDLE** Ένα ολοκληρωμένο περιβάλλον ανάπτυξης και μάθησης για την Python. idle είναι ένα βασικό περιβάλλον επεξεργασίας και διερμηνέα που συνοδεύεται από την τυπική διανομή της Python.

**immutable** Ένα αντικείμενο με σταθερή τιμή. Τα αμετάβλητα αντικείμενα περιλαμβάνουν αριθμούς, συμβολοσειρές και πλειάδες. Ένα τέτοιο αντικείμενο δεν μπορεί να αλλάξει. Ένα νέο αντικείμενο πρέπει να δημιουργηθεί εάν πρέπει να αποθηκευτεί μια διαφορετική τιμή. Παίζουν σημαντικό ρόλο σε μέρη όπου μια σταθερά απαιτείται, για παράδειγμα ως κλειδί σε ένα λεξικό.

**εισαγόμενο path** Μια λίστα από τοποθεσίες (ή *καταχωρίσεις διαδρομής*) που μπορούν να αναζητηθούν *path based finder* για να εισαχθούν modules. Κατά την διαδικασία εισαγωγής, αυτή η λίστα με τοποθεσίες συνήθως έρχεται από `sys.path`, αλλά για τα υποπακέτα μπορεί επίσης να έρθει από το χαρακτηριστικό του πακέτου γονέα `__path__`.

**εισαγωγή** Η διαδικασία κατά την οποία ο κώδικας της Python σε ένα module είναι διαθέσιμη στον κώδικα Python ενός άλλου module.

**εισαγωγέας** Ένα αντικείμενο μπορεί και να αναζητεί και να φορτώνει ένα module” και ένα *finder* και *loader* αντικείμενο.

**διαδραστικός** Η Python έχει έναν διαδραστικό διερμηνέα όπου σημαίνει ότι μπορείς να εισάγεις δηλώσεις και εκφράσεις στην εισαγωγή εντολών του διερμηνέα, εκτελώντας τις άμεσα και εμφανίζοντας τα αντικείμενα. Απλώς εκκινήστε την python χωρίς ορίσματα (πιθανώς επιλέγοντας το από το κύριο μενού του υπολογιστή σας). Αποτελεί έναν αποδοτικό τρόπο για να δοκιμάστε νέες ιδέες ή να εξετάσετε λειτουργικές μονάδες και πακέτα (θυμηθείτε `help(x)`).

**interpreted** Η Python είναι μια interpreted γλώσσα, σε αντίθεση με μια μεταγλωττισμένη, αν και η διάκριση μπορεί να είναι και θολή λόγω της παρουσία του bytecode μεταγλωττιστή. Αυτό σημαίνει ότι τα αρχεία προέλευσης μπορούν να εκτελεστούν απευθείας χωρίς να δημιουργηθεί ρητά ένα εκτελέσιμο αρχείο που στην συνέχεια εκτελείται. Οι interpreted γλώσσες συνήθως έχουν μικρότερο κύκλο ανάπτυξης/εντοπισμού σφαλμάτων από τις μεταγλωττισμένες, αν και τα προγράμματά τους γενικά εκτελούνται πιο αργά. Βλ. επίσης *interactive*.

**τερματισμός λειτουργίας διερμηνέα** Όταν ζητείται τερματισμός λειτουργίας, ο διερμηνέας της Python εισέρχεται σε μια ειδική φάση όπου απελευθερώνει σταδιακά όλους τους διατιθέμενους πόρους, όπως λειτουργικές μονάδες και πολλαπλές κρίσιμες εσωτερικές δομές. Επίσης πραγματοποιεί αρκετές κλήσεις στο *συλλέκτης σκουπιδιών*. Αυτό μπορεί να ενεργοποιήσει την εκτέλεση κώδικα σε καταστροφείς που ορίζονται από το χρήστη ή σε callbacks ασθενούς ανταποκρίσεις. Ο κώδικας που εκτελείται κατά τη φάση τερματισμού λειτουργίας μπορεί να συναντήσει διάφορες εξαιρέσεις, καθώς οι πόροι στους οποίους βασίζεται ενδέχεται να μην λειτουργούν πλέον (συνήθη παραδείγματα είναι οι λειτουργικές μονάδες βιβλιοθήκης ή ο μηχανισμός ειδοποιήσεων).

Ο βασικός λόγος τερματισμού λειτουργίας του διερμηνέα είναι ότι το `__main__` module ή ολοκληρώθηκε η εκτέλεση του κώδικα που έτρεχε.

**iterable** An object capable of returning its members one at a time. Examples of iterables include all sequence types (such as `list`, `str`, and `tuple`) and some non-sequence types like `dict`, *file objects*, and objects of any classes you define with an `__iter__()` method or with a `__getitem__()` method that implements *Sequence* semantics.

Iterables can be used in a `for` loop and in many other places where a sequence is needed (`zip()`, `map()`, ...). When an iterable object is passed as an argument to the built-in function `iter()`, it returns an iterator for the object. This iterator is good for one pass over the set of values. When using iterables, it is usually not necessary to call `iter()` or deal with iterator objects yourself. The `for` statement does that automatically for you, creating a temporary unnamed variable to hold the iterator for the duration of the loop. See also *iterator*, *sequence*, and *generator*.

**iterator** An object representing a stream of data. Repeated calls to the iterator’s `__next__()` method (or passing it to the built-in function `next()`) return successive items in the stream. When no more data are available a `StopIteration` exception is raised instead. At this point, the iterator object is exhausted and any further calls to its `__next__()` method just raise `StopIteration` again. Iterators are required to have an `__iter__()` method that returns the iterator object itself so every iterator is also iterable and may be used in most places where other iterables are accepted. One notable exception is code which attempts multiple iteration passes. A container object (such as a `list`) produces a fresh new iterator each time you pass it to the

`iter()` function or use it in a `for` loop. Attempting this with an iterator will just return the same exhausted iterator object used in the previous iteration pass, making it appear like an empty container.

Περισσότερες πληροφορίες μπορούν να βρεθούν στο `typeiter`.

**Λεπτομέρεια υλοποίησης CPython:** CPython does not consistently apply the requirement that an iterator define `__iter__()`.

**συνάρτηση key** Μια συνάρτηση κλειδί ή μια συνάρτηση ταξινόμησης είναι μια δυνατότητα κλήσης που επιστρέφει μια τιμή που χρησιμοποιείται για ταξινόμηση ή διάταξη. Για παράδειγμα, `locale.strxfrm()` χρησιμοποιείται για την παραγωγή ενός κλειδιού ταξινόμησης που γνωρίζει τις συμβάσεις ταξινόμησης για συγκεκριμένες τοπικές ρυθμίσεις.

Ένα αριθμός εργαλείων στην Python δέχεται βασικές συναρτήσεις για τον έλεγχο του τρόπου με τον οποίο τα στοιχεία ταξινομούνται ή ομαδοποιούνται. Αυτά περιέχουν `min()`, `max()`, `sorted()`, `list.sort()`, `heapq.merge()`, `heapq.nsmallest()`, `heapq.nlargest()`, και `itertools.groupby()`.

There are several ways to create a key function. For example, the `str.lower()` method can serve as a key function for case insensitive sorts. Alternatively, a key function can be built from a `lambda` expression such as `lambda r: (r[0], r[2])`. Also, the `operator` module provides three key function constructors: `attrgetter()`, `itemgetter()`, and `methodcaller()`. See the Sorting HOW TO for examples of how to create and use key functions.

**όρισμα keyword** Βλ. *argument*.

**lambda** Μια ανώνυμη ενσωματωμένη συνάρτηση που αποτελείται από μια μοναδική *expression* η οποία αξιολογείται όταν καλείται η συνάρτηση. Η σύνταξη για τη δημιουργία μιας συνάρτησης `lambda` είναι `lambda [parameters]: expression`

**LBYL** Look before you leap. Αυτό το στυλ κωδικοποίησης ελέγχει ρητά τις προϋποθέσεις πριν πραγματοποιήσει κλήσεις ή αναζητήσεις. Αυτό το στυλ έρχεται σε αντίθεση με την προσέγγιση *EAFP* και χαρακτηρίζεται από την παρουσία πολλών δηλώσεων `if`.

Σε ένα περιβάλλον πολλαπλών νημάτων, η προσέγγιση LBYL μπορεί να διακινδυνεύσει να εισάγει μια συνθήκη αγώνα μεταξύ «the Looking» και «the leaping». Για παράδειγμα ο κώδικας, `if key in mapping: return mapping[key]` μπορεί να αποτύχει εάν ένα άλλο νήμα αφαιρέσει το `key` από το `mapping` μετά τη δοκιμή, αλλά πριν από την αναζήτηση. Αυτό το πρόβλημα μπορεί να λυθεί με κλειδώματα ή χρησιμοποιώντας την προσέγγιση EAFP.

**τοπική κωδικοποίηση** On Unix, it is the encoding of the LC\_CTYPE locale. It can be set with `locale.setlocale(locale.LC_CTYPE, new_locale)`.

On Windows, it is the ANSI code page (ex: cp1252).

`locale.getpreferredencoding(False)` can be used to get the locale encoding.

Python uses the *filesystem encoding and error handler* to convert between Unicode filenames and bytes filenames.

**λίστα** A built-in Python *sequence*. Despite its name it is more akin to an array in other languages than to a linked list since access to elements is O(1).

**list comprehension** Ένα συμπαγής τρόπος για να επεξεργαστείτε όλα ή μέρος των στοιχείων σε μια ακολουθία και να επιστρέψετε μια λίστα με τα αποτελέσματα. `result = ['{: #04x}'.format(x) for x in range(256) if x % 2 == 0]` δημιουργεί μια λίστα συμβολοσειρών που περιέχουν ζυγούς δεκαεξαδικούς αριθμούς (0x..) στο εύρος από 0 έως 255. Η πρόταση `if` είναι προαιρετική. Εάν παραλειφθεί, όλα τα στοιχεία στο `range(256)` υποβάλλονται σε επεξεργασία.

**loader** An object that loads a module. It must define a method named `load_module()`. A loader is typically returned by a *finder*. See **PEP 302** for details and `importlib.abc.Loader` for an *abstract base class*.

**μαγική μέθοδος** Ένα άτυπο συνώνυμο για *special method*.

**mapping** A container object that supports arbitrary key lookups and implements the methods specified in the Mapping or MutableMapping abstract base classes. Examples include `dict`, `collections.defaultdict`, `collections.OrderedDict` and `collections.Counter`.

**meta path finder** Ένας *finder* που επιστράφηκε με αναζήτηση στο `sys.meta_path`. Οι *finders* μετα-διαδρομής σχετίζονται, αλλά διαφέρουν από τα *finders entry διαδρομής*.

Βλ. `importlib.abc.MetaPathFinder` για τις μεθόδους που υλοποιούν οι meta path finders.

