
Distributing Python Modules

Release 3.7.17

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

junho 28, 2023

Python Software Foundation
Email: docs@python.org

1	Termos chave	3
2	Licenciamento de código aberto e colaboração	5
3	Instalando as ferramentas	7
4	Lendo o “Python Packaging User Guide”	9
5	Como eu...?	11
5.1	... escolho um nome para meu projeto?	11
5.2	... crio e distribuo extensões binárias?	11
A	Glossário	13
B	Sobre a Documentação	27
B.1	Contribuidores da Documentação do Python	27
C	História e Licença	29
C.1	História do software	29
C.2	Termos e condições para acessar ou usar Python	30
C.3	Licenças e Reconhecimentos para Software Incorporado	33
D	Direitos Autorais	47
	Índice	49

Email distutils-sig@python.org

Como um projeto popular de desenvolvimento open source, Python tem uma comunidade de apoio ativa de colaboradores e usuários, que também fazem o seu software disponível para outros desenvolvedores de Python para usar sob os termos da licença de código aberto.

Isso permite aos usuários Python compartilhar e colaborar efetivamente, se beneficiando das soluções que outros já tenham criado para os problemas mais comuns (em alguns casos até mesmo os raros), bem como potencialmente contribuindo com suas próprias soluções para o conjunto de soluções comuns.

Este guia cobre a parte de distribuição do processo. Para um guia de instalação de outros projetos Python, consulte [installation guide](#).

Nota: Para corporações e outros usuários institucionais, esteja ciente que muitas organizações têm suas próprias políticas em relação ao uso e contribuição para o software de código aberto. Por favor, leve em conta essas políticas ao usar as ferramentas de distribuição e instalação fornecidas com o Python.

CAPÍTULO 1

Termos chave

- [Python Packaging Index](#) é um repositório público de pacotes licenciados como código aberto e disponíveis para uso de outros usuários Python
- o [Python Packaging Authority](#) é o grupo de desenvolvedores e autores de documentação responsáveis pela manutenção e evolução das ferramentas de empacotamento padrão e pelos padrões de metadados e formato de arquivo associados. Eles mantêm uma variedade de ferramentas, documentação e rastreadores de problemas no [GitHub](#) e [Bitbucket](#).
- `distutils` é o primeiro sistema de construção e distribuição adicionado na biblioteca padrão do Python, em 1998. Embora o uso direto do `distutils` esteja sendo eliminado, ele estabelece as bases para a infraestrutura atual de empacotamento e distribuição, e não somente permanece como parte da biblioteca padrão, mas seu nome continua sendo usado de outras maneiras (como o nome da lista de e-mail usada para coordenar o desenvolvimento dos padrões de empacotamento Python).
- [setuptools](#) é um substituto (em grande parte) para `distutils`, publicado pela primeira vez em 2004. Sua adição mais notável sobre as ferramentas `distutils` originais foi a capacidade de declarar dependências em outros pacotes. Atualmente é recomendado como uma alternativa, mais regularmente atualizada, para `distutils`, que oferece suporte consistente para padrões de empacotamento mais recentes, em uma ampla gama de versões do Python.
- [wheel](#) (nesse contexto) é um projeto que adiciona o comando `bdist_wheel` ao `distutils/setuptools`. Isso produz um formato de pacote binário, portátil entre várias plataformas (chamado “wheels” ou “wheel files” e definido no [PEP 427](#)), permitindo que bibliotecas Python, mesmo aquelas incluindo extensões binárias, sejam instaladas em um sistema, sem precisarem ser construídas localmente.

Licenciamento de código aberto e colaboração

Na maior parte do mundo, o software é automaticamente coberto por direitos autorais. Isso significa que outros desenvolvedores precisam de permissão explícita para copiar, usar, modificar e redistribuir o software.

O licenciamento de código aberto é uma forma de conceder explicitamente tal permissão, de maneira relativamente consistente, permitindo aos desenvolvedores compartilhar e colaborar de forma eficiente, criando soluções comuns para vários problemas disponíveis gratuitamente. Isso permite que os desenvolvedores dediquem mais tempo focados nos problemas que são relativamente exclusivos para sua situação específica.

As ferramentas de distribuição fornecidas com o Python são projetadas para tornar razoavelmente simples, para os desenvolvedores, fazerem suas próprias contribuições, de volta para esse conjunto comum de software, se optarem por fazê-lo.

As mesmas ferramentas de distribuição também podem ser usadas para distribuir software dentro de uma organização, independentemente de esse software ser publicado como software de código aberto ou não.

Instalando as ferramentas

A biblioteca padrão não inclui ferramentas de construção que suportem padrões de empacotamento modernos do Python, pois a equipe de desenvolvimento principal achou que é importante ter ferramentas padrão que funcionem consistentemente, mesmo em versões mais antigas do Python.

As ferramentas de construção e distribuição, recomendadas atualmente, podem ser instaladas chamando o módulo `pip` na linha de comando:

```
python -m pip install setuptools wheel twine
```

Nota: Para usuários POSIX (incluindo usuários Mac OS X e Linux), essas instruções assumem o uso de um *virtual environment*.

Para usuários Windows, essas instruções assumem que a opção de ajustar a variável de ambiente do sistema, `PATH`, foi selecionado durante a instalação do Python.

O “Python Packaging User Guide” inclui mais detalhes sobre [currently recommended tools](#).

Lendo o “Python Packaging User Guide”

O “Python Packaging User Guide” abrange as várias etapas e elementos-chave envolvidos na criação e publicação de um projeto:

- [Project structure](#)
- [Building and packaging the project](#)
- [Uploading the project to the Python Packaging Index](#)

Essas são respostas rápidas ou links para algumas tarefas comuns.

5.1 ... escolho um nome para meu projeto?

Não é um tema fácil, mas aqui estão algumas dicas:

- verifique o “Python Packaging Index” para ver se o nome já está em uso
- verifique sites de hospedagem populares como GitHub, Bitbucket etc. para ver se já existe um projeto com esse nome
- verifique o que surge, em uma pesquisa na Web, para o nome que você está considerando
- evite palavras particularmente comuns, especialmente aquelas com vários significados, pois elas podem dificultar que os usuários encontrem o seu software ao procurá-lo

5.2 ... crio e distribuo extensões binárias?

Este é realmente um tema bastante complexo, com uma variedade de alternativas disponíveis, dependendo exatamente do que você pretende alcançar. Veja “Python Packaging User Guide” para mais informação e recomendações.

Ver também:

[Python Packaging User Guide: Binary Extensions](#)

>>> O prompt padrão do shell interativo do Python. Normalmente visto em exemplos de código que podem ser executados interativamente no interpretador.

... O prompt padrão do shell interativo do Python ao inserir o código para um bloco de código recuado, quando dentro de um par de delimitadores correspondentes esquerdo e direito (parênteses, colchetes, chaves ou aspas triplas) ou após especificar um decorador.

2to3 Uma ferramenta que tenta converter código Python 2.x em código Python 3.x tratando a maioria das incompatibilidades que podem se detectadas com análise do código-fonte e navegação na árvore sintática.

O 2to3 está disponível na biblioteca padrão como `lib2to3`; um ponto de entrada é disponibilizado como `Tools/scripts/2to3`. Veja `2to3-reference`.

classe base abstrata Classes básicas abstratas complementam tipagem `pato`, fornecendo uma maneira de definir interfaces quando outras técnicas, como `hasattr()`, seriam desajeitadas ou sutilmente erradas (por exemplo, com métodos mágicos). ABCs introduzem subclasses virtuais, que são classes que não herdam de uma classe mas ainda são reconhecidas por `isinstance()` e `issubclass()`; veja a documentação do módulo `abc`. O Python vem com muitas ABCs internas para estruturas de dados (no módulo `collections.abc`), números (no módulo `numbers`), fluxos (no módulo `io`), localizadores e carregadores de importação (no módulo `importlib.abc`). Você pode criar suas próprias ABCs com o módulo: `mod:abc`.

Anotação Um rótulo associado a uma variável, um atributo de classe ou um parâmetro de função ou valor de retorno, usado por convenção como: *term: type hint*.