**μετα-κλάση** Η κλάση μιας κλάσης. Οι ορισμοί κλάσης δημιουργούν ένα όνομα κλάσης, ένα λεξικό κλάσης και μια λίστα βασικών κλάσεων. Η μετα-κλάση είναι υπεύθυνη για την απόκτηση αυτών των τριών ορισμάτων και την δημιουργία της κλάσης. Οι περισσότερες αντικειμενοστρεφείς γλώσσες προγραμματισμού παρέχουν μια προεπιλεγμένη υλοποίηση. Αυτό που κάνει την Python ξεχωριστή είναι ότι είναι δυνατή η δημιουργία προσαρμοσμένων μετακλάσεων. Οι περισσότεροι χρήστες δεν χρειάζονται ποτέ αυτό το εργαλείο, αλλά όταν παραστεί ανάγκη, αυτό το εργαλείο, οι μετα-κλάσεις μπορούν να παρέχουν ισχυρές, κομψές λύσεις. Έχουν χρησιμοποιηθεί για την καταγραφή πρόσβασης χαρακτηριστικών, την προσθήκη ασφάλειας νημάτων, την παρακολούθηση δημιουργίας αντικειμένων, την υλοποίηση *singletons*, και πολλές άλλες εργασίες.

Περισσότερες πληροφορίες μπορούν να βρεθούν στο *metaclasses*.

**μέθοδος** Μια συνάρτηση που ορίζεται μέσα στο σώμα μιας κλάσης. Εάν καλείται ως χαρακτηριστικό μιας περίπτωσης αυτής της κλάσης, η μέθοδος θα λάβει αντικείμενο περίπτωσης ως πρώτο της *argument* (το οποίο συνήθως ονομάζεται `self`). Βλ. *function* και *nested scope*.

**σειρά ανάλυσης μεθόδων** Method Resolution Order is the order in which base classes are searched for a member during lookup. See [The Python 2.3 Method Resolution Order](#) for details of the algorithm used by the Python interpreter since the 2.3 release.

**module** Ένα αντικείμενο που χρησιμεύει ως οργανωτική μονάδα του κώδικα της Python. Τα modules έχουν έναν χώρο ονομάτων που περιέχει αυθαίρετα αντικείμενα Python. Τα modules φορτώνονται στην Python με την διαδικασία *importing*.

Βλ. επίσης *package*.

**τεχνικές προδιαγραφές module** Ένα namespace που περιέχει τις πληροφορίες που σχετίζονται με την εισαγωγή που χρησιμοποιούνται για την φόρτωση ενός module. Μια περίπτωση του `importlib.machinery.ModuleSpec`.

**MRO** Βλ. *method resolution order*.

**mutable** Τα ευμετάβλητα αντικείμενα μπορούν να αλλάξουν τις τιμές αλλά να κρατήσουν τα `id()`. Βλ. επίσης *immutable*.

**named tuple** Ο όρος «named tuple» εφαρμόζεται για οποιονδήποτε τύπο ή κλάση που κληρονομείται από την πλειάδα και των οποίων τα στοιχεία μπορούν να ευρετηριοποιηθούν είναι προσβάσιμα χρησιμοποιώντας επώνυμα χαρακτηριστικά. Ο τύπος ή η κλάση μπορεί να έχει και άλλα χαρακτηριστικά.

Πολλοί ενσωματωμένοι τύποι είναι named tuples, συμπεριλαμβανομένων των τιμών που επιστρέφονται από `time.localtime()` και `os.stat()`. Ένα άλλο παράδειγμα είναι το `sys.float_info`:

```
>>> sys.float_info[1]           # indexed access
1024
>>> sys.float_info.max_exp      # named field access
1024
>>> isinstance(sys.float_info, tuple) # kind of tuple
True
```

Some named tuples are built-in types (such as the above examples). Alternatively, a named tuple can be created from a regular class definition that inherits from `tuple` and that defines named fields. Such a class can be written by hand or it can be created with the factory function `collections.namedtuple()`. The latter technique also adds some extra methods that may not be found in hand-written or built-in named tuples.

**namespace** Το μέρος όπου αποθηκεύεται μια μεταβλητή. Τα namespaces υλοποιούνται ως λεξικά. Υπάρχουν οι τοπικοί, οι καθολικοί και οι ενσωματωμένοι namespaces καθώς και οι ένθετοι namespaces σε αντικείμενα (σε μεθόδους). Για παράδειγμα οι συναρτήσεις `builtins.open` και `os.open()` διακρίνονται από τους χώρους ονομάτων τους. Οι χώροι ονομάτων βοηθούν επίσης την αναγνωσιμότητα και τη συντηρησιμότητα καθιστώντας σαφές ποιο module υλοποιεί μια λειτουργία. Για παράδειγμα, γράφοντας



`random.seed()` ή `itertools.islice()` καθιστά σαφές ότι αυτές οι συναρτήσεις υλοποιούνται από τα module `random` και `itertools`, αντίστοιχα.

**πακέτο namespace** A [PEP 420 package](#) which serves only as a container for subpackages. Namespace packages may have no physical representation, and specifically are not like a [regular package](#) because they have no `__init__.py` file.

Βλ. επίσης [module](#).

**nested scope** Η δυνατότητα αναφοράς σε μια μεταβλητή σε έναν περικλειόμενο ορισμό. Για παράδειγμα μια συνάρτηση που ορίζεται μέσα σε μια άλλη συνάρτηση μπορεί να αναφέρεται σε μεταβλητές στην εξωτερική συνάρτηση. Σημειώστε ότι τα ένθετα πεδία από προεπιλογή λειτουργούν μόνο για αναφορά και όχι για εκχώρηση. Οι τοπικές μεταβλητές διαβάζονται και γράφονται στο εσωτερικό πεδίο εφαρμογής. Ομοίως, οι καθολικές μεταβλητές διαβάζουν και γράφουν στον καθολικό χώρο ονομάτων. Το `nonlocal` επιτρέπει την εγγραφή σε εξωτερικά πεδία.

**κλάση νέου στυλ** Old name for the flavor of classes now used for all class objects. In earlier Python versions, only new-style classes could use Python's newer, versatile features like `__slots__`, descriptors, properties, `__getattr__()`, class methods, and static methods.

**αντικείμενο** Οποιαδήποτε δεδομένα με κατάσταση (χαρακτηριστικά ή τιμή) και καθορισμένη συμπεριφορά (μέθοδοι). Επίσης, η τελική βασική κλάση οποιασδήποτε [new-style class](#).

**πακέτο** Ένα Python [module](#) που μπορεί να περιέχει submodules ή αναδρομικά, υποπακέτα. Τεχνικά, ένα πακέτο είναι μια λειτουργική μονάδα Python με ένα `__path__` χαρακτηριστικό.

Βλ. επίσης [regular package](#) και [namespace package](#).

**παράμετρος** Μια έγκυρη οντότητα σε έναν ορισμό [function](#) (ή μέθοδος) που καθορίζει ένα [argument](#) (ή σε ορισμένες περιπτώσεις, ορίσματα) που μπορεί να δεχθεί η συνάρτηση. Υπάρχουν πέντε είδη παραμέτρων:

- **λέξη-κλειδί ή θέση:** καθορίζει ένα όρισμα που μπορεί να μεταβιβαστεί είτε *θέσεως* ή ως *όρισμα λέξης-κλειδιού*. Αυτό είναι το προεπιλεγμένο είδος παραμέτρου, για παράδειγμα `foo` και `bar` στα ακόλουθα:

```
def func(foo, bar=None): ...
```

- **θέσεως μόνο:** καθορίζει ένα όρισμα που μπορεί να παρέχεται μόνο από τη θέση. Οι παράμετροι μόνο θέσης μπορούν να οριστούν συμπεριλαμβάνοντας έναν χαρακτήρα / στη λίστα παραμέτρων του ορισμού συνάρτησης μετά από αυτές, για παράδειγμα `posonly1` και `posonly2` στα εξής:

```
def func(posonly1, posonly2, /, positional_or_keyword): ...
```

- **λέξης-κλειδί μόνο:** καθορίζει ένα όρισμα που μπορεί να παρέχεται μόνο με λέξη κλειδί. Οι παράμετροι μόνο για λέξη-κλειδί μπορούν να οριστούν συμπεριλαμβάνοντας μια παράμετρο θέσης ή σκέτο `*` στη λίστα παραμέτρων του ορισμού συνάρτησης πριν από αυτές, για παράδειγμα `kw_only1` και `kw_only2` στα ακόλουθα:

```
def func(arg, *, kw_only1, kw_only2): ...
```

- **μεταβλητή θέσης:** καθορίζει ότι μπορεί να παρασχεθεί μια αυθαίρετη ακολουθία ορισμάτων θέσης (επιπλέον των ορισμάτων θέσης που είναι ήδη αποδεκτά από άλλες παραμέτρους). Μια τέτοια παράμετρος μπορεί να οριστεί προσαρτώντας το όνομα της παραμέτρου με `*`, για παράδειγμα `args` στα ακόλουθα:

```
def func(*args, **kwargs): ...
```

- **μεταβλητή λέξη-κλειδί:** καθορίζει ότι μπορούν να παρέχονται αυθαίρετα πολλά ορίσματα λέξης-κλειδιού (επιπλέον των ορισμάτων λέξης κλειδιού που είναι αποδεκτά από άλλες παραμέτρους). Μια τέτοια παράμετρος μπορεί να οριστεί προσαρτώντας το όνομα της παραμέτρου με `**`, για παράδειγμα `kwargs` όπως παραπάνω.

Οι παράμετροι μπορούν να καθορίσουν τόσο τα προαιρετικά όσο και τα απαιτούμενα ορίσματα, καθώς και προεπιλεγμένες τιμές για ορισμένα προαιρετικά ορίσματα.

Βλ. επίσης την [argument](#) καταχώριση ευρετηρίου, την ερώτηση FAQ σχετικά με τη διαφορά μεταξύ ορισμάτων και παραμέτρων, την κλάση `inspect.Parameter`, την ενότητα `function` και [PEP 362](#).

**path entry** Μια μεμονωμένη τοποθεσία στο `import path` την οποία συμβουλεύεται ο `path based finder` για να βρει modules για εισαγωγή.

**path entry finder** Ένας `finder` που επιστρέφεται από έναν καλούμενο στο `sys.path_hooks` (δηλαδή ένα `path entry hook`) που ξέρει πως να εντοπίζει modules με `path entry`.

Βλ. `importlib.abc.PathEntryFinder` για τις μεθόδους που ο `entry finder` διαδρομής υλοποιεί.

**path entry hook** A callable on the `sys.path_hook` list which returns a `path entry finder` if it knows how to find modules on a specific `path entry`.

**path based finder** Ένα από τα προεπιλεγμένα `meta path finders` που αναζητά ένα `import path` για modules.

**path-like αντικείμενο** Ένα αντικείμενο που αντιπροσωπεύει ένα `path` συστήματος αρχείων. Ένα αντικείμενο `path` είναι είτε ένα αντικείμενο `str` ή `bytes` που αντιπροσωπεύει ένα `path` ή ένα αντικείμενο που υλοποιεί το πρωτόκολλο `os.PathLike`. Ένα αντικείμενο που υποστηρίζει το πρωτόκολλο `os.PathLike` μπορεί να μετατραπεί σε `path` συστήματος αρχείων `str` ή `bytes` καλώντας την συνάρτηση `os.fspath()` τα `os.fsdecode()` και `os.fsencode()` μπορούν να χρησιμοποιηθούν για την εγγύηση ενός αποτελέσματος `str` ή `bytes`, αντίστοιχα. Εισήχθη από τον [PEP 519](#).