Anotações de variáveis locais não podem ser acessadas em tempo de execução, mas anotações de variáveis globais, atributos de classe e funções são armazenadas no atributo especial: `attr: __annotations__` de módulos, classes e funções, respectivamente.

Ver *variable annotation*, *function annotation*, **PEP 484** e **PEP 526**, que descrevem esta funcionalidade

argumento Um valor passado para um *function* (ou *method*) ao chamar a função. Existem dois tipos de argumento:

- *argumento nomeado*: um argumento precedido por um identificador (por exemplo, `nome=`) na chamada de uma função ou passada como um valor em um dicionário precedido por `**`. Por exemplo, 3 e 5 são ambos argumentos nomeados na chamada da função `complex()` a seguir:

```
complex(real=3, imag=5)
complex(**{'real': 3, 'imag': 5})
```

- *argumento posicional*: um argumento que não é um argumento nomeado. Argumentos posicionais podem aparecer no início da lista de argumentos e/ou podem ser passados com elementos de um *iterável* precedido por *. Por exemplo, 3 e 5 são ambos argumentos posicionais nas chamadas a seguir:

```
complex(3, 5)
complex(*(3, 5))
```

Argumentos são atribuídos às variáveis locais nomeadas no corpo da função. Veja a seção *calls* para as regras de atribuição. Sintaticamente, qualquer expressão pode ser usada para representar um argumento; avaliada a expressão, o valor é atribuído à variável local.

Veja também o termo *parâmetro* no glossário, a pergunta *the difference between arguments and parameters* na FAQ, e [PEP 362](#).

gerenciador de contexto assíncrono An object which controls the environment seen in an `async with` statement by defining `__aenter__()` and `__aexit__()` methods. Introduced by [PEP 492](#).

gerador assíncrono A function which returns an *asynchronous generator iterator*. It looks like a coroutine function defined with `async def` except that it contains `yield` expressions for producing a series of values usable in an `async for` loop.

Normalmente se refere a uma função geradora assíncrona, mas pode se referir a um iterador gerador assíncrono em alguns contextos. Em casos em que o significado não esteja claro, usar o termo completo evita a ambiguidade.

Uma função geradora assíncrona pode conter expressões `await` e também `async for` e `async with`.

gerador iterador assíncrono Um objeto criado por uma função *asynchronous generator*.

Este é um *iterador assíncrono* que, quando chamado usando o método `__anext__()`, retorna um objeto aguardável que executará o corpo da função de gerador assíncrono até a próxima expressão `yield`.

Cada `yield` suspende temporariamente o processamento, lembrando o estado de execução do local (incluindo variáveis locais e instruções de tentativa pendentes). Quando o *iterador do gerador assíncrono* efetivamente é retomado com outro retorno esperado por `__anext__()`, ele inicia de onde parou. Veja [PEP 492](#) e [PEP 525](#).

iterável assíncrono Um objeto que pode ser usado em uma instrução `async for`. Deve retornar um *iterador assíncrono* do seu método `__aiter__()`. Introduzido por [PEP 492](#).

iterador assíncrono Um objeto que implementa os métodos `__aiter__()` e `__anext__()`. `__anext__` deve retornar um objeto *aguardável*. `async for` resolve os aguardáveis retornados por um método `__anext__()` do iterador assíncrono até que ele levante uma exceção `StopAsyncIteration`. Introduzido pela [PEP 492](#).

atributo Um valor associado a um objeto que é referenciado pelo nome separado por um ponto. Por exemplo, se um objeto *o* tem um atributo *a* esse seria referenciado como *o.a*.

aguardável Um objeto que pode ser usado em uma expressão `await`. Pode ser uma *coroutine* ou um objeto com um método `__await__()`. Veja também a [PEP 492](#).

BDFL Abreviação da expressão da língua inglesa “Benevolent Dictator for Life” (em português, “Ditador Benevolente Vitalício”), referindo-se a [Guido van Rossum](#), criador do Python.

arquivo binário Um *objeto arquivo* capaz de ler e gravar em *objetos byte ou similar*. Exemplos de arquivos binários são arquivos abertos no modo binário ('rb', 'wb' ou 'rb+'), `sys.stdin.buffer`, `sys.stdout.buffer` e instâncias de `io.BytesIO` e `gzip.GzipFile`.

Veja também *arquivo texto* para um arquivo objeto capaz de ler e gravar em objetos `str`.

objeto byte ou similar Um objeto que suporta o `bufferobjects` e pode exportar um `buffer C contíguo`. Isso inclui todos os objetos `bytes`, `bytearray` e `array.array`, além de muitos objetos comuns `memoryview`. Objetos `byte`

ou similar podem ser usados para várias operações que funcionam com dados binários; isso inclui compactação, salvamento em um arquivo binário e envio por um soquete.

Algumas operações precisam que os dados binários sejam mutáveis. A documentação geralmente se refere a eles como “objetos `byte` ou similar para leitura-escrita”. Exemplos de objetos de buffer mutável incluem `bytearray` e um `memoryview` de um `bytearray`. Outras operações exigem que os dados binários sejam armazenados em objetos imutáveis (“objetos `byte` ou similar para somente leitura”); exemplos disso incluem `bytes` e a `memoryview` de um objeto `bytes`.

bytecode Python source code is compiled into bytecode, the internal representation of a Python program in the CPython interpreter. The bytecode is also cached in `.pyc` files so that executing the same file is faster the second time (recompilation from source to bytecode can be avoided). This “intermediate language” is said to run on a *virtual machine* that executes the machine code corresponding to each bytecode. Do note that bytecodes are not expected to work between different Python virtual machines, nor to be stable between Python releases.

A list of bytecode instructions can be found in the documentation for the `dis` module.

Classe A template for creating user-defined objects. Class definitions normally contain method definitions which operate on instances of the class.

variável de classe Uma variável definida em uma classe e destinada a ser modificada apenas no nível da classe (ou seja, não em uma instância da classe).

Coerção A conversão implícita de uma instância de um tipo para outro durante uma operação que envolve dois argumentos do mesmo tipo. Por exemplo, `int(3.15)` converte o número do ponto flutuante no número inteiro 3, mas em `3+4.5`, cada argumento é de um tipo diferente (um `int`, um `float`), e ambos devem ser convertidos para o mesmo tipo antes de poderem ser adicionados ou isso levantará um `TypeError`. Sem coerção, todos os argumentos de tipos compatíveis teriam que ser normalizados com o mesmo valor pelo programador, por exemplo, `float(3)+4.5` em vez de apenas `3+4.5`.

número complexo An extension of the familiar real number system in which all numbers are expressed as a sum of a real part and an imaginary part. Imaginary numbers are real multiples of the imaginary unit (the square root of -1), often written `i` in mathematics or `j` in engineering. Python has built-in support for complex numbers, which are written with this latter notation; the imaginary part is written with a `j` suffix, e.g., `3+1j`. To get access to complex equivalents of the `math` module, use `cmath`. Use of complex numbers is a fairly advanced mathematical feature. If you’re not aware of a need for them, it’s almost certain you can safely ignore them.

gerenciador de contexto An object which controls the environment seen in a `with` statement by defining `__enter__()` and `__exit__()` methods. See [PEP 343](#).

variável de contexto Uma variável que pode ter valores diferentes, dependendo do seu contexto. Isso é semelhante ao armazenamento local do encadeamento, no qual cada encadeamento de execução pode ter um valor diferente para uma variável. No entanto, com variáveis de contexto, pode haver vários contextos em um encadeamento de execução e o principal uso para variáveis de contexto é acompanhar as variáveis em tarefas assíncronas simultâneas. Veja `contextvars`.

contíguo Um buffer é considerado contíguo exatamente se for **contíguo C ** ou **contíguo Fortran **. Os buffers de dimensão zero são contíguos C e Fortran. Em matrizes unidimensionais, os itens devem ser dispostos na memória próximos um do outro, em ordem crescente de índices, começando do zero. Em matrizes multidimensionais contíguas C, o último índice varia mais rapidamente ao visitar itens em ordem de endereço de memória. No entanto, nas matrizes contíguas do Fortran, o primeiro índice varia mais rapidamente.

co-rotina Coroutines são uma forma mais generalizada de sub-rotinas. Sub-rotinas tem a entrada iniciada em um ponto, e a saída em outro ponto. Coroutines podem entrar, sair, e continuar em muitos pontos diferentes. Elas podem ser implementadas com a instrução `async def`. Veja também [PEP 492](#).

função de co-rotina Uma função que retorna um objeto do tipo *coroutine*. Uma função coroutine pode ser definida com a instrução `async def`, e pode conter as palavras chaves `await`, `async for`, e `async with`. Isso foi introduzido pela [PEP 492](#).