**PEP** Πρόταση Βελτίωσης Python. Ένα PEP είναι ένα έγγραφο σχεδιασμού που παρέχει πληροφορίες στην κοινότητα Python ή περιγράφει μια νέα δυνατότητα για την Python ή τις διαδικασίες ή το περιβάλλον της. Τα PEP θα πρέπει να παρέχουν μια συνοπτική τεχνική προδιαγραφή και μια λογική για τα προτεινόμενα χαρακτηριστικά.

Τα PEP προορίζονται να είναι οι κύριοι μηχανισμοί για την πρόταση σημαντικών νέων χαρακτηριστικών, για τη συλλογή πληροφοριών της κοινότητας για ένα ζήτημα και για την τεκμηρίωση των αποφάσεων σχεδιασμού που έχουν εισαχθεί στην Python. Ο συγγραφέας του PEP είναι υπεύθυνος για την οικοδόμηση συναίνεσης εντός της κοινότητας και την τεκμηρίωση αντίθετων απόψεων.

Βλ. [PEP 1](#).

**τμήμα** Ένα σύνολο από αρχεία σε έναν μόνο κατάλογο (ενδεχομένως αποθηκευμένο σε αρχείο `zip`) που συμβάλλουν σε ένα namespace πακέτο, όπως ορίζεται στο [PEP 420](#).

**όρισμα θέσης** Βλ. [argument](#).

**provisional API** Ένα provisional API είναι αυτό που έχει εσκεμμένα εξαιρεθεί από τις backwards εγγυήσεις συμβατότητας της τυπικής βιβλιοθήκης. Αν και δεν αναμένονται σημαντικές αλλαγές σε τέτοιες διαπαφές, εφόσον επισημαίνονται ως προσωρινές, αλλαγές μη backwards συμβατότητας (μέχρι και κατάργηση της διεπαφής) μπορεί να προκύψουν εάν κριθεί απαραίτητο από τους βασικούς προγραμματιστές. Τέτοιες αλλαγές δεν θα γίνουν άσκοπα – θα συμβούν μόνο εάν αποκαλυφθούν σοβαρά θεμελιώδη ελαττώματα που παραλείφθηκαν πριν από τη συμπερίληψη του API.

Ακόμη και για provisional API, οι μη backwards συμβατές αλλαγές θεωρούνται «λύση έσχατης ανάγκης»- θα εξακολουθεί να γίνεται κάθε προσπάθεια για να βρεθεί μια λύση backwards συμβατή σε τυχόν εντοπισμένα προβλήματα.

Αυτή η διαδικασία επιτρέπει στην τυπική βιβλιοθήκη να συνεχίσει να εξελίσσεται με την πάροδο του χρόνου, χωρίς να κλειδώνει προβληματικά σφάλματα σχεδιασμού για εκτεταμένες χρονικές περιόδους. Βλ. [PEP 411](#) για περισσότερες λεπτομέρειες.

**provisional πακέτο** Βλ. [provisional API](#).

**Python 3000** Ψευδώνυμο για το σύνολο εκδόσεων Python 3.x (επινοήθηκε πριν από πολύ καιρό όταν η κυκλοφορία της έκδοσης 3 ήταν κάτι στο μακρινό μέλλον.) Αυτό ονομάζεται επίσης ως συντομογραφία «Py3k».

**Pythonic** Μια ιδέα ή ένα κομμάτι κώδικα που ακολουθεί πιστά τα πιο κοινά ιδιώματα της γλώσσας Python, αντί να υλοποιεί κώδικα χρησιμοποιώντας έννοιες κοινές σε άλλες γλώσσες. Για παράδειγμα, ένα κοινό

ιδίωμα στην Python είναι να κάνει μια επανάληψη πάνω από όλα τα στοιχεία ενός iterable χρησιμοποιώντας μια δήλωση `for`. Πολλές άλλες γλώσσες που δεν έχουν αυτόν τον τύπο κατασκευής, έτσι οι άνθρωποι που δεν είναι εξοικειωμένοι με την Python χρησιμοποιούν μερικές φορές έναν αριθμητικό μετρητή:

```
for i in range(len(food)):
    print(food[i])
```

Αντίθετα, μια πιο καθαρή μέθοδος Pythonic:

```
for piece in food:
    print(piece)
```

**αναγνωρισμένο όνομα** Ένα όνομα με κουκκίδες που δείχνει τη «διαδρομή» από το καθολικό εύρος ενός module σε μια κλάση, συνάρτηση ή μέθοδο που ορίζεται σε αυτήν την ενότητα, όπως ορίζεται στο [PEP 3155](#). Για συναρτήσεις και κλάσεις ανώτατου επιπέδου, το αναγνωρισμένο όνομα είναι ίδιο με το όνομα του αντικειμένου:

```
>>> class C:
...     class D:
...         def meth(self):
...             pass
...
>>> C.__qualname__
'C'
>>> C.D.__qualname__
'C.D'
>>> C.D.meth.__qualname__
'C.D.meth'
```

Όταν χρησιμοποιείται για αναφορά σε modules, το *πλήρως αναγνωρισμένο όνομα* σημαίνει ολόκληρο το διακεκομμένο path προς το module, συμπεριλαμβανομένων τυχόν γονικών πακέτων π.χ. `email.mime.text`:

```
>>> import email.mime.text
>>> email.mime.text.__name__
'email.mime.text'
```

**πλήθος αναφοράς** The number of references to an object. When the reference count of an object drops to zero, it is deallocated. Reference counting is generally not visible to Python code, but it is a key element of the [CPython](#) implementation. The `sys` module defines a `getrefcount()` function that programmers can call to return the reference count for a particular object.

**κανονικό πακέτο** Ένα παραδοσιακό *package*, όπως ένας κατάλογος που περιέχει ένα `__init__.py` αρχείο.

Βλ. επίσης [namespace package](#).

**\_\_slots\_\_** Μια δήλωση μέσα σε μια κλάση που εξοικονομεί μνήμη δηλώνοντας εκ των προτέρων χώρο για παράδειγμα χαρακτηριστικά και εξαλείφοντας λεξικά στιγμιότυπων. Αν και δημοφιλής, η τεχνική είναι κάπως δύσκολο να γίνει σωστή και προορίζεται καλύτερα για σπάνιες περιπτώσεις όπου υπάρχει μεγάλος αριθμός στιγμιότυπων σε μια εφαρμογή κρίσιμης-μνήμης.

**ακολουθία** An *iterable* which supports efficient element access using integer indices via the `__getitem__()` special method and defines a `__len__()` method that returns the length of the sequence. Some built-in sequence types are `list`, `str`, `tuple`, and `bytes`. Note that `dict` also supports `__getitem__()` and `__len__()`, but is considered a mapping rather than a sequence because the lookups use arbitrary *immutable* keys rather than integers.

The `collections.abc.Sequence` abstract base class defines a much richer interface that goes beyond just `__getitem__()` and `__len__()`, adding `count()`, `index()`, `__contains__()`, and `__reversed__()`. Types that implement this expanded interface can be registered explicitly using `register()`.

**set comprehension** Ένας συμπαγής τρόπος για να επεξεργαστείτε όλα ή μέρος των στοιχείων σε ένα iterable και να επιστραφεί ένα σύνολο με τα αποτελέσματα. `results = {c for c in 'abracadabra' if c not in 'abc'}` δημιουργεί το σύνολο συμβολοσειρών {'r', 'd'}. Βλ. [comprehensions](#).

**μοναδικό dispatch** Μια μορφή dispatch [generic function](#) όπου η υλοποίηση επιλέγεται με βάση τον τύπο ενός μεμονωμένου ορίσματος.

**slice** Ένα αντικείμενο που συνήθως περιέχει ένα τμήμα μιας ακολουθίας [sequence](#). Δημιουργείται ένα slice χρησιμοποιώντας τη σημείωση subscript, `[]` με άνω και κάτω τελείες μεταξύ αριθμών όταν δίνονται πολλοί, όπως στο `variable_name[1:3:5]`. Η σημείωση αγκύλης (subscript) χρησιμοποιεί εσωτερικά αντικείμενα slice.

**ειδική μέθοδος** Μια μέθοδος που καλείται σιωπηρά από την Python για να εκτελέσει μια συγκεκριμένη λειτουργία σε έναν τύπο, όπως η προσθήκη. Τέτοιες μέθοδοι έχουν ονόματα που ξεκινούν και τελειώνουν με διπλές κάτω παύλες. Οι ειδικές μέθοδοι τεκμηριώνονται στο `specialnames`.

**δήλωση** Μια πρόταση είναι μέρος μιας σουίτας (ένα «μπλοκ» κώδικα). Μια πρόταση είναι είτε ένας [expression](#) είτε μια από πολλές δομές με μια λέξη-κλειδί όπως `if`, `while` ή `for`.

**strong reference** Στο C API της Python, μια ισχυρή αναφορά είναι μια αναφορά σε ένα αντικείμενο που ανήκει στον κώδικα που περιέχει την αναφορά. Η ισχυρή αναφορά λαμβάνεται καλώντας το `Py_INCREF()` όταν η αναφορά δημιουργείται και απελευθερώνεται με `Py_DECREF()` όταν διαγραφεί η αναφορά.

Η συνάρτηση `Py_NewRef()` μπορεί να χρησιμοποιηθεί για τη δημιουργία ισχυρής αναφοράς σε ένα αντικείμενο. Συνήθως, η συνάρτηση `Py_DECREF()` πρέπει να καλείται στην ισχυρή αναφορά πριν βγει από το εύρος της ισχυρής αναφοράς, για να αποφευχθεί η διαρροή μιας αναφοράς.

Βλ. επίσης [borrowed reference](#).

**κωδικοποίηση κειμένου** Μια συμβολοσειρά στην Python είναι μια ακολουθία σημείων κώδικα Unicode (στο εύρος U+0000–U+10FFFF). Για να αποθηκεύσετε ή να μεταφέρετε μια συμβολοσειρά, πρέπει να σειριοποιηθεί ως δυαδική ακολουθία.

Η σειριοποίηση μιας συμβολοσειράς σε μια δυαδική ακολουθία είναι γνωστή ως «κωδικοποίηση», και η αναδιουργία της συμβολοσειράς από την δυαδική ακολουθία είναι γνωστή ως «αποκωδικοποίηση».

Υπάρχει μια ποικιλία διαφορετικής σειριοποίησης κειμένου codecs, οι οποίοι συλλογικά αναφέρονται ως «κωδικοποιήσεις κειμένου».

**αρχείο κειμένου** Ένα [file object](#) ικανό να διαβάζει και να γράφει αντικείμενα `str`. Συχνά, ένα αρχείο κειμένου αποκτά πραγματικά πρόσβαση σε μια ροή δυαδική ροή δεδομένων και χειρίζεται αυτόματα την [text encoding](#). Παραδείγματα αρχείων κειμένου είναι αρχεία που ανοίγουν σε λειτουργία κειμένου ('r' ή 'w'), `sys.stdin`, `sys.stdout`, και στιγμιότυπα του `io.StringIO`.

Βλ. επίσης [binary file](#) για ένα αντικείμενο αρχείου με δυνατότητα ανάγνωσης και εγγραφής [δυαδικά αντικείμενα](#).

**συμβολοσειρά τριπλών εισαγωγικών** Μια συμβολοσειρά που δεσμεύεται από τρεις περιπτώσεις είτε ενός εισαγωγικού (») ή μιας αποστρόφου ("). Αν και δεν παρέχουν καμία λειτουργικότητα που δεν είναι διαθέσιμη με συμβολοσειρές με μονά εισαγωγικά, είναι χρήσιμες για διαφόρους λόγους. Σας επιτρέπουν να συμπεριλάβετε μονά και διπλά εισαγωγικά χωρίς διαφυγή σε μια συμβολοσειρά και μπορούν να εκτείνονται σε πολλές γραμμές χωρίς τη χρήση του χαρακτήρα συνέχεια, καθιστώντας τα ιδιαίτερα χρήσιμα κατά τη σύνταξη εγγράφων με συμβολοσειρές.

**τύπος** The type of a Python object determines what kind of object it is; every object has a type. An object's type is accessible as its `__class__` attribute or can be retrieved with `type(obj)`.

**type alias** Ένα συνώνυμο για έναν τύπο, που δημιουργείται με την ανάθεση τύπου σε ένα αναγνωριστικό.

Τα type aliases είναι χρήσιμα για την απλοποίηση [type alias](#). Για παράδειγμα:

```
def remove_gray_shades(
    colors: list[tuple[int, int, int]]) -> list[tuple[int, int, int]]:
    pass
```

μπορεί να γίνει πιο ευανάγνωστο όπως:

```
Color = tuple[int, int, int]

def remove_gray_shades(colors: list[Color]) -> list[Color]:
    pass
```

Βλ. `typing` και [PEP 484](#), που περιγράφει αυτήν την λειτουργικότητα.

**type hint** Ένας *annotation* που καθορίζει τον αναμενόμενο τύπο για μια μεταβλητή, ένα χαρακτηριστικό κλάσης ή μια παράμετρο συνάρτησης ή τιμή επιστροφής.

Type hints are optional and are not enforced by Python but they are useful to static type analysis tools, and aid IDEs with code completion and refactoring.

Υποδείξεις τύπου (type hints) για καθολικές μεταβλητές, χαρακτηριστικά κλάσης και συναρτήσεις, αλλά όχι τοπικές μεταβλητές, μπορούν να προσπελαστούν χρησιμοποιώντας το `typing.get_type_hints()`.

Βλ. `typing` και [PEP 484](#), που περιγράφει αυτήν την λειτουργικότητα.

**καθολικές νέες γραμμές** Ένα τρόπος ερμηνείας ροών κειμένου στον οποίο όλα τα ακόλουθα αναγνωρίζονται ως λήξεις μιας γραμμής: η σύμβαση τέλους γραμμής του Unix `'\n'`, η σύμβαση των Windows `'\r\n'`, και την παλιά σύμβαση Macintosh `'\r'`. Βλ. [PEP 278](#) και [PEP 3116](#), καθώς και `bytes.splitlines()` για πρόσθετη χρήση.

**annotation μεταβλητής** Ένας *annotation* μια μεταβλητής ή ενός χαρακτηριστικού κλάσης.

Όταν annotating μια μεταβλητή ή ένα χαρακτηριστικό κλάσης, η ανάθεση είναι προαιρετική:

```
class C:
    field: 'annotation'
```

Τα annotations μεταβλητών χρησιμοποιούνται συνήθως για *type hints*: για παράδειγμα αυτή η μεταβλητή αναμένεται να λάβει τιμές `int`:

```
count: int = 0
```

Η σύνταξη annotation μεταβλητής περιγράφεται στην ενότητα `annassign`.

Βλ. *function annotation*, [PEP 484](#) και [PEP 526](#), που περιγράφουν αυτή τη λειτουργία. Δείτε επίσης `annotations-howto` για βέλτιστες πρακτικές σχετικά με την εργασία με σχολιασμούς.

**virtual environment** Ένα συνεργατικά απομονωμένο περιβάλλον χρόνου εκτέλεσης που επιτρέπει στους χρήστες και τις εφαρμογές της Python να εγκαταστήσουν και να αναβαθμίσουν πακέτα διανομής Python χωρίς να παρεμβαίνουν στη συμπεριφορά άλλων εφαρμογών Python που εκτελούνται στο ίδιο σύστημα.

Βλ. επίσης `venv`.

**virtual machine** Ένας υπολογιστής ορίζεται εξ ολοκλήρου από το λογισμικό. Η εικονική μηχανή της Python εκτελεί το *bytecode* που εκπέμπεται από τον μεταγλωττιστή `bytecode`.

**Zen της Python** Κατάλογος σχεδιαστικών αρχών και φιλοσοφιών που είναι χρήσιμες για την κατανόηση και τη χρήση της γλώσσας. Ο κατάλογος μπορεί να βρεθεί πληκτρολογώντας «`import this`» στην διαδραστική κονσόλα.



## ΠΑΡΑΡΤΗΜΑ Β΄

---

### About these documents

---

These documents are generated from [reStructuredText](#) sources by [Sphinx](#), a document processor specifically written for the Python documentation.

Η ανάπτυξη των εγγράφων και των εργαλείων τους είναι εξ΄ ολοκλήρου εθελοντική προσπάθεια, όπως και η ίδια η Python. Εάν θέλετε να συνεισφέρετε, ρίξτε μια ματιά στη σελίδα [reporting-bugs](#) για πληροφορίες σχετικές με το πως να το κάνετε. Καινούριοι εθελοντές είναι πάντα ευπρόσδεκτοι!

Πολλές ευχαριστίες πηγαίνουν στους:

- Fred L. Drake, Jr., the creator of the original Python documentation toolset and writer of much of the content;
- το [Docutils](#) πρότζεκτ για την δημιουργία των εφαρμογών [reStructuredText](#) και [Docutils](#)·
- Fredrik Lundh για το δικό του Alternative Python Reference πρότζεκτ από το οποίο το Sphinx πήρε πολύ καλές ιδέες.

### B'.1 Contributors to the Python Documentation

Πολλοί άνθρωποι έχουν συνεισφέρει στη γλώσσα Python, την βιβλιοθήκη της Python, και τα έγγραφα της Python. Δείτε [Misc/ACKS](#) στις πηγές διανομής της Python για μια λίστα των συντελεστών.

Μόνο με τη συμβολή και τις συνεισφορές της κοινότητας της Python, η Python έχει τέτοια υπέροχα έγγραφα – Σας ευχαριστούμε!





## Γ'.1 Η ιστορία του λογισμικού

Python was created in the early 1990s by Guido van Rossum at Stichting Mathematisch Centrum (CWI, see <https://www.cwi.nl/>) in the Netherlands as a successor of a language called ABC. Guido remains Python's principal author, although it includes many contributions from others.

In 1995, Guido continued his work on Python at the Corporation for National Research Initiatives (CNRI, see <https://www.cnri.reston.va.us/>) in Reston, Virginia where he released several versions of the software.

In May 2000, Guido and the Python core development team moved to BeOpen.com to form the BeOpen PythonLabs team. In October of the same year, the PythonLabs team moved to Digital Creations (now Zope Corporation; see <https://www.zope.org/>). In 2001, the Python Software Foundation (PSF, see <https://www.python.org/psf/>) was formed, a non-profit organization created specifically to own Python-related Intellectual Property. Zope Corporation is a sponsoring member of the PSF.

All Python releases are Open Source (see <https://opensource.org/> for the Open Source Definition). Historically, most, but not all, Python releases have also been GPL-compatible; the table below summarizes the various releases.

Έκδοση	Προερχόμενη από	Έτος	Ιδιοκτησία	GPL compatible?
0.9.0 έως 1.2	δ/ν	1991-1995	CWI	ναι
1.3 έως 1.5.2	1.2	1995-1999	CNRI	ναι
1.6	1.5.2	2000	CNRI	όχι
2.0	1.6	2000	BeOpen.com	όχι
1.6.1	1.6	2001	CNRI	όχι
2.1	2.0+1.6.1	2001	PSF	όχι
2.0.1	2.0+1.6.1	2001	PSF	ναι
2.1.1	2.1+2.0.1	2001	PSF	ναι
2.1.2	2.1.1	2002	PSF	ναι
2.1.3	2.1.2	2002	PSF	ναι
2.2 και πάνω	2.1.1	2001-σήμερα	PSF	ναι

**Σημείωση:** GPL-compatible doesn't mean that we're distributing Python under the GPL. All Python licenses, unlike the GPL, let you distribute a modified version without making your changes open source. The GPL-compatible licenses make it possible to combine Python with other software that is released under the GPL; the others don't.

Χάρη, στους πολλούς εξωτερικούς εθελοντές που εργάστηκαν κάτω από τις οδηγίες του Guido, αυτές οι εκδόσεις έγιναν εφικτές.

## Γ'.2 Όροι και προϋποθέσεις για την πρόσβαση ή την χρήση της Python με άλλους τρόπους

Python software and documentation are licensed under the *PSF License Agreement*.

Starting with Python 3.8.6, examples, recipes, and other code in the documentation are dual licensed under the PSF License Agreement and the *Zero-Clause BSD license*.

Κάποιο λογισμικό που είναι ενσωματωμένο στην Python είναι υπό διαφορετικές άδειες χρήσης. Οι άδειες παρατίθενται με κώδικα που εμπίπτει σε αυτήν την άδεια. Δείτε *Άδειες και Ευχαριστίες για Ενσωματωμένο Λογισμικό* για μια ελλιπή λίστα αυτών των αδειών.

### Γ'.2.1 PSF LICENSE AGREEMENT FOR PYTHON 3.10.18

1. This LICENSE AGREEMENT is between the Python Software Foundation,  
→ ("PSF"), and  
the Individual or Organization ("Licensee") accessing and otherwise  
→ using Python  
3.10.18 software in source or binary form and its associated  
→ documentation.
2. Subject to the terms and conditions of this License Agreement, PSF  
→ hereby  
grants Licensee a nonexclusive, royalty-free, world-wide license to  
→ reproduce,  
analyze, test, perform and/or display publicly, prepare derivative  
→ works,  
distribute, and otherwise use Python 3.10.18 alone or in any derivative  
version, provided, however, that PSF's License Agreement and PSF's  
→ notice of  
copyright, i.e., "Copyright © 2001-2023 Python Software Foundation; All  
→ Rights  
Reserved" are retained in Python 3.10.18 alone or in any derivative  
→ version  
prepared by Licensee.
3. In the event Licensee prepares a derivative work that is based on or  
incorporates Python 3.10.18 or any part thereof, and wants to make the  
derivative work available to others as provided herein, then Licensee  
→ hereby  
agrees to include in any such work a brief summary of the changes made  
→ to Python  
3.10.18.
4. PSF is making Python 3.10.18 available to Licensee on an "AS IS" basis.  
PSF MAKES NO REPRESENTATIONS OR WARRANTIES, EXPRESS OR IMPLIED. BY WAY  
→ OF  
EXAMPLE, BUT NOT LIMITATION, PSF MAKES NO AND DISCLAIMS ANY  
→ REPRESENTATION OR  
WARRANTY OF MERCHANTABILITY OR FITNESS FOR ANY PARTICULAR PURPOSE OR  
→ THAT THE  
USE OF PYTHON 3.10.18 WILL NOT INFRINGE ANY THIRD PARTY RIGHTS.