CPython The canonical implementation of the Python programming language, as distributed on python.org. The term “CPython” is used when necessary to distinguish this implementation from others such as Jython or IronPython.

decorador Uma função que retorna outra função, geralmente aplicada como uma transformação de função usando a sintaxe `@wrapper`. Exemplos comuns para decoradores são `classmethod()` e `staticmethod()`.

A sintaxe do decorador é meramente um açúcar-sintático, as duas definições de funções a seguir são semanticamente equivalentes:

```
def f(...):  
    ...  
f = staticmethod(f)  
  
@staticmethod  
def f(...):  
    ...
```

O mesmo conceito existe para as classes, mas não é comumente utilizado. Veja a documentação de `function definitions` e `class definitions` para obter mais informações sobre decoradores.

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.

Para obter mais informações sobre os métodos dos descritores, veja: `descriptors`.

dicionário Um Array associativo em que chaves arbitrárias são mapeadas para valores. As chaves podem ser quaisquer objetos que possuam os métodos `__hash__()` e `__eq__()`. Dicionários são estruturas chamadas de hash na linguagem Perl.

visualização de dicionário Os objetos retornados por `dict.keys()`, `dict.values()` e `dict.items()` são chamados de Views de Dicionário. Eles fornecem uma visualização dinâmica das entradas do dicionário, o que significa que quando o dicionário é alterado, a View reflete essas alterações. Para forçar a View do dicionário a se tornar uma lista completa use `list(dictview)`. Veja: *ref:dict-views*.

docstring Uma string literal que aparece como primeira expressão numa classe, função ou módulo. Ainda que sejam ignoradas quando a suíte é executada, é reconhecida pelo compilador que a coloca no atributo `__doc__` da classe, função ou módulo que a encapsula. Como ficam disponíveis por meio de introspecção, docstrings são o lugar canônico para documentação do objeto.

duck-typing (tipagem pato) A programming style which does not look at an object’s type to determine if it has the right interface; instead, the method or attribute is simply called or used (“If it looks like a duck and quacks like a duck, it must be a duck.”) By emphasizing interfaces rather than specific types, well-designed code improves its flexibility by allowing polymorphic substitution. Duck-typing avoids tests using `type()` or `isinstance()`. (Note, however, that duck-typing can be complemented with *abstract base classes*.) Instead, it typically employs `hasattr()` tests or *EAFP* programming.

EAFP Easier to ask for forgiveness than permission. This common Python coding style assumes the existence of valid keys or attributes and catches exceptions if the assumption proves false. This clean and fast style is characterized by the presence of many `try` and `except` statements. The technique contrasts with the *LYBL* style common to many other languages such as C.

expressão Uma parte da sintaxe que pode ser avaliada para algum valor. Em outras palavras, uma expressão é a acumulação de elementos de expressão como literais, nomes, atributos de acesso, operadores ou chamadas de funções, todos os quais retornam um valor. Em contraste com muitas outras linguagens, nem todas as construções de linguagem são expressões. Também existem *statement*, os quais não podem ser usadas como expressões, como por exemplo `while`. Atribuições também são instruções, não expressões.

módulo de extensão Um módulo escrito em C ou C++, usando a API C de Python para interagir tanto com código de usuário quanto do núcleo.

f-string Literais string prefixadas com 'f' ou 'F' são conhecidas como “f-strings” que é uma abreviação de formatted string literals. Veja também [PEP 498](#).

file object (arquivo objeto) Um objeto que expõe uma API orientada a arquivos (com métodos tais como `read()` ou `write()`) para um recurso subjacente. Dependendo da maneira como foi criado, um objeto arquivo pode mediar o acesso a um arquivo real no disco ou outro tipo de dispositivo de armazenamento ou de comunicação (por exemplo a entrada/saída padrão, buffers em memória, sockets, pipes, etc.). Objetos arquivo também são chamados de *file-like objects* ou *streams*.

Atualmente há três categorias de objetos arquivo: arquivos binários raw, .. XXX: sugestões para “raw” e “bufferizados”? arquivos binários bufferizados e arquivos texto. Suas interfaces estão definidas no módulo `io`. A forma canônica de se criar um objeto arquivo é por meio da função `open()`.

file-like object (objeto como a um arquivo) Um sinônimo do termo *file object*.

finder An object that tries to find the *loader* for a module that is being imported.

Desde o Python 3.3, existem dois tipos de localizadores: *meta path finders* para uso com `sys.meta_path`, e *path entry finders* para uso com `sys.path_hooks`.

Veja [PEP 302](#), [PEP 420](#) e [PEP 451](#) para maiores informações.

divisão pelo piso Mathematical division that rounds down to nearest integer. The floor division operator is `//`. For example, the expression `11 // 4` evaluates to 2 in contrast to the `2.75` returned by float true division. Note that `(-11) // 4` is `-3` because that is `-2.75` rounded *downward*. See [PEP 238](#).

function (função) A series of statements which returns some value to a caller. It can also be passed zero or more *arguments* which may be used in the execution of the body. See also *parameter*, *method*, and the function section.

function annotation (anotação de função) Uma *annotation* de um parâmetro ou retorno de uma função.

Anotações de função são comumente usados por *type hints*: por exemplo, essa função espera receber dois argumentos `int` e também é esperado que devolva um valor `int`:

```
def sum_two_numbers(a: int, b: int) -> int:
    return a + b
```

A sintaxe de anotação de uma função é explicada na seção *function*.

Veja *variable annotation* e [PEP 484](#), que descrevem essa funcionalidade.

__future__ A pseudo-module which programmers can use to enable new language features which are not compatible with the current interpreter.

Ao importar o módulo `__future__` e avaliar suas variáveis, você pode ver quando uma nova funcionalidade foi adicionada pela primeira vez à linguagem e quando ela se tornará padrão:

```
>>> import __future__
>>> __future__.division
_Feature((2, 2, 0, 'alpha', 2), (3, 0, 0, 'alpha', 0), 8192)
```

garbage collection (coletor de lixo) O processo de liberar a memória quando ela não é mais utilizada. Python executa a liberação da memória através da contagem de referências e um coletor de lixo cíclico que é capaz de detectar e interromper referências cíclicas. O coletor de lixo pode ser controlado usando o módulo `gc`.

gerador A function which returns a *generator iterator*. It looks like a normal function except that it contains `yield` expressions for producing a series of values usable in a for-loop or that can be retrieved one at a time with the `next()` function.

Normalmente refere-se a uma função geradora, mas pode referir-se a um *iterador gerador* em alguns contextos. Em alguns casos onde o significado desejado não está claro, usar o termo completo evita ambiguidade.

iterador gerador Um objeto criado por uma função *gerador*.

Cada `yield` suspende temporariamente o processamento, memorizando o estado da execução local (incluindo variáveis locais e instruções `try` pendentes). Quando o *iterador gerador* retorna, ele se recupera do último ponto onde estava (em contrapartida as funções que iniciam uma nova execução a cada vez que são invocadas).

generator expression Uma expressão que retorna um iterador. Parece uma expressão normal, seguido de uma cláusula `for` definindo uma variável de loop, um `range`, e uma cláusula `if` opcional. A expressão combinada gera valores para uma função encapsuladora:

```
>>> sum(i*i for i in range(10))           # sum of squares 0, 1, 4, ... 81
285
```

generic function (função genérica) Uma função composta por múltiplas funções implementando a mesma operação para diferentes tipos. Qual implementação deverá ser usada durante a execução é determinada pelo algoritmo de despacho.