5. PSF SHALL NOT BE LIABLE TO LICENSEE OR ANY OTHER USERS OF PYTHON 3.10.18 FOR ANY INCIDENTAL, SPECIAL, OR CONSEQUENTIAL DAMAGES OR LOSS AS A RESULT OF MODIFYING, DISTRIBUTING, OR OTHERWISE USING PYTHON 3.10.18, OR ANY DERIVATIVE THEREOF, EVEN IF ADVISED OF THE POSSIBILITY THEREOF.
6. This License Agreement will automatically terminate upon a material breach of its terms and conditions.
7. Nothing in this License Agreement shall be deemed to create any relationship of agency, partnership, or joint venture between PSF and Licensee. This License Agreement does not grant permission to use PSF trademarks or trade name in a trademark sense to endorse or promote products or services of Licensee, or any third party.
8. By copying, installing or otherwise using Python 3.10.18, Licensee agrees to be bound by the terms and conditions of this License Agreement.

## Γ'.2.2 ΣΥΜΦΩΝΙΑ ΑΔΕΙΑΣ BEOPEN.COM ΓΙΑ PYTHON 2.0

### ΣΥΜΦΩΝΙΑ ΑΔΕΙΑΣ ΑΝΟΙΧΤΟΥ ΚΩΔΙΚΑ BEOPEN PYTHON ΕΚΔΟΣΗ 1

1. This LICENSE AGREEMENT is between BeOpen.com ("BeOpen"), having an office at 160 Saratoga Avenue, Santa Clara, CA 95051, and the Individual or Organization ("Licensee") accessing and otherwise using this software in source or binary form and its associated documentation ("the Software").
2. Subject to the terms and conditions of this BeOpen Python License Agreement, BeOpen hereby grants Licensee a non-exclusive, royalty-free, world-wide license to reproduce, analyze, test, perform and/or display publicly, prepare derivative works, distribute, and otherwise use the Software alone or in any derivative version, provided, however, that the BeOpen Python License is retained in the Software, alone or in any derivative version prepared by Licensee.
3. BeOpen is making the Software available to Licensee on an "AS IS" basis. BEOPEN MAKES NO REPRESENTATIONS OR WARRANTIES, EXPRESS OR IMPLIED. BY WAY OF EXAMPLE, BUT NOT LIMITATION, BEOPEN MAKES NO AND DISCLAIMS ANY REPRESENTATION OR WARRANTY OF MERCHANTABILITY OR FITNESS FOR ANY PARTICULAR PURPOSE OR THAT THE USE OF THE SOFTWARE WILL NOT INFRINGE ANY THIRD PARTY RIGHTS.
4. BEOPEN SHALL NOT BE LIABLE TO LICENSEE OR ANY OTHER USERS OF THE SOFTWARE FOR ANY INCIDENTAL, SPECIAL, OR CONSEQUENTIAL DAMAGES OR LOSS AS A RESULT OF USING, MODIFYING OR DISTRIBUTING THE SOFTWARE, OR ANY DERIVATIVE THEREOF, EVEN IF ADVISED OF THE POSSIBILITY THEREOF.
5. This License Agreement will automatically terminate upon a material breach of its terms and conditions.
6. This License Agreement shall be governed by and interpreted in all respects by the law of the State of California, excluding conflict of law provisions. Nothing in this License Agreement shall be deemed to create any relationship of agency, partnership, or joint venture between BeOpen and Licensee. This License

(συνέχεια στην επόμενη σελίδα)

(συνεχίζεται από την προηγούμενη σελίδα)

Agreement does not grant permission to use BeOpen trademarks or trade names in a trademark sense to endorse or promote products or services of Licensee, or any third party. As an exception, the "BeOpen Python" logos available at <http://www.pythonlabs.com/logos.html> may be used according to the permissions granted on that web page.

7. By copying, installing or otherwise using the software, Licensee agrees to be bound by the terms and conditions of this License Agreement.

## Γ'.2.3 ΣΥΜΦΩΝΙΑ ΑΔΕΙΑΣ CNRI ΓΙΑ PYTHON 1.6.1

1. This LICENSE AGREEMENT is between the Corporation for National Research Initiatives, having an office at 1895 Preston White Drive, Reston, VA 20191 ("CNRI"), and the Individual or Organization ("Licensee") accessing and otherwise using Python 1.6.1 software in source or binary form and its associated documentation.
2. Subject to the terms and conditions of this License Agreement, CNRI hereby grants Licensee a nonexclusive, royalty-free, world-wide license to reproduce, analyze, test, perform and/or display publicly, prepare derivative works, distribute, and otherwise use Python 1.6.1 alone or in any derivative version, provided, however, that CNRI's License Agreement and CNRI's notice of copyright, i.e., "Copyright © 1995-2001 Corporation for National Research Initiatives; All Rights Reserved" are retained in Python 1.6.1 alone or in any derivative version prepared by Licensee. Alternately, in lieu of CNRI's License Agreement, Licensee may substitute the following text (omitting the quotes): "Python 1.6.1 is made available subject to the terms and conditions in CNRI's License Agreement. This Agreement together with Python 1.6.1 may be located on the internet using the following unique, persistent identifier (known as a handle): 1895.22/1013. This Agreement may also be obtained from a proxy server on the internet using the following URL: <http://hdl.handle.net/1895.22/1013>."
3. In the event Licensee prepares a derivative work that is based on or incorporates Python 1.6.1 or any part thereof, and wants to make the derivative work available to others as provided herein, then Licensee hereby agrees to include in any such work a brief summary of the changes made to Python 1.6.1.
4. CNRI is making Python 1.6.1 available to Licensee on an "AS IS" basis. CNRI MAKES NO REPRESENTATIONS OR WARRANTIES, EXPRESS OR IMPLIED. BY WAY OF EXAMPLE, BUT NOT LIMITATION, CNRI MAKES NO AND DISCLAIMS ANY REPRESENTATION OR WARRANTY OF MERCHANTABILITY OR FITNESS FOR ANY PARTICULAR PURPOSE OR THAT THE USE OF PYTHON 1.6.1 WILL NOT INFRINGE ANY THIRD PARTY RIGHTS.
5. CNRI SHALL NOT BE LIABLE TO LICENSEE OR ANY OTHER USERS OF PYTHON 1.6.1 FOR ANY INCIDENTAL, SPECIAL, OR CONSEQUENTIAL DAMAGES OR LOSS AS A RESULT OF MODIFYING, DISTRIBUTING, OR OTHERWISE USING PYTHON 1.6.1, OR ANY DERIVATIVE THEREOF, EVEN IF ADVISED OF THE POSSIBILITY THEREOF.
6. This License Agreement will automatically terminate upon a material breach of its terms and conditions.
7. This License Agreement shall be governed by the federal intellectual property law of the United States, including without limitation the federal copyright law, and, to the extent such U.S. federal law does not apply, by the law of the Commonwealth of Virginia, excluding Virginia's conflict of law provisions. Notwithstanding the foregoing, with regard to derivative works based on Python 1.6.1 that incorporate non-separable material that was previously distributed under the GNU General Public License (GPL), the law of the Commonwealth of Virginia shall govern this License Agreement only as to issues arising under or

(συνέχεια στην επόμενη σελίδα)

(συνεχίζεται από την προηγούμενη σελίδα)

with respect to Paragraphs 4, 5, and 7 of this License Agreement. Nothing in this License Agreement shall be deemed to create any relationship of agency, partnership, or joint venture between CNRI and Licensee. This License Agreement does not grant permission to use CNRI trademarks or trade name in a trademark sense to endorse or promote products or services of Licensee, or any third party.

8. By clicking on the "ACCEPT" button where indicated, or by copying, installing or otherwise using Python 1.6.1, Licensee agrees to be bound by the terms and conditions of this License Agreement.

## Γ'.2.4 ΣΥΜΦΩΝΙΑ ΑΔΕΙΑΣ CWI ΓΙΑ PYTHON 0.9.0 ΕΩΣ 1.2

Copyright © 1991 - 1995, Stichting Mathematisch Centrum Amsterdam, The Netherlands. All rights reserved.

Permission to use, copy, modify, and distribute this software and its documentation for any purpose and without fee is hereby granted, provided that the above copyright notice appear in all copies and that both that copyright notice and this permission notice appear in supporting documentation, and that the name of Stichting Mathematisch Centrum or CWI not be used in advertising or publicity pertaining to distribution of the software without specific, written prior permission.

STICHTING MATHEMATISCH CENTRUM DISCLAIMS ALL WARRANTIES WITH REGARD TO THIS SOFTWARE, INCLUDING ALL IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS, IN NO EVENT SHALL STICHTING MATHEMATISCH CENTRUM BE LIABLE FOR ANY SPECIAL, INDIRECT OR CONSEQUENTIAL DAMAGES OR ANY DAMAGES WHATSOEVER RESULTING FROM LOSS OF USE, DATA OR PROFITS, WHETHER IN AN ACTION OF CONTRACT, NEGLIGENCE OR OTHER TORTIOUS ACTION, ARISING OUT OF OR IN CONNECTION WITH THE USE OR PERFORMANCE OF THIS SOFTWARE.

## Γ'.2.5 ZERO-CLAUSE BSD LICENSE FOR CODE IN THE PYTHON 3.10.18 DOCUMENTATION

Permission to use, copy, modify, and/or distribute this software for any purpose with or without fee is hereby granted.

THE SOFTWARE IS PROVIDED "AS IS" AND THE AUTHOR DISCLAIMS ALL WARRANTIES WITH REGARD TO THIS SOFTWARE INCLUDING ALL IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS. IN NO EVENT SHALL THE AUTHOR BE LIABLE FOR ANY SPECIAL, DIRECT, INDIRECT, OR CONSEQUENTIAL DAMAGES OR ANY DAMAGES WHATSOEVER RESULTING FROM LOSS OF USE, DATA OR PROFITS, WHETHER IN AN ACTION OF CONTRACT, NEGLIGENCE OR OTHER TORTIOUS ACTION, ARISING OUT OF OR IN CONNECTION WITH THE USE OR PERFORMANCE OF THIS SOFTWARE.

## Γ'.3 Άδειες και Ευχαριστίες για Ενσωματωμένο Λογισμικό

Αυτή η ενότητα είναι μια ημιτελής, αλλά αυξανόμενη λίστα αδειών και ευχαριστιών για λογισμικό τρίτων, που ενσωματώνεται στην διανομή της Python.

### Γ'.3.1 Mersenne Twister

The `_random` module includes code based on a download from <http://www.math.sci.hiroshima-u.ac.jp/~m-mat/MT/MT2002/emt19937ar.html>. The following are the verbatim comments from the original code:

```
A C-program for MT19937, with initialization improved 2002/1/26.
Coded by Takuji Nishimura and Makoto Matsumoto.

Before using, initialize the state by using init_genrand(seed)
or init_by_array(init_key, key_length).

Copyright (C) 1997 - 2002, Makoto Matsumoto and Takuji Nishimura,
All rights reserved.

Redistribution and use in source and binary forms, with or without
modification, are permitted provided that the following conditions
are met:

1. Redistributions of source code must retain the above copyright
   notice, this list of conditions and the following disclaimer.

2. Redistributions in binary form must reproduce the above copyright
   notice, this list of conditions and the following disclaimer in the
   documentation and/or other materials provided with the distribution.

3. The names of its contributors may not be used to endorse or promote
   products derived from this software without specific prior written
   permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
"AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR
A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR
CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL,
EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO,
PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR
PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF
LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING
NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS
SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

Any feedback is very welcome.
http://www.math.sci.hiroshima-u.ac.jp/~m-mat/MT/emt.html
email: m-mat @ math.sci.hiroshima-u.ac.jp (remove space)
```

## Γ'.3.2 Sockets

The `socket` module uses the functions, `getaddrinfo()`, and `getnameinfo()`, which are coded in separate source files from the WIDE Project, <https://www.wide.ad.jp/>.