Veja também a entrada *single dispatch* no glossário, o decorador `functools.singledispatch()`, e a **PEP 443**.

GIL Veja *global interpreter lock*.

global interpreter lock (bloqueio global do intérprete) The mechanism used by the *CPython* interpreter to assure that only one thread executes Python *bytecode* at a time. This simplifies the CPython implementation by making the object model (including critical built-in types such as `dict`) implicitly safe against concurrent access. Locking the entire interpreter makes it easier for the interpreter to be multi-threaded, at the expense of much of the parallelism afforded by multi-processor machines.

However, some extension modules, either standard or third-party, are designed so as to release the GIL when doing computationally-intensive tasks such as compression or hashing. Also, the GIL is always released when doing I/O.

Past efforts to create a “free-threaded” interpreter (one which locks shared data at a much finer granularity) have not been successful because performance suffered in the common single-processor case. It is believed that overcoming this performance issue would make the implementation much more complicated and therefore costlier to maintain.

pyc baseado em hash Um arquivo de cache em bytecode que usa hash ao invés do tempo (no qual o arquivo de código-fonte foi modificado pela última vez) para determinar a sua validade. Veja `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.

Hashability makes an object usable as a dictionary key and a set member, because these data structures use the hash value internally.

A maioria dos objetos embutidos imutáveis do Python são hasheáveis; containers mutáveis (tais como listas ou dicionários) não são; containers imutáveis (tais como tuplas e `frozensets`) são hasheáveis apenas se os seus elementos são hasheáveis. Objetos que são instâncias de classes definidas pelo usuário são hasheáveis por padrão. Todos eles comparam de forma desigual (exceto entre si mesmos), e o seu valor hash é derivado a partir do seu `id()`.

IDLE Um ambiente de desenvolvimento integrado para Python. IDLE é um editor básico e um ambiente interpretador que vem junto com a distribuição padrão do Python.

imutável Um objeto que possui um valor fixo. Objetos imutáveis incluem números, strings e tuplas. Estes objetos não podem ser alterados. Um novo objeto deve ser criado se um valor diferente tiver de ser armazenado. Objetos imutáveis têm um papel importante em lugares onde um valor constante de hash seja necessário, como por exemplo uma chave em um dicionário.

import path Uma lista de localizações (ou *path entries*) que são buscadas pelo *path based finder* por módulos para importar. Durante a importação, esta lista de localizações usualmente vem a partir de `sys.path`, mas para sub-pacotes ela também pode vir do atributo `__path__` de pacotes-pai.

importando O processo pelo qual o código Python em um módulo é disponibilizado para o código Python em outro módulo.

importer Um objeto que localiza e carrega um módulo; Tanto um *finder* e o objeto *loader*.

interactive Python has an interactive interpreter which means you can enter statements and expressions at the interpreter prompt, immediately execute them and see their results. Just launch `python` with no arguments (possibly by selecting it from your computer's main menu). It is a very powerful way to test out new ideas or inspect modules and packages (remember `help(x)`).

interpretado Python é uma linguagem interpretada, em oposição àquelas que são compiladas, embora esta distinção possa ser nebulosa devido à presença do compilador de bytecode. Isto significa que os arquivos-fontes podem ser executados diretamente sem necessidade explícita de se criar um arquivo executável. Linguagens interpretadas normalmente têm um ciclo de desenvolvimento/depuração mais curto que as linguagens compiladas, apesar de seus programas geralmente serem executados mais lentamente. Veja também *interativo*.

interpreter shutdown Quando solicitado para desligar, o interpretador Python entra em uma fase especial, onde ele gradualmente libera todos os recursos alocados, tais como módulos e várias estruturas internas críticas. Ele também faz diversas chamadas para o *garbage collector*. Isto pode disparar a execução de código em destrutores definidos pelo usuário ou callbacks de referência fraca. Código executado durante a fase de desligamento pode encontrar diversas exceções, pois os recursos que ele depende podem não funcionar mais (exemplos comuns são os módulos de bibliotecas, ou os mecanismos de avisos).

A principal razão para o interpretador desligar, é que o módulo `__main__` ou o script sendo executado terminou sua execução.

iterável Um objeto capaz de retornar seus membros um de cada vez. Exemplos de iteráveis incluem todos os tipos de sequência (tais como `list`, `str` e `tuple`) e alguns tipos de não-sequência, como o `dict`, *file objects*, além dos objetos de quaisquer classes que você definir com um método `__iter__()` ou `__getitem__()` que implementam a semântica de *sequência*.

Iteráveis podem ser usados em um laço `for` e em vários outros lugares em que uma sequência é necessária (`zip()`, `map()`, ...). Quando um objeto iterável é passado como argumento para a função nativa `iter()`, ela retorna um iterador para o objeto. Este iterador é adequado para se varrer todo o conjunto de valores. Ao usar iteráveis, normalmente não é necessário chamar `iter()` ou lidar com os objetos iteradores em si. A instrução `for` faz isso automaticamente para você, criando uma variável temporária para armazenar o iterador durante a execução do laço. Veja também *iterador*, *sequência*, e *gerador*.

iterador Um objeto que representa um fluxo de dados. Repetidas chamadas ao método `__next__()` de um iterador (ou passando o objeto para a função nativa `next()`) vão retornar itens sucessivos do fluxo. Quando não houver mais dados disponíveis uma exceção `StopIteration` exception será levantada. Neste ponto, o objeto iterador se esgotou e quaisquer chamadas subsequentes a seu método `__next__()` vão apenas levantar a exceção `StopIteration` novamente. Iteradores precisam ter um método `__iter__()` que retorne o objeto iterador em si, de forma que todo iterador também é iterável e pode ser usado na maioria dos lugares em que um iterável é requerido. Uma notável exceção é código que tenta realizar passagens em múltiplas iterações. Um objeto contêiner (como uma `list`) produz um novo iterador a cada vez que você passá-lo para a função `iter()` ou utilizá-lo em um laço `for`. Tentar isso com o mesmo iterador apenas iria retornar o mesmo objeto iterador esgotado já utilizado na iteração anterior, como se fosse um contêiner vazio.

Mais informações podem ser encontradas em *typeiter*.

Função chave A key function or collation function is a callable that returns a value used for sorting or ordering. For example, `locale.strxfrm()` is used to produce a sort key that is aware of locale specific sort conventions.

A number of tools in Python accept key functions to control how elements are ordered or grouped. They include `min()`, `max()`, `sorted()`, `list.sort()`, `heapq.merge()`, `heapq.nsmallest()`, `heapq`.

`nlargest()`, and `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.

keyword argument (Argumento de Palavra-Chave) Veja o [argument](#).

lambda Uma função de linha anônima consistindo de uma única [expression](#), que é avaliada quando a função é chamada. A sintaxe para criar uma função `lambda` é `lambda [parameters]: expression`

LBYL Look before you leap. This coding style explicitly tests for pre-conditions before making calls or lookups. This style contrasts with the [EAFP](#) approach and is characterized by the presence of many `if` statements.

In a multi-threaded environment, the LBYL approach can risk introducing a race condition between “the looking” and “the leaping”. For example, the code, `if key in mapping: return mapping[key]` can fail if another thread removes *key* from *mapping* after the test, but before the lookup. This issue can be solved with locks or by using the EAFP approach.

list Uma [sequence](#) embutida no Python. Apesar do seu nome, é mais próximo de um vetor em outras linguagens do que uma lista encadeada, como o acesso aos elementos é da ordem $O(1)$.

list comprehension A compact way to process all or part of the elements in a sequence and return a list with the results. `result = ['{:04x}'.format(x) for x in range(256) if x % 2 == 0]` generates a list of strings containing even hex numbers (0x..) in the range from 0 to 255. The `if` clause is optional. If omitted, all elements in `range(256)` are processed.

carregador 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](#).

método mágico Um sinônimo informal para um [special method](#).

mapeando 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 Um [finder](#) retornado por uma busca de `sys.meta_path`. Meta localizadores de diretórios são relacionados a, mas diferentes de [path entry finders](#).

Veja `importlib.abc.MetaPathFinder` para os métodos que meta localizadores de diretórios implementam.

metaclass The class of a class. Class definitions create a class name, a class dictionary, and a list of base classes. The metaclass is responsible for taking those three arguments and creating the class. Most object oriented programming languages provide a default implementation. What makes Python special is that it is possible to create custom metaclasses. Most users never need this tool, but when the need arises, metaclasses can provide powerful, elegant solutions. They have been used for logging attribute access, adding thread-safety, tracking object creation, implementing singletons, and many other tasks.

More information can be found in metaclasses.

method (método) A function which is defined inside a class body. If called as an attribute of an instance of that class, the method will get the instance object as its first [argument](#) (which is usually called `self`). See [function](#) and [nested scope](#).

method resolution order (ordem de resolução de método) 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.

módulo Um objeto que serve como uma unidade organizacional de código Python. Os módulos têm um namespace contendo objetos Python arbitrários. Os módulos são carregados pelo Python através do processo de: *term: importing*.

Veja também *package*.

module spec (módulo spec) Uma namespace que contém as informações relacionadas à importação usadas para carregar um módulo. Uma instância de `class:importlib.machinery.ModuleSpec`.

MRO See *method resolution order*.

mutable (mutável) Objeto mutável é aquele que pode modificar seus valor mas manter seu `id()`. Veja também *immutable*.

named tuple O termo “tupla nomeada” é aplicado a qualquer tipo ou classe que herda de `tuple` e cujos elementos indexáveis também são acessíveis usando atributos nomeados. O tipo ou classe pode ter outras funcionalidades também.

Diversos tipos embutidos são tuplas nomeadas, incluindo os valores retornados por `time.localtime()` e `os.stat()`. Outro exemplo é `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
```

Algumas tuplas nomeadas são tipos embutidos (tal como os exemplos acima). Alternativamente, uma tupla nomeada pode ser criada a partir da definição de uma classe regular, que herde de `tuple` e que defina campos nomeados. Tal classe pode ser escrita a mão, ou ela pode ser criada com uma função `factory collections.namedtuple()`. A segunda técnica também adiciona alguns métodos extras, que podem não ser encontrados quando foi escrita manualmente, ou em tuplas nomeadas embutidas.

namespace O lugar em que uma variável é armazenada. Namespaces são implementados como dicionários. Existem os namespaces local, global e nativo, bem como namespaces aninhados em objetos (em métodos). Namespaces suportam modularidade ao prevenir conflitos de nomes. Por exemplo, as funções `__builtin__.open()` e `os.open()` são diferenciadas por seus namespaces. Namespaces também auxiliam na legibilidade e na manutenibilidade ao tornar mais claro quais módulos implementam uma função. Escrever `random.seed()` ou `itertools.izip()`, por exemplo, deixa claro que estas funções são implementadas pelos módulos `random` e `itertools` respectivamente.

namespace package (espaço de nomes do pacote) Um *package PEP 420* que serve apenas como container para sub pacotes. Pacotes de namespaces podem não ter representação física, e especificamente não são como um *regular package* porque eles não tem um arquivo `__init__.py`.

Veja também *module*.

nested scope (escopo aninhado) A habilidade de referir-se a uma variável em uma definição de fechamento. Por exemplo, uma função definida dentro de outra pode referenciar variáveis da função externa. Perceba que escopos aninhados por padrão funcionam apenas por referência e não por atribuição. Variáveis locais podem ler e escrever no escopo mais interno. De forma similar, variáveis globais podem ler e escrever para o namespace global. O `nonlocal` permite escrita para escopos externos.

new-style class (novo estilo de classes) Antigo nome para o tipo de classes agora usado para todos os objetos de classes. Em versões anteriores do Python, apenas classes com o novo estilo podiam usar recursos novos e versáteis do Python, tais como `__slots__`, descritores, propriedades, `__getattr__()`, métodos de classe, e métodos estáticos.

object (objeto) Qualquer dado que tenha estado (atributos ou valores) e comportamento definidos (métodos). Também a última classe base de qualquer *new-style class*.

pacote Um *module* Python é capaz de conter submódulos ou recursivamente, sub-pacotes. Tecnicamente, um pacote é um módulo Python com um atributo `__path__`.

Veja também *regular package* e *namespace package*.

parameter (parâmetro) Uma entidade nomeada na definição de uma *função* (ou método) que especifica um *argumento* (ou em alguns casos, argumentos) que a função pode receber. Existem cinco tipos de parâmetros:

- *posicional-ou-nomeado*: especifica um argumento que pode ser tanto *posicional* quanto *nomeado*. Esse é o tipo padrão de parâmetro, por exemplo *foo* e *bar* a seguir:

```
def func(foo, bar=None): ...
```

- *somente-posicional*: especifica um argumento que pode ser passado para a função somente por posição. Python não possui sintaxe para definir parâmetros somente-posicionais. Contudo, algumas funções embutidas possuem argumentos somente-posicionais (por exemplo, `abs()`).
- *somente-nomeado*: especifica um argumento que pode ser passado para a função somente por nome. Parâmetros somente-nomeados podem ser definidos com um simples parâmetro var-posicional ou um `*` antes deles na lista de parâmetros na definição da função, por exemplo *kw_only1* and *kw_only2* a seguir:

```
def func(arg, *, kw_only1, kw_only2): ...
```

- *var-posicional*: especifica quem uma sequência arbitrária de argumentos posicionais pode ser fornecida (em adição a qualquer argumento posicional já aceito por outros parâmetros). Tal parâmetro pode ser definido colocando um `*` antes do nome, por exemplo *args* a seguir:

```
def func(*args, **kwargs): ...
```

- *var-nomeado*: especifica que, arbitrariamente, muitos argumentos nomeados podem ser fornecidos (em adição a qualquer argumento nomeado já aceito por outros parâmetros). Tal parâmetro pode ser definido colocando-se `**` antes do nome, por exemplo *kwargs* no exemplo acima.

Parâmetros podem especificar tanto argumentos opcionais quanto obrigatórios, assim como valores padrões para alguns argumentos opcionais.

Veja o termo *argument* no glossário, a questão :ref:`sobre a diferença entre argumentos e parâmetros <faq-argument-vs-parameter>` na FAQ, a classe `inspect.Parameter`, a seção *função*, e [PEP 362](#).

entrada de caminho Um local único no term:`import path` que o *path based finder* consulta para encontrar módulos a serem importados.

path entry finder (localizador de entrada de path) Um *finder* retornado por um callable em `sys.path_hooks` (ou seja, um *path entry hook*) que sabe como localizar os módulos *path entry*.

Veja `importlib.abc.PathEntryFinder` para os métodos implementadores da entrada do path.

path entry hook (hook do path de entrada) Um callable na lista `sys.path_hook` que retorna um *path entry finder* caso saiba como encontrar módulos em um local específico *path entry*.

path based finder Uma das opções padrão *meta path finders* que será procurado por módulos *import path*.

objeto caminho ou similar Um objeto representando um arquivo de caminho do sistema. Um objeto caminho ou similar é ou um objeto `str` ou `bytes` representando um caminho, ou um objeto implementando o protocolo `os.PathLike`. Um objeto que suporta o protocolo `os.PathLike` pode ser convertido para um arquivo de caminho do sistema `str` ou `bytes`, através da chamada da função `os.fspath()`; `os.fsdecode()` e `os.fsencode()` podem ser usadas para garantir um `str` ou `bytes` como resultado, respectivamente. Introduzido na [PEP 519](#).

PEP Proposta de melhoria do Python. Uma PEP é um documento de design que fornece informação para a comunidade Python, ou descreve uma nova funcionalidade para o Python ou seus predecessores ou ambientes. PEPs devem prover uma especificação técnica concisa e um racional para funcionalidades propostas.

PEPs tem a intenção de ser os mecanismos primários para propor novas funcionalidades significativas, para coletar opiniões da comunidade sobre um problema, e para documentar as decisões de design que foram adicionadas ao Python. O autor da PEP é responsável por construir um consenso dentro da comunidade e documentar opiniões dissidentes.

Veja [PEP 1](#).

parte Um conjunto de arquivos em um único diretório (possivelmente armazenado em um arquivo zip) que contribuem para um pacote de namespace, conforme definido em [PEP 420](#).

positional argument (argumento posicional) Veja o [argument](#).

API provisória Uma API provisória é uma API que foi deliberadamente excluída das bibliotecas padrões com compatibilidade retroativa garantida. Enquanto mudanças maiores para tais interfaces não são esperadas, contanto que elas sejam marcadas como provisórias, mudanças retroativas incompatíveis (até e incluindo a remoção da interface) podem ocorrer se consideradas necessárias pelos desenvolvedores principais. Tais mudanças não serão feitas gratuitamente – elas irão ocorrer apenas se sérias falhas fundamentais forem descobertas, que foram esquecidas anteriormente a inclusão da API.

Mesmo para APIs provisórias, mudanças retroativas incompatíveis são vistas como uma “solução em último caso” - cada tentativa ainda será feita para encontrar uma resolução retroativa compatível para quaisquer problemas encontrados.

Esse processo permite que a biblioteca padrão continue a evoluir com o passar do tempo, sem se prender em erros de design problemáticos por períodos de tempo prolongados. Veja [PEP 411](#) para mais detalhes.

pacote provisório Veja [provisional API](#).

Python 3000 Apelido para a versão do Python 3.x linha de lançamento (cunhado há muito tempo, quando o lançamento da versão 3 era algo em um futuro muito distante.) Esse termo possui a seguinte abreviação: “Py3k”.

Pythonic Uma ideia ou um pedaço de código que segue de perto os idiomas mais comuns da linguagem Python, ao invés de implementar códigos usando conceitos comuns a outros idiomas. Por exemplo, um idioma comum em Python é fazer um loop sobre todos os elementos de uma iterável usando a instrução: *for* statement. Muitas outras línguas não têm esse tipo de construção, então as pessoas que não estão familiarizadas com o Python usam um contador numérico:

```
for i in range(len(food)):
    print(food[i])
```

Ao contrário do método limpo, ou então, Pythonico:

```
for piece in food:
    print(piece)
```

qualified name (nome qualificado) Um nome pontilhado (quando 2 termos são ligados por um ponto) que mostra o “path” do escopo global de um módulo para uma classe, função ou método definido num determinado módulo, conforme definido pela [PEP 3155](#). Para funções e classes de nível superior, o nome qualificado é o mesmo que o nome do objeto:

```
>>> class C:
...     class D:
...         def meth(self):
...             pass
...
>>> C.__qualname__
```

(continua na próxima página)

(continuação da página anterior)

```
'C'
>>> C.D.__qualname__
'C.D'
>>> C.D.meth.__qualname__
'C.D.meth'
```

Quando usado para se referir a módulos, o *fully qualified name* significa todo o caminho pontilhado para o módulo, incluindo quaisquer pacotes pai, por exemplo: `email.mime.text`:

```
>>> import email.mime.text
>>> email.mime.text.__name__
'email.mime.text'
```

reference count O número de referências para um objeto. Quando a contagem de referências de um objeto atinge zero, ele é desalocado. Contagem de referências geralmente não é visível no código Python, mas é um elemento chave da implementação *CPython*. O módulo `sys` define a função `getrefcount()` que programadores podem chamar para retornar a contagem de referências para um objeto em particular.

regular package Um *package* tradicional, como um diretório contendo um arquivo `__init__.py`.

Veja também *namespace package*.

__slots__ A declaração dentro de uma classe que salva memória através de pré-declarações de espaço para atributos das instâncias, e eliminando dicionários de instâncias. Apesar de popular, a técnica é um tanto quanto complicada de acertar, e é melhor se for reservada para casos raros, onde existe uma grande quantidade de instâncias em uma aplicação onde a memória é crítica.

sequência Um *iterable* com suporte para acesso eficiente a seus elementos através de índices inteiros via método especial `__getitem__()` e que define o método `__len__()` que devolve o tamanho da sequência. Alguns tipos de sequência nativos são: `list`, `str`, `tuple`, e `bytes`. Note que `dict` também tem suporte para `__getitem__()` e `__len__()`, mas é considerado um mapa e não uma sequência porque a busca usa uma chave *imutável* arbitrária em vez de inteiros.

A classe base abstrata `collections.abc.Sequence` define uma interface mais rica que vai além de apenas `__getitem__()` e `__len__()`, adicionando `count()`, `index()`, `__contains__()`, e `__reversed__()`. Tipos que implementam essa interface podem ser explicitamente registrados usando `register()`.

single dispatch (despacho único) Uma forma do *generic function* despacho onde a implementação é escolhida com base no tipo de um único argumento.

slice Um objeto geralmente contendo uma parte de uma *sequence*. Uma fatia é criada usando a notação de subscrito `[]` pode conter também até dois pontos entre números, como em `variable_name[1:3:5]`. A notação de suporte (subscrito) utiliza objetos `slice` internamente.

método especial Um método que é chamado implicitamente pelo Python para executar uma certa operação em um tipo, como uma adição por exemplo. Tais métodos tem nomes iniciando e terminando com dois underscores. Métodos especiais estão documentados em `specialnames`.

declaração Uma instrução é parte de uma suíte (um “bloco” de código). Uma instrução é ou uma *expression* ou uma de várias construções com uma palavra-chave, tal como `if`, `while` ou `for`.

codificador de texto Um codec que codifica strings Unicode para bytes.

arquivo texto Um *file object* apto a ler e escrever objetos `str`. Geralmente, um arquivo texto, na verdade, acesse um fluxo de dados de bytes e captura o *text encoding* automaticamente. Exemplos de arquivos texto são: arquivos abertos em modo texto (`'r'` or `'w'`), `sys.stdin`, `sys.stdout`, e instâncias de `io.StringIO`.

Veja também *binary file* para um objeto arquivo apto a ler e escrever *bytes-like objects*.

aspas triplas Uma string que está definida com três ocorrências de aspas duplas (") ou apóstrofes ('). Enquanto elas não fornecem nenhuma funcionalidade não disponível com strings de aspas simples, elas são úteis para inúmeras razões. Elas permitem que você inclua aspas simples e duplas não encerradas dentro de uma string, e elas podem utilizar múltiplas linhas sem o uso de caracteres de continuação, fazendo-as especialmente úteis quando escrevemos documentação em docstrings.

tipo O tipo de um objeto Python determina qual tipo de objeto ele é; todos objetos tem um tipo. Um tipo de objeto é acessível pelo atributo `__class__` ou pode ser recuperado com `type(obj)`.

tipo alias Um sinônimo para tipo, criado através da atribuição do tipo para um identificador.

Tipos alias são úteis para simplificar *type hints*. Por exemplo:

```
from typing import List, Tuple

def remove_gray_shades(
    colors: List[Tuple[int, int, int]]) -> List[Tuple[int, int, int]]:
    pass
```

pode tornar-se mais legível desta forma:

```
from typing import List, Tuple

Color = Tuple[int, int, int]

def remove_gray_shades(colors: List[Color]) -> List[Color]:
    pass
```

Veja `typing` e [PEP 484](#), o qual descreve esta funcionalidade.

dica do tipo Uma *annotation* que especifica o tipo esperado para uma variável, um atributo de classe, ou um parâmetro de função ou um valor de retorno.

Dicas de tipo são opcionais e não são forçadas pelo Python, mas elas são úteis para ferramentas de análise estática de tipos, e ajudam IDEs a completar e refatorar código.

Dicas de tipos de variáveis globais, atributos de classes, e funções, mas não de variáveis locais, podem ser acessadas usando `typing.get_type_hints()`.

Veja `typing` e [PEP 484](#), o qual descreve esta funcionalidade.

Novas linhas universais Uma maneira de interpretar fluxos de textos, na qual todos estes são reconhecidos como caracteres de encerramento de linha: a convenção para fim-de-linha no Unix `'\\n'`, a convenção no Windows `'\\r\\n'`, e a antiga convenção no Macintosh `'\\r'`. Veja [PEP 278](#) e [PEP 3116](#), bem como `bytes.splitlines()` para uso adicional.

anotação variável Uma *annotation* de uma variável ou um atributo de classe.

Ao fazer uma anotação de uma variável ou atributo de classe, a atribuição é opcional:

```
class C:
    field: 'annotation'
```

Variable annotations are usually used for *type hints*: for example this variable is expected to take `int` values:

```
count: int = 0
```

A sintaxe de anotação de variável é explicada na seção `annassign`.

Veja *function annotation*, [PEP 484](#) e [PEP 526](#), que descrevem esta funcionalidade.

ambiente virtual Um ambiente de execução isolado que permite usuários Python e aplicações instalarem e atualizarem pacotes Python sem interferir no comportamento de outras aplicações Python em execução no mesmo sistema.

Veja também `venv`.

virtual machine Um computador definido inteiramente em software. A máquina virtual de Python executa o *bytecode* emitido pelo compilador de bytecode.

Zen of Python Lista de princípios de projeto e filosofias do Python que são úteis para a compreensão e uso da linguagem. A lista é exibida quando se digita `“import this”` no console interativo.

Sobre a Documentação

Estes documentos são gerados a partir de fontes [reStructuredText](#) utilizando [Sphinx](#), um processador de documentos escrito especificamente para a documentação do Python.

Desenvolvimento da documentação e suas ferramentas é um esforço totalmente voluntário, como o Python em si. Se você quer contribuir, por favor dê uma olhada na página [reporting-bugs](#) para informações sobre como fazer. Novos voluntários são sempre bem vindos!

Agradecimentos especiais para:

- Fred L. Drake, Jr., o criador do primeiro conjunto de ferramentas para documentar o Python e escritor de boa parte do conteúdo;
- O projeto [Docutils](#) por ter criado [reStructuredText](#) e a suíte [Docutils](#);
- Fredrik Lundh por sua [Referência Alternativa para Python](#) projeto do qual, [Sphinx](#) tirou muitas idéias boas.

B.1 Contribuidores da Documentação do Python

Muitas pessoas tem contribuído para a linguagem Python, sua biblioteca padrão e sua documentação. Veja [Misc/ACKS](#) na distribuição do código-fonte do Python para ver uma lista parcial de contribuidores.

Tudo isso só foi possível com o esforço e a contribuição da comunidade Python, por isso temos essa maravilhosa documentação – Obrigado a todos!

História e Licença

C.1 História do software

O Python foi criado no início dos anos 1990 por Guido van Rossum na Stichting Mathematisch Centrum (CWI, veja <https://www.cwi.nl/>) na Holanda como um sucessor de uma linguagem chamada ABC. Guido continua a ser o principal autor de Python, embora inclua muitas contribuições de outros.

Em 1995, Guido continuou seu trabalho em Python na Corporação para Iniciativas Nacionais de Pesquisa (CNRI, veja <https://www.cnri.reston.va.us/>) em Reston, Virgínia, onde lançou várias versões do software.

Em maio de 2000, Guido e a equipe de desenvolvimento principal do Python foram para BeOpen.com para formar a equipe do BeOpen PythonLabs. Em outubro do mesmo ano, a equipe do PythonLabs mudou-se para a Digital Creations (agora Zope Corporation; consulte <https://www.zope.org/>). Em 2001, a Python Software Foundation (PSF, consulte <https://www.python.org/psf/>) foi formada, uma organização sem fins lucrativos criada especificamente para possuir a Propriedade Intelectual relacionada ao Python. A Zope Corporation é um membro patrocinador do PSF.

Todas as versões do Python são de código aberto (consulte <https://opensource.org/> para a definição de código aberto). Historicamente, a maioria, mas não todas, versões do Python também são compatíveis com GPL; a tabela abaixo resume os vários lançamentos.

Release	Derivado de	Ano	Proprietário	GPL compatível?
0.9.0 a 1.2	n/a	1991-1995	CWI	sim
1.3 a 1.5.2	1.2	1995-1999	CNRI	sim
1.6	1.5.2	2000	CNRI	não
2.0	1.6	2000	BeOpen.com	não
1.6.1	1.6	2001	CNRI	não
2.1	2.0+1.6.1	2001	PSF	não
2.0.1	2.0+1.6.1	2001	PSF	sim
2.1.1	2.1+2.0.1	2001	PSF	sim
2.1.2	2.1.1	2002	PSF	sim
2.1.3	2.1.2	2002	PSF	sim
2.2 e acima	2.1.1	2001-agora	PSF	sim

Nota: Compatível com GPL não significa que estamos distribuindo Python sob a GPL. Todas as licenças do Python, ao contrário da GPL, permitem distribuir uma versão modificada sem fazer alterações em código aberto. As licenças compatíveis com GPL possibilitam combinar o Python com outro software lançado sob a GPL; os outros não.

Graças aos muitos voluntários externos que trabalharam sob a direção de Guido para tornar esses lançamentos possíveis.

C.2 Termos e condições para acessar ou usar Python

C.2.1 PSF LICENSE AGREEMENT FOR PYTHON 3.7.17

1. This LICENSE AGREEMENT is between the Python Software Foundation ("PSF"),
→and
the Individual or Organization ("Licensee") accessing and otherwise using
→Python
3.7.17 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 3.7.17 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 3.7.17 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 3.7.17 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
3.7.17.
4. PSF is making Python 3.7.17 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 3.7.17 WILL NOT INFRINGE ANY THIRD PARTY RIGHTS.
5. PSF SHALL NOT BE LIABLE TO LICENSEE OR ANY OTHER USERS OF PYTHON 3.7.17
FOR ANY INCIDENTAL, SPECIAL, OR CONSEQUENTIAL DAMAGES OR LOSS AS A RESULT
→OF
MODIFYING, DISTRIBUTING, OR OTHERWISE USING PYTHON 3.7.17, 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 3.7.17, Licensee agrees to be bound by the terms and conditions of this License Agreement.

C.2.2 ACORDO DE LICENÇA DA BEOPEN.COM PARA PYTHON 2.0

CONTRATO DE LICENÇA DE FONTE ABERTA DO BEOPEN PYTHON VERSÃO 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 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

(continua na próxima página)

(continuação da página anterior)

<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.

C.2.3 CONTRATO DE LICENÇA DA CNRI PARA O 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

(continua na próxima página)

(continuação da página anterior)

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 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.

C.2.4 ACORDO DE LICENÇA DA CWI PARA PYTHON 0.9.0 A 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.

C.3 Licenças e Reconhecimentos para Software Incorporado

Esta seção é uma lista incompleta, mas crescente, de licenças e confirmações para softwares de terceiros incorporados na distribuição do Python.

C.3.1 Mersenne Twister

O módulo: `mod: _random` inclui código baseado em um download de <http://www.math.sci.hiroshima-u.ac.jp/~m-mat/MT/MT2002/emt19937ar.html>. A seguir estão os comentários literais do código original:

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)`.

(continua na próxima página)

(continuação da página anterior)

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)

C.3.2 Sockets

O módulo: `mod: socket` usa as funções: `func: getaddrinfo` e: `func: getnameinfo`, que são codificadas em arquivos de origem separados do Projeto WIDE, <http://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.

(continua na próxima página)

(continuação da página anterior)

```
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.
```

C.3.3 Serviços de soquete assíncrono

Os módulos: `mod: asynchat` e: `mod: asyncore` contêm o seguinte aviso

```
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.
```

C.3.4 Gerenciamento de cookies

O módulo: `mod: http.cookies` contém o seguinte aviso

```
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.
```

(continua na próxima página)

(continuação da página anterior)

```
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.
```

C.3.5 Rastreamento de execução

O módulo: `mod: trace` contém o seguinte aviso

```
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.
```

C.3.6 Funções UUencode e UUdecode

O módulo: `mod: uu` contém o seguinte aviso

```
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
```

(continua na próxima página)

(continuação da página anterior)

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

C.3.7 Chamadas de Procedimento Remoto XML

O módulo: `mod: xmlrpc.client` contém o seguinte aviso

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 MERCHANTABILITY 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.

C.3.8 test_epoll

O módulo: mod: *test_epoll* contém o seguinte aviso