Copyright (C) 1995, 1996, 1997, and 1998 WIDE Project.  
All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
3. Neither the name of the project nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE PROJECT AND CONTRIBUTORS ``AS IS'' AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE PROJECT OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

## Γ'.3.3 Ασύγχρονες socket υπηρεσίες

The `asynchat` and `asyncore` modules contain the following notice:

Copyright 1996 by Sam Rushing

All Rights Reserved

Permission to use, copy, modify, and distribute this software and its documentation for any purpose and without fee is hereby granted, provided that the above copyright notice appear in all copies and that both that copyright notice and this permission notice appear in supporting documentation, and that the name of Sam Rushing not be used in advertising or publicity pertaining to distribution of the software without specific, written prior permission.

SAM RUSHING DISCLAIMS ALL WARRANTIES WITH REGARD TO THIS SOFTWARE, INCLUDING ALL IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS, IN NO EVENT SHALL SAM RUSHING BE LIABLE FOR ANY SPECIAL, INDIRECT OR CONSEQUENTIAL DAMAGES OR ANY DAMAGES WHATSOEVER RESULTING FROM LOSS OF USE, DATA OR PROFITS, WHETHER IN AN ACTION OF CONTRACT, NEGLIGENCE OR OTHER TORTIOUS ACTION, ARISING OUT OF OR IN CONNECTION WITH THE USE OR PERFORMANCE OF THIS SOFTWARE.

### Γ'.3.4 Διαχείριση Cookie

Η ενότητα `http.cookies` περιέχει την παρακάτω ειδοποίηση:

```
Copyright 2000 by Timothy O'Malley <timo@alum.mit.edu>

    All Rights Reserved

Permission to use, copy, modify, and distribute this software
and its documentation for any purpose and without fee is hereby
granted, provided that the above copyright notice appear in all
copies and that both that copyright notice and this permission
notice appear in supporting documentation, and that the name of
Timothy O'Malley not be used in advertising or publicity
pertaining to distribution of the software without specific, written
prior permission.

Timothy O'Malley DISCLAIMS ALL WARRANTIES WITH REGARD TO THIS
SOFTWARE, INCLUDING ALL IMPLIED WARRANTIES OF MERCHANTABILITY
AND FITNESS, IN NO EVENT SHALL Timothy O'Malley BE LIABLE FOR
ANY SPECIAL, INDIRECT OR CONSEQUENTIAL DAMAGES OR ANY DAMAGES
WHATSOEVER RESULTING FROM LOSS OF USE, DATA OR PROFITS,
WHETHER IN AN ACTION OF CONTRACT, NEGLIGENCE OR OTHER TORTIOUS
ACTION, ARISING OUT OF OR IN CONNECTION WITH THE USE OR
PERFORMANCE OF THIS SOFTWARE.
```

### Γ'.3.5 Ανίχνευση εκτέλεσης

Η ενότητα `trace` περιέχει την παρακάτω ειδοποίηση:

```
portions copyright 2001, Autonomous Zones Industries, Inc., all rights...
err... reserved and offered to the public under the terms of the
Python 2.2 license.
Author: Zooko O'Whielacronx
http://zooko.com/
mailto:zooko@zooko.com

Copyright 2000, Mojam Media, Inc., all rights reserved.
Author: Skip Montanaro

Copyright 1999, Bioreason, Inc., all rights reserved.
Author: Andrew Dalke

Copyright 1995-1997, Automatrix, Inc., all rights reserved.
Author: Skip Montanaro

Copyright 1991-1995, Stichting Mathematisch Centrum, all rights reserved.

Permission to use, copy, modify, and distribute this Python software and
its associated documentation for any purpose without fee is hereby
granted, provided that the above copyright notice appears in all copies,
and that both that copyright notice and this permission notice appear in
supporting documentation, and that the name of neither Automatrix,
Bioreason or Mojam Media be used in advertising or publicity pertaining to
distribution of the software without specific, written prior permission.
```



### Γ'.3.6 Συναρτήσεις UUencode και UUdecode

Η ενότητα uu περιέχει την παρακάτω ειδοποίηση:

```
Copyright 1994 by Lance Ellinghouse
Cathedral City, California Republic, United States of America.
    All Rights Reserved
Permission to use, copy, modify, and distribute this software and its
documentation for any purpose and without fee is hereby granted,
provided that the above copyright notice appear in all copies and that
both that copyright notice and this permission notice appear in
supporting documentation, and that the name of Lance Ellinghouse
not be used in advertising or publicity pertaining to distribution
of the software without specific, written prior permission.
LANCE ELLINGHOUSE DISCLAIMS ALL WARRANTIES WITH REGARD TO
THIS SOFTWARE, INCLUDING ALL IMPLIED WARRANTIES OF MERCHANTABILITY AND
FITNESS, IN NO EVENT SHALL LANCE ELLINGHOUSE CENTRUM BE LIABLE
FOR ANY SPECIAL, INDIRECT OR CONSEQUENTIAL DAMAGES OR ANY DAMAGES
WHATSOEVER RESULTING FROM LOSS OF USE, DATA OR PROFITS, WHETHER IN AN
ACTION OF CONTRACT, NEGLIGENCE OR OTHER TORTIOUS ACTION, ARISING OUT
OF OR IN CONNECTION WITH THE USE OR PERFORMANCE OF THIS SOFTWARE.

Modified by Jack Jansen, CWI, July 1995:
- Use binascii module to do the actual line-by-line conversion
  between ascii and binary. This results in a 1000-fold speedup. The C
  version is still 5 times faster, though.
- Arguments more compliant with Python standard
```

### Γ'.3.7 Κλήσεις Απομακρυσμένης Διαδικασίας XML

Η ενότητα xmlrpc.client περιέχει την παρακάτω ειδοποίηση:

```
The XML-RPC client interface is

Copyright (c) 1999-2002 by Secret Labs AB
Copyright (c) 1999-2002 by Fredrik Lundh

By obtaining, using, and/or copying this software and/or its
associated documentation, you agree that you have read, understood,
and will comply with the following terms and conditions:

Permission to use, copy, modify, and distribute this software and
its associated documentation for any purpose and without fee is
hereby granted, provided that the above copyright notice appears in
all copies, and that both that copyright notice and this permission
notice appear in supporting documentation, and that the name of
Secret Labs AB or the author not be used in advertising or publicity
pertaining to distribution of the software without specific, written
prior permission.

SECRET LABS AB AND THE AUTHOR DISCLAIMS ALL WARRANTIES WITH REGARD
TO THIS SOFTWARE, INCLUDING ALL IMPLIED WARRANTIES OF MERCHANT-
ABILITY AND FITNESS. IN NO EVENT SHALL SECRET LABS AB OR THE AUTHOR
BE LIABLE FOR ANY SPECIAL, INDIRECT OR CONSEQUENTIAL DAMAGES OR ANY
DAMAGES WHATSOEVER RESULTING FROM LOSS OF USE, DATA OR PROFITS,
WHETHER IN AN ACTION OF CONTRACT, NEGLIGENCE OR OTHER TORTIOUS
ACTION, ARISING OUT OF OR IN CONNECTION WITH THE USE OR PERFORMANCE
OF THIS SOFTWARE.
```

## Γ'.3.8 test\_epoll

The test\_epoll module contains the following notice:

```
Copyright (c) 2001-2006 Twisted Matrix Laboratories.
```

```
Permission is hereby granted, free of charge, to any person obtaining
a copy of this software and associated documentation files (the
"Software"), to deal in the Software without restriction, including
without limitation the rights to use, copy, modify, merge, publish,
distribute, sublicense, and/or sell copies of the Software, and to
permit persons to whom the Software is furnished to do so, subject to
the following conditions:
```

```
The above copyright notice and this permission notice shall be
included in all copies or substantial portions of the Software.
```

```
THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND,
EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF
MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND
NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE
LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION
OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION
WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
```

## Γ'.3.9 Επιλογή kqueue

Η ενότητα select περιέχει την παρακάτω ειδοποίηση για την kqueue διεπαφή:

```
Copyright (c) 2000 Doug White, 2006 James Knight, 2007 Christian Heimes
All rights reserved.
```

```
Redistribution and use in source and binary forms, with or without
modification, are permitted provided that the following conditions
are met:
```

1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.

```
THIS SOFTWARE IS PROVIDED BY THE AUTHOR AND CONTRIBUTORS ``AS IS'' AND
ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
ARE DISCLAIMED. IN NO EVENT SHALL THE AUTHOR OR CONTRIBUTORS BE LIABLE
FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL
DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS
OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION)
HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT
LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY
OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF
SUCH DAMAGE.
```

### Γ'.3.10 SipHash24

Το αρχείο `Python/pyhash.c` περιέχει την υλοποίηση του Marek Majkowski του αλγορίθμου του Dan Bernstein, SipHash24. Αυτό περιέχει την παρακάτω σημείωση:

```
<MIT License>
Copyright (c) 2013  Marek Majkowski <marek@popcount.org>

Permission is hereby granted, free of charge, to any person obtaining a copy
of this software and associated documentation files (the "Software"), to deal
in the Software without restriction, including without limitation the rights
to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
copies of the Software, and to permit persons to whom the Software is
furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in
all copies or substantial portions of the Software.
</MIT License>

Original location:
  https://github.com/majek/csiphash/

Solution inspired by code from:
  Samuel Neves (supercop/crypto_auth/siphash24/little)
  djb (supercop/crypto_auth/siphash24/little2)
  Jean-Philippe Aumasson (https://131002.net/siphash/siphash24.c)
```

### Γ'.3.11 strtod και dtoa

Το αρχείο `Python/dtoa.c`, που παρέχει τις συναρτήσεις `dtoa` και `strtod` της C για μετατροπή των C doubles προς και από strings, προέρχεται από το ομώνυμο αρχείο του David M. Gay, προς το παρόν διαθέσιμο από <https://web.archive.org/web/20220517033456/http://www.netlib.org/fp/dtoa.c>. Το αρχικό αρχείο, όπως ανακτήθηκε στις 16 Μαρτίου, 2009, περιέχει τα ακόλουθα πνευματικά δικαιώματα και την ειδοποίηση αδειοδότησης:

```
/*****
 *
 * The author of this software is David M. Gay.
 *
 * Copyright (c) 1991, 2000, 2001 by Lucent Technologies.
 *
 * Permission to use, copy, modify, and distribute this software for any
 * purpose without fee is hereby granted, provided that this entire notice
 * is included in all copies of any software which is or includes a copy
 * or modification of this software and in all copies of the supporting
 * documentation for such software.
 *
 * THIS SOFTWARE IS BEING PROVIDED "AS IS", WITHOUT ANY EXPRESS OR IMPLIED
 * WARRANTY. IN PARTICULAR, NEITHER THE AUTHOR NOR LUCENT MAKES ANY
 * REPRESENTATION OR WARRANTY OF ANY KIND CONCERNING THE MERCHANTABILITY
 * OF THIS SOFTWARE OR ITS FITNESS FOR ANY PARTICULAR PURPOSE.
 *****/
```

## Γ'.3.12 OpenSSL

The modules `hashlib`, `posix`, `ssl`, `crypt` use the OpenSSL library for added performance if made available by the operating system. Additionally, the Windows and macOS installers for Python may include a copy of the OpenSSL libraries, so we include a copy of the OpenSSL license here:

### LICENSE ISSUES

=====

The OpenSSL toolkit stays under a dual license, i.e. both the conditions of the OpenSSL License and the original SSLeay license apply to the toolkit. See below for the actual license texts. Actually both licenses are BSD-style Open Source licenses. In case of any license issues related to OpenSSL please contact [openssl-core@openssl.org](mailto:openssl-core@openssl.org).