```
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.
```

C.3.9 Selezione o kqueue

O módulo: mod: *select* contém o seguinte aviso para a interface do kqueue

```
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.
```

C.3.10 SipHash24

O arquivo: file: *Python / pyhash.c* contém a implementação de Marek Majkowski do algoritmo SipHash24 de Dan Bernstein. Contém a seguinte nota

```
<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)
```

C.3.11 strtod e dtoa

O arquivo: file: *Python / dtoa.c*, que fornece as funções C *dtoa* e *strtod* para conversão de duplas de C para e de strings, é derivado do arquivo com o mesmo nome de David M. Gay, atualmente disponível em <http://www.netlib.org/fp/>. O arquivo original, conforme recuperado em 16 de março de 2009, contém os seguintes avisos de direitos autorais e de licenciamento

```
/*****
 *
 * 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.
 *****/
```

C.3.12 OpenSSL

Os módulos: `mod: hashlib`; `mod: posix`; `mod: ssl`; `mod: crypt` usam a biblioteca OpenSSL para desempenho adicional se forem disponibilizados pelo sistema operacional. Além disso, os instaladores do Windows e do Mac OS X para Python podem incluir uma cópia das bibliotecas do OpenSSL, portanto incluímos uma cópia da licença do OpenSSL aqui:

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.

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
```

(continua na próxima página)

(continuação da página anterior)

```

* 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.
* =====
*
* 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

```

(continua na próxima página)

(continuação da página anterior)

```
* 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.
*
* 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.]
*/
```

C.3.13 expat

A extensão: mod: *pyexpat* é construída usando uma cópia incluída das fontes de expatriadas, a menos que a compilação esteja configurada `--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.
```

C.3.14 libffi

A extensão: mod: `_ctypes` é construída usando uma cópia incluída das fontes libffi, a menos que a compilação esteja configurada `--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
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.
```

C.3.15 zlib

A extensão: mod: `zlib` é construída usando uma cópia incluída das fontes zlib se a versão do zlib encontrada no sistema for muito antiga para ser usada na construção

```
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          Mark Adler
jloup@gzip.org            madler@alumni.caltech.edu
```

C.3.16 cfuhash

A implementação da tabela de hash usada pelo: mod: *tracemalloc* é baseada no projeto 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 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.
```

C.3.17 libmpdec

O módulo: mod: *_decimal* é construído usando uma cópia incluída da biblioteca libmpdec, a menos que a compilação esteja configurada `--with-system-libmpdec`:

```
Copyright (c) 2008-2016 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.

(continua na próxima página)

(continuação da página anterior)

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.

APÊNDICE D

Direitos Autorais

Python e essa documentação é:

Copyright © 2001-2023 Python Software Foundation. All rights reserved.

Copyright © 2000 BeOpen.com. Todos os direitos reservados.

Copyright © 1995-2000 Corporation for National Research Initiatives. Todos os direitos reservados.

Copyright © 1991-1995 Stichting Mathematisch Centrum. Todos os direitos reservados.

Veja: *História e Licença* para informações completas de licença e permissões.

Não alfabético

..., [13](#)
2to3, [13](#)
>>>, [13](#)
__future__, [17](#)
__slots__, [24](#)

A

aguardável, [14](#)
ambiente virtual, [26](#)
Anotação, [13](#)
anotação variável, [25](#)
API provisória, [23](#)
argumento, [13](#)
arquivo binário, [14](#)
arquivo texto, [24](#)
aspas triplas, [25](#)
atributo, [14](#)

B

BDFL, [14](#)
bytecode, [15](#)

C

carregador, [20](#)
C-contiguous, [15](#)
Classe, [15](#)
classe base abstrata, [13](#)
co-rotina, [15](#)
codificador de texto, [24](#)
Coerção, [15](#)
contíguo, [15](#)
CPython, [16](#)

D

declaração, [24](#)
decorador, [16](#)
descritor, [16](#)
dica do tipo, [25](#)

dicionário, [16](#)
divisão pelo piso, [17](#)
docstring, [16](#)
duck-typing (*tipagem pato*), [16](#)

E

EAFP, [16](#)
entrada de caminho, [22](#)
expressão, [16](#)

F

f-string, [17](#)
file object (*arquivo objeto*), [17](#)
file-like object (*objeto como a um arquivo*), [17](#)
finder, [17](#)
Fortran contiguous, [15](#)
Função chave, [19](#)
função de co-rotina, [15](#)
function (*função*), [17](#)
function annotation (*anotação de função*), [17](#)

G

garbage collection (*coletor de lixo*), [17](#)
generator, [17](#)
generator expression, [18](#)
generic function (*função genérica*), [18](#)
gerador, [17](#)
gerador assíncrono, [14](#)
gerador iterador assíncrono, [14](#)
gerenciador de contexto, [15](#)
gerenciador de contexto assíncrono, [14](#)
GIL, [18](#)
global interpreter lock (*bloqueio global do intérprete*), [18](#)

H

hashable, [18](#)

I

IDLE, [18](#)

import path, [19](#)
importando, [19](#)
importer, [19](#)
imutável, [18](#)
interactive, [19](#)
interpretado, [19](#)
interpreter shutdown, [19](#)
iterador, [19](#)
iterador assíncrono, [14](#)
iterador gerador, [18](#)
iterável, [19](#)
iterável assíncrono, [14](#)

K

keyword argument (*Argumento de Palavra-Chave*), [20](#)

L

lambda, [20](#)
LBYL, [20](#)
list, [20](#)
list comprehension, [20](#)

M

magic
 method, [20](#)
mapeando, [20](#)
meta path finder, [20](#)
metaclass, [20](#)
method
 magic, [20](#)
 special, [24](#)
method (*método*), [20](#)
method resolution order (*ordem de resolução de método*), [20](#)
método especial, [24](#)
método mágico, [20](#)
module spec (*módulo spec*), [21](#)
módulo, [21](#)
módulo de extensão, [17](#)
MRO, [21](#)
mutable (*mutável*), [21](#)

N

named tuple, [21](#)
namespace, [21](#)
namespace package (*espaço de nomes do pacote*), [21](#)
nested scope (*escopo aninhado*), [21](#)
new-style class (*novo estilo de classes*), [21](#)
Novas linhas universais, [25](#)
número complexo, [15](#)

O

object (*objeto*), [21](#)

objeto byte ou similar, [14](#)
objeto caminho ou similar, [22](#)

P

pacote, [22](#)
pacote provisório, [23](#)
parameter (*parâmetro*), [22](#)
parte, [23](#)
path based finder, [22](#)
path entry finder (*localizador de entrada de path*), [22](#)
path entry hook (*hook do path de entrada*), [22](#)
PEP, [23](#)
positional argument (*argumento posicional*), [23](#)
Propostas Estendidas Python
 PEP 1, [23](#)
 PEP 238, [17](#)
 PEP 278, [25](#)
 PEP 302, [17](#), [20](#)
 PEP 343, [15](#)
 PEP 362, [14](#), [22](#)
 PEP 411, [23](#)
 PEP 420, [17](#), [21](#), [23](#)
 PEP 427, [3](#)
 PEP 443, [18](#)
 PEP 451, [17](#)
 PEP 484, [13](#), [17](#), [25](#)
 PEP 492, [14](#), [15](#)
 PEP 498, [17](#)
 PEP 519, [22](#)
 PEP 525, [14](#)
 PEP 526, [13](#), [25](#)
 PEP 3116, [25](#)
 PEP 3155, [23](#)
pyc baseado em hash, [18](#)
PyPI
 (see Python Package Index (PyPI)), [7](#)
Python 3000, [23](#)
Python Package Index (*PyPI*), [7](#)
Pythonic, [23](#)

Q

qualified name (*nome qualificado*), [23](#)

R

reference count, [24](#)
regular package, [24](#)

S

sequência, [24](#)
single dispatch (*despacho único*), [24](#)
slice, [24](#)
special

method, [24](#)

T

tipo, [25](#)

tipo alias, [25](#)

V

variável de classe, [15](#)

variável de contexto, [15](#)

virtual machine, [26](#)

visualização de dicionário, [16](#)

Z

Zen of Python, [26](#)