### OpenSSL License

-----

```
/* =====
 * Copyright (c) 1998-2008 The OpenSSL Project. All rights reserved.
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions
 * are met:
 *
 * 1. Redistributions of source code must retain the above copyright
 * notice, this list of conditions and the following disclaimer.
 *
 * 2. Redistributions in binary form must reproduce the above copyright
 * notice, this list of conditions and the following disclaimer in
 * the documentation and/or other materials provided with the
 * distribution.
 *
 * 3. All advertising materials mentioning features or use of this
 * software must display the following acknowledgment:
 * "This product includes software developed by the OpenSSL Project
 * for use in the OpenSSL Toolkit. (http://www.openssl.org/)"
 *
 * 4. The names "OpenSSL Toolkit" and "OpenSSL Project" must not be used to
 * endorse or promote products derived from this software without
 * prior written permission. For written permission, please contact
 * openssl-core@openssl.org.
 *
 * 5. Products derived from this software may not be called "OpenSSL"
 * nor may "OpenSSL" appear in their names without prior written
 * permission of the OpenSSL Project.
 *
 * 6. Redistributions of any form whatsoever must retain the following
 * acknowledgment:
 * "This product includes software developed by the OpenSSL Project
 * for use in the OpenSSL Toolkit (http://www.openssl.org/)"
 *
 * THIS SOFTWARE IS PROVIDED BY THE OpenSSL PROJECT ``AS IS'' AND ANY
 * EXPRESSED OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
 * IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR
 * PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE OpenSSL PROJECT OR
 * ITS CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
 * SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT
 * NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES;
 * LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION)
 * HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT,
 * STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
```

(συνέχεια στην επόμενη σελίδα)

(συνεχίζεται από την προηγούμενη σελίδα)

```
* ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED
* OF THE POSSIBILITY OF SUCH DAMAGE.
* =====
*
* This product includes cryptographic software written by Eric Young
* (eay@cryptsoft.com). This product includes software written by Tim
* Hudson (tjh@cryptsoft.com).
*
*/
```

Original SSLeay License

-----

```
/* Copyright (C) 1995-1998 Eric Young (eay@cryptsoft.com)
* All rights reserved.
*
* This package is an SSL implementation written
* by Eric Young (eay@cryptsoft.com).
* The implementation was written so as to conform with Netscapes SSL.
*
* This library is free for commercial and non-commercial use as long as
* the following conditions are aheared to. The following conditions
* apply to all code found in this distribution, be it the RC4, RSA,
* lhash, DES, etc., code; not just the SSL code. The SSL documentation
* included with this distribution is covered by the same copyright terms
* except that the holder is Tim Hudson (tjh@cryptsoft.com).
*
* Copyright remains Eric Young's, and as such any Copyright notices in
* the code are not to be removed.
* If this package is used in a product, Eric Young should be given attribution
* as the author of the parts of the library used.
* This can be in the form of a textual message at program startup or
* in documentation (online or textual) provided with the package.
*
* Redistribution and use in source and binary forms, with or without
* modification, are permitted provided that the following conditions
* are met:
* 1. Redistributions of source code must retain the copyright
* notice, this list of conditions and the following disclaimer.
* 2. Redistributions in binary form must reproduce the above copyright
* notice, this list of conditions and the following disclaimer in the
* documentation and/or other materials provided with the distribution.
* 3. All advertising materials mentioning features or use of this software
* must display the following acknowledgement:
* "This product includes cryptographic software written by
* Eric Young (eay@cryptsoft.com)"
* The word 'cryptographic' can be left out if the rouines from the library
* being used are not cryptographic related :-).
* 4. If you include any Windows specific code (or a derivative thereof) from
* the apps directory (application code) you must include an acknowledgement:
* "This product includes software written by Tim Hudson (tjh@cryptsoft.com)"
*
* THIS SOFTWARE IS PROVIDED BY ERIC YOUNG ``AS IS'' AND
* ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
* IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
* ARE DISCLAIMED. IN NO EVENT SHALL THE AUTHOR OR CONTRIBUTORS BE LIABLE
* FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL
* DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS
* OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION)
* HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT
* LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY
```

(συνέχεια στην επόμενη σελίδα)

(συνεχίζεται από την προηγούμενη σελίδα)

```
* OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF
* SUCH DAMAGE.
*
* The licence and distribution terms for any publically available version or
* derivative of this code cannot be changed. i.e. this code cannot simply be
* copied and put under another distribution licence
* [including the GNU Public Licence.]
*/
```

### Γ'.3.13 expat

The `pyexpat` extension is built using an included copy of the `expat` sources unless the build is configured `--with-system-expat`:

```
Copyright (c) 1998, 1999, 2000 Thai Open Source Software Center Ltd
and Clark Cooper

Permission is hereby granted, free of charge, to any person obtaining
a copy of this software and associated documentation files (the
"Software"), to deal in the Software without restriction, including
without limitation the rights to use, copy, modify, merge, publish,
distribute, sublicense, and/or sell copies of the Software, and to
permit persons to whom the Software is furnished to do so, subject to
the following conditions:

The above copyright notice and this permission notice shall be included
in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND,
EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF
MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT.
IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY
CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT,
TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE
SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
```

### Γ'.3.14 libffi

The `_ctypes` extension is built using an included copy of the `libffi` sources unless the build is configured `--with-system-libffi`:

```
Copyright (c) 1996-2008 Red Hat, Inc and others.

Permission is hereby granted, free of charge, to any person obtaining
a copy of this software and associated documentation files (the
``Software''), to deal in the Software without restriction, including
without limitation the rights to use, copy, modify, merge, publish,
distribute, sublicense, and/or sell copies of the Software, and to
permit persons to whom the Software is furnished to do so, subject to
the following conditions:

The above copyright notice and this permission notice shall be included
in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED ``AS IS'', WITHOUT WARRANTY OF ANY KIND,
EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF
MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND
```

(συνέχεια στην επόμενη σελίδα)

(συνεχίζεται από την προηγούμενη σελίδα)

```
NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT
HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY,
WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER
DEALINGS IN THE SOFTWARE.
```

### Γ'.3.15 zlib

Η επέκταση zlib δημιουργείται χρησιμοποιώντας ένα συμπεριλαμβανόμενου αντίγραφο των πηγών zlib, εάν η έκδοση του zlib που βρίσκεται στο σύστημα είναι πολύ παλιά για να χρησιμοποιηθεί για την κατασκευή:

```
Copyright (C) 1995-2011 Jean-loup Gailly and Mark Adler
```

```
This software is provided 'as-is', without any express or implied
warranty. In no event will the authors be held liable for any damages
arising from the use of this software.
```

```
Permission is granted to anyone to use this software for any purpose,
including commercial applications, and to alter it and redistribute it
freely, subject to the following restrictions:
```

1. The origin of this software must not be misrepresented; you must not claim that you wrote the original software. If you use this software in a product, an acknowledgment in the product documentation would be appreciated but is not required.
2. Altered source versions must be plainly marked as such, and must not be misrepresented as being the original software.
3. This notice may not be removed or altered from any source distribution.

```
Jean-loup Gailly
jloup@gzip.org
```

```
Mark Adler
madler@alumni.caltech.edu
```

### Γ'.3.16 cfuhash

Η υλοποίηση του πίνακα κατακερματισμού που χρησιμοποιείται από το `tracemalloc` βασίζεται στο έργο `cfuhash`:

```
Copyright (c) 2005 Don Owens
All rights reserved.
```

```
This code is released under the BSD license:
```

```
Redistribution and use in source and binary forms, with or without
modification, are permitted provided that the following conditions
are met:
```

- \* Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
- \* Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
- \* Neither the name of the author nor the names of its

(συνέχεια στην επόμενη σελίδα)

(συνεχίζεται από την προηγούμενη σελίδα)

```
contributors may be used to endorse or promote products derived
from this software without specific prior written permission.
```

```
THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
"AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS
FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE
COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT,
INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES
(INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR
SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION)
HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT,
STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED
OF THE POSSIBILITY OF SUCH DAMAGE.
```

### Γ'.3.17 libmpdec

The `_decimal` module is built using an included copy of the libmpdec library unless the build is configured `--with-system-libmpdec`:

```
Copyright (c) 2008-2020 Stefan Krah. All rights reserved.
```

```
Redistribution and use in source and binary forms, with or without
modification, are permitted provided that the following conditions
are met:
```

1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.

```
THIS SOFTWARE IS PROVIDED BY THE AUTHOR AND CONTRIBUTORS "AS IS" AND
ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
ARE DISCLAIMED. IN NO EVENT SHALL THE AUTHOR OR CONTRIBUTORS BE LIABLE
FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL
DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS
OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION)
HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT
LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY
OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF
SUCH DAMAGE.
```

### Γ'.3.18 W3C C14N σουίτα δοκιμής

Η σουίτα δοκιμής C14N 2.0 στο πακέτο `test` (`Lib/test/xmltestdata/c14n-20/`) ανακτήθηκε από τον ιστότοπο του W3C <https://www.w3.org/TR/xml-c14n2-testcases/> και διανέμεται με την άδεια 3 ρήτρων BSD:

```
Copyright (c) 2013 W3C(R) (MIT, ERCIM, Keio, Beihang),
All Rights Reserved.
```

```
Redistribution and use in source and binary forms, with or without
modification, are permitted provided that the following conditions
```

(συνέχεια στην επόμενη σελίδα)



(συνεχίζεται από την προηγούμενη σελίδα)

```
are met:
```

- \* Redistributions of works must retain the original copyright notice, this list of conditions and the following disclaimer.
- \* Redistributions in binary form must reproduce the original copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
- \* Neither the name of the W3C nor the names of its contributors may be used to endorse or promote products derived from this work without specific prior written permission.

```
THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
"AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR
A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT
OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT
LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE,
DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY
THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
(INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
```

### Γ'.3.19 Audioop

To module audioop χρησιμοποιεί ως βάση κώδικα του αρχείου g771.c του έργου Sox. <https://sourceforge.net/projects/sox/files/sox/12.17.7/sox-12.17.7.tar.gz>

Αυτό ο πηγαίος κώδικας είναι προϊόν της Sun Microsystems, Inc. και παρέχεται για απεριόριστη χρήση. Οι χρήστες μπορούν να αντιγράψουν ή να τροποποιήσουν αυτόν τον πηγαίο κώδικα χωρίς χρέωση.

Ο ΠΗΓΑΙΟΣ ΚΩΔΙΚΑΣ ΤΟΥ SUN ΠΑΡΕΧΕΤΑΙ ΟΠΩΣ ΕΧΕΙ ΧΩΡΙΣ ΚΑΝΕΝΟΣ ΕΙΔΟΥΣ ΕΓΓΥΗΣΕΙΣ ΣΥΜΠΕΡΙΛΑΜΒΑΝΟΜΕΝΩΝ ΕΓΓΥΗΣΕΩΝ ΣΧΕΔΙΑΣΜΟΥ, ΕΜΠΟΡΕΥΣΙΜΟΤΗΤΑΣ ΚΑΙ ΚΑΤΑΛΛΗΛΟΤΗΤΑΣ ΓΙΑ ΣΥΓΚΕΚΡΙΜΕΝΟ ΣΚΟΠΟ Ή ΠΟΥ ΠΡΟΚΥΠΤΕΙ ΑΠΟ ΚΑΠΟΙΑ ΠΟΡΕΙΑ ΣΥΝΑΛΛΑΓΗΣ, ΧΡΗΣΗΣ Ή ΕΜΠΟΡΙΚΗΣ ΠΡΑΚΤΙΚΗΣ.

Ο πηγαίος κώδικας του Sun παρέχεται χωρίς την υποστήριξη και χωρίς καμία υποχρέωση εκ μέρους της Sun Microsystems, Inc. να βοηθήσει στην χρήση, στη διόρθωση, τροποποίηση ή βελτίωση του.

SUN MICROSYSTEMS, INC. SHALL HAVE NO LIABILITY WITH RESPECT TO THE INFRINGEMENT OF COPYRIGHTS, TRADE SECRETS OR ANY PATENTS BY THIS SOFTWARE OR ANY PART THEREOF.

Σε καμία περίπτωση η Sun Microsystems, Inc. δεν φέρει ευθύνη για τυχόν απώλεια εσόδων ή κερδών ή άλλες ειδικές, έμμεσες και επακόλουθες ζημιές, ακόμη και αν η Sun έχει ενημερωθεί για την πιθανότητα τέτοιων ζημιών.

Sun Microsystems, Inc. 2550 Garcia Avenue Mountain View, Καλιφόρνια 94043



---

### Copyright

---

Η Python και αυτή η τεκμηρίωση είναι:

Copyright © 2001-2023 Python Software Foundation. Όλα τα δικαιώματα διατηρούνται.

Copyright © 2000 BeOpen.com. Όλα τα δικαιώματα διατηρούνται.

Copyright © 1995-2000 Corporation for National Research Initiatives. Όλα τα δικαιώματα διατηρούνται.

Copyright © 1991-1995 Stichting Mathematisch Centrum. Όλα τα δικαιώματα διατηρούνται.

---

Ανατρέξτε στο *[Ιστορία και Άδεια](#)* για πλήρης πληροφόρηση σχετικά με την άδεια χρήσης και τις εξουσιοδοτήσεις.



## μη-αλφαβητικά

- ..., [121](#)
- # (*hash*)
  - comment, [9](#)
- \* (*asterisk*)
  - in function calls, [32](#)
- \*\*
  - in function calls, [33](#)
- 2to3, [121](#)
- : (*colon*)
  - function annotations, [34](#)
- >
  - function annotations, [34](#)
- >>>, [121](#)
- BDFL, [123](#)
- CPython, [124](#)
- C-contiguous, [124](#)
- EAFP, [125](#)
- Fortran contiguous, [124](#)
- GIL, [127](#)
- IDLE, [128](#)
- LBYL, [129](#)
- MRO, [130](#)
- PATH, [52](#), [117](#)
- PEP, [132](#)
- PYTHONPATH, [52](#), [53](#)
- PYTHONSTARTUP, [118](#)
- Python 3000, [132](#)
- Python Enhancement Proposals
  - PEP 1, [132](#)
  - PEP 8, [34](#)
  - PEP 238, [126](#)
  - PEP 278, [135](#)
  - PEP 302, [126](#), [129](#)
  - PEP 343, [124](#)
  - PEP 362, [122](#), [132](#)
  - PEP 411, [132](#)
  - PEP 420, [126](#), [131](#), [132](#)
  - PEP 443, [127](#)
  - PEP 451, [126](#)
  - PEP 483, [127](#)
  - PEP 484, [34](#), [121](#), [126](#), [127](#), [135](#)
  - PEP 492, [122](#), [124](#)
  - PEP 498, [125](#)
  - PEP 519, [132](#)
  - PEP 525, [122](#)
  - PEP 526, [121](#), [135](#)
  - PEP 585, [127](#)
  - PEP 636, [25](#)
  - PEP 3107, [34](#)
  - PEP 3116, [135](#)
  - PEP 3147, [52](#)
  - PEP 3155, [133](#)
- Pythonic, [132](#)
- RFC
  - RFC 2822, [94](#)
- Zen της Python, [135](#)
- \_\_all\_\_, [56](#)
- \_\_future\_\_, [126](#)
- \_\_slots\_\_, [133](#)
- annotation, [121](#)
- annotation μεταβλητής, [135](#)
- annotations
  - function, [34](#)
- awaitable, [122](#)
- builtins
  - μονάδα, [54](#)
- bytecode, [123](#)
- bytes-like αντικείμενα, [123](#)
- callable, [123](#)
- callback, [123](#)
- coding
  - style, [34](#)
- coercion, [123](#)
- context μεταβλητή, [124](#)
- contiguous, [124](#)
- coroutine, [124](#)
- coroutine συνάρτηση, [124](#)
- decorator, [124](#)
- descriptor, [124](#)
- docstring, [125](#)
- docstrings, [25](#), [33](#)
- documentation strings, [25](#), [33](#)
- duck-typing, [125](#)
- f-string, [125](#)
- file
  - αντικείμενο, [63](#)

- finder, [126](#)
- for
  - δήλωση, [20](#)
- function
  - annotations, [34](#)
- generator, [126](#)
- generator expression, [127](#)
- generator iterator, [127](#)
- generator έκφραση, [127](#)
- global interpreter lock, [127](#)
- hash-based pyc, [127](#)
- hashable, [127](#)
- help
  - ενσωματωμένη συνάρτηση, [89](#)
- immutable, [128](#)
- interpreted, [128](#)
- iterable, [128](#)
- iterator, [128](#)
- json
  - μονάδα, [66](#)
- lambda, [129](#)
- list comprehension, [129](#)
- loader, [129](#)
- magic
  - method, [129](#)
- mangling
  - name, [85](#)
- mapping, [129](#)
- meta path finder, [130](#)
- method
  - magic, [129](#)
  - special, [134](#)
  - αντικείμενο, [80](#)
- module, [130](#)
  - search path, [52](#)
- module επέκτασης, [125](#)
- mutable, [130](#)
- name
  - mangling, [85](#)
- named tuple, [130](#)
- namespace, [130](#)
- nested scope, [131](#)
- open
  - ενσωματωμένη συνάρτηση, [63](#)
- path
  - module search, [52](#)
- path based finder, [132](#)
- path entry, [132](#)
- path entry finder, [132](#)
- path entry hook, [132](#)
- path-like αντικείμενο, [132](#)
- provisional API, [132](#)
- provisional πακέτο, [132](#)
- search
  - path, module, [52](#)
- set comprehension, [134](#)
- slice, [134](#)
- special

- method, [134](#)
- strings, documentation, [25](#), [33](#)
- strong reference, [134](#)
- style
  - coding, [34](#)
- sys
  - μονάδα, [53](#)
- type alias, [134](#)
- type hint, [135](#)
- virtual environment, [135](#)
- virtual machine, [135](#)

## A

- ακέραια διαίρεση, [126](#)
- ακολουθία, [133](#)
- αναγνωρισμένο όνομα, [133](#)
- αντικείμενο, [131](#)
  - file, [63](#)
  - method, [80](#)
- αντικείμενο αρχείου, [125](#)
- αντικείμενο που μοιάζει με αρχείο, [125](#)
- αρχείο κειμένου, [134](#)
- ασύγχρονος generator, [122](#)
- ασύγχρονος generator iterator, [122](#)
- ασύγχρονος iterable, [122](#)
- ασύγχρονος iterator, [122](#)
- ασύγχρονος διαχειριστής context, [122](#)
- αφηρημένη βασική κλάση, [121](#)

## Γ

- γενική συνάρτηση, [127](#)
- γενικός τύπος, [127](#)

## Δ

- δανεική αναφορά, [123](#)
- δήλωση, [134](#)
  - for, [20](#)
- διαδραστικός, [128](#)
- διαχειριστής context, [124](#)
- δυαδικό αρχείο, [123](#)

## Ε

- ειδική μέθοδος, [134](#)
- εισαγόμενο path, [128](#)
- εισαγωγέας, [128](#)
- εισαγωγή, [128](#)
- έκφραση, [125](#)
- ενσωματωμένη συνάρτηση
  - help, [89](#)
  - open, [63](#)

## Κ

- καθολικές νέες γραμμές, [135](#)
- κανονικό πακέτο, [133](#)
- κατανόηση λεξικού, [125](#)
- κλάση, [123](#)
- κλάση νέου στυλ, [131](#)
- κωδικοποίηση κειμένου, [134](#)

κωδικοποίηση συστήματος αρχείων και  
χειριστής σφαλμάτων, [125](#)

## Λ

λεξικό, [125](#)  
λίστα, [129](#)

## Μ

μαγική μέθοδος, [129](#)  
μέθοδος, [130](#)  
μετα-κλάση, [130](#)  
μεταβλητή κλάσης, [123](#)  
μεταβλητή περιβάλλοντος  
PATH, [52](#), [117](#)  
PYTHONPATH, [52](#), [53](#)  
PYTHONSTARTUP, [118](#)  
μιγαδικός αριθμός, [124](#)  
μονάδα  
builtins, [54](#)  
json, [66](#)  
sys, [53](#)  
μοναδικό dispatch, [134](#)

## Ο

όρισμα, [121](#)  
όρισμα keyword, [129](#)  
όρισμα θέσης, [132](#)  
όψη λεξικού, [125](#)

## Π

πακέτο, [131](#)  
πακέτο namespace, [131](#)  
παράμετρος, [131](#)  
πλήθος αναφοράς, [133](#)

## Σ

σειρά ανάλυσης μεθόδων, [130](#)  
συλλογή απορριμάτων, [126](#)  
συμβολοσειρά τριπλών εισαγωγικών, [134](#)  
συνάρτηση, [126](#)  
συνάρτηση annotation, [126](#)  
συνάρτηση key, [129](#)

## Τ

τερματισμός λειτουργίας διερμηνέα, [128](#)  
τεχνικές προδιαγραφές module, [130](#)  
τμήμα, [132](#)  
τοπική κωδικοποίηση, [129](#)  
τύπος, [134](#)

## Χ

χαρακτηριστικό, [122](#)