Python Setup and Usage

Release 3.8.20

Guido van Rossum
and the Python development team

setembro 08, 2024

Python Software Foundation
Email: docs@python.org

Sumário

Esta parte da documentação é dedicada a informações gerais sobre a configuração do ambiente Python em diferentes plataformas, a chamada do interpretador e outras coisas que facilitam o trabalho com o Python.

CAPÍTULO 1

Linha de comando e ambiente

O interpretador do CPython verifica a linha de comando e o ambiente em busca de várias configurações.

CPython implementation detail: Os esquemas de linha de comando de outras implementações podem ser diferentes. Consulte implementations para mais recursos.

1.1 Linha de comando

Ao invocar o Python, você pode especificar qualquer uma destas opções:

```
python [-bBdEhiIOqsSuvVWx?] [-c command | -m module-name | script | - ] [args]
```

O caso de uso mais comum é, obviamente, uma simples invocação de um script:

```
python myscript.py
```

1.1.1 Opções de interface

A interface do interpretador é semelhante à do console do UNIX, mas fornece alguns métodos adicionais de chamada:

- Quando chamado com a entrada padrão conectada a um dispositivo tty, ele solicita comandos e os executa até um EOF (um caractere de fim de arquivo, você pode produzi-lo com `Ctrl-D` no UNIX ou `Ctrl-Z, Enter` no Windows) ser lido.
- Quando chamado com um argumento de nome de arquivo ou com um arquivo como entrada padrão, ele lê e executa um script desse arquivo.
- Quando chamado com um argumento de nome de diretório, ele lê e executa um script nomeado adequadamente desse diretório.
- Quando chamado com `-c command`, ele executa as instruções Python fornecidas como *command*. Aqui *command* pode conter várias instruções separadas por novas linhas. O espaço em branco à esquerda é significativo nas instruções do Python!
- Quando chamado com `-m module-name`, o módulo fornecido está localizado no caminho do módulo Python e é executado como um script.

No modo não interativo, toda a entrada é analisada antes de ser executada.

Uma opção de interface termina a lista de opções consumidas pelo interpretador, todos os argumentos consecutivos terminam em `sys.argv` – observe que o primeiro elemento, subscrito zero (`sys.argv[0]`) , é uma string que reflete a fonte do programa.

-c <command>
: Executa o código Python em *command*. *command* pode ser uma ou mais instruções separadas por novas linhas, com espaços em branco à esquerda significativos, como no código normal do módulo.

  Se esta opção for fornecida, o primeiro elemento de `sys.argv` será `"-c"` e o diretório atual será adicionado ao início de `sys.path` (permitindo módulos nesse diretório para ser importado como módulos de nível superior).

  Levanta um evento de auditoria `cpython.run_command` com o argumento `command`.

-m <module-name>
: Procura `sys.path` pelo módulo nomeado e executa seu conteúdo como o módulo `__main__`.

  Como o argumento é um nome de *module*, você não deve fornecer uma extensão de arquivo (`.py`). O nome do módulo deve ser um nome de módulo Python absoluto válido, mas a implementação nem sempre pode impor isso (por exemplo, pode permitir que você use um nome que inclua um hífen).

  Nomes de pacotes (incluindo pacotes de espaço de nomes) também são permitidos. Quando um nome de pacote é fornecido ao invés de um módulo normal, o interpretador irá executar `<pkg>.__main__` como o módulo principal. Esse comportamento é deliberadamente semelhante ao tratamento de diretórios e arquivos zip que são passados para o interpretador como o argumento do script.

  Nota: Esta opção não pode ser usada com módulos embutidos e módulos de extensão escritos em C, uma vez que eles não possuem arquivos de módulo Python. No entanto, ele ainda pode ser usado para módulos pré-compilados, mesmo se o arquivo fonte original não estiver disponível.

  Se esta opção for fornecida, o primeiro elemento de `sys.argv` será o caminho completo para o arquivo do módulo (enquanto o arquivo do módulo está sendo localizado, o primeiro elemento será definido como `"-m"`). Como com a opção *-c*, o diretório atual será adicionado ao início de `sys.path`.

  A opção *-I* pode ser usada para executar o script em modo isolado onde `sys.path` não contém nem o diretório atual nem o diretório de pacotes de sites do usuário. Todas as variáveis de ambiente `PYTHON*` são ignoradas também.

  Muitos módulos de biblioteca padrão contêm código que é chamado em sua execução como um script. Um exemplo é o módulo `timeit`:

  ```
  python -m timeit -s 'setup here' 'benchmarked code here'
  python -m timeit -h # for details
  ```

  Levanta um evento de auditoria `cpython.run_module` com o argumento `module-name`.

  Ver também:

  `runpy.run_module()` Funcionalidade equivalente diretamente disponível para o código Python

  PEP 338 – Executando módulos como scripts

  Alterado na versão 3.1: Forneça o nome do pacote para executar um submódulo `__main__`.

  Alterado na versão 3.4: pacotes de espaço de nomes também são suportados

-
: Lê os comandos da entrada padrão (`sys.stdin`). Se a entrada padrão for um terminal, *-i* está implícito.

  Se esta opção for fornecida, o primeiro elemento de `sys.argv` será `"-"` e o diretório atual será adicionado ao início de `sys.path`.

  Levanta um evento de auditoria `cpython.run_stdin` sem argumentos.

<script>
Executa o código Python contido em *script*, que deve ser um caminho do sistema de arquivos (absoluto ou relativo) referindo-se a um arquivo Python, um diretório contendo um arquivo `__main__.py`, ou um arquivo zip contendo um arquivo `__main__.py`.

Se esta opção for fornecida, o primeiro elemento de `sys.argv` será o nome do script conforme fornecido na linha de comando.

Se o nome do script se referir diretamente a um arquivo Python, o diretório que contém esse arquivo é adicionado ao início de `sys.path`, e o arquivo é executado como o módulo `__main__`.

Se o nome do script se referir a um diretório ou arquivo zip, o nome do script será adicionado ao início de `sys.path` e o arquivo `__main__.py` nesse local será executado como o módulo `__main__`.

A opção *-I* pode ser usada para executar o script em modo isolado onde `sys.path` não contém nem o diretório do script nem o diretório de pacotes de sites do usuário. Todas as variáveis de ambiente `PYTHON*` são ignoradas também.

Levanta um evento de auditoria `cpython.run_file` com o argumento `filename`.

Ver também:

`runpy.run_path()` Funcionalidade equivalente diretamente disponível para o código Python

Se nenhuma opção de interface for fornecida, *-i* está implícito, `sys.argv[0]` é uma string vazia (`""`) e o diretório atual será adicionado ao início de `sys.path`. Além disso, o preenchimento por tab e a edição do histórico são habilitados automaticamente, se disponíveis em sua plataforma (veja rlcompleter-config).

Ver também:

tut-invoking

Alterado na versão 3.4: Ativação automática de preenchimento com tab e edição de histórico.

1.1.2 Opções genéricas

-?
-h
--help
Exibe uma breve descrição de todas as opções de linha de comando.

-V
--version
Exibe o número da versão do Python e saia. O exemplo de saída poderia ser:

```
Python 3.8.0b2+
```

Quando fornecido duas vezes, exibe mais informações sobre a construção, como:

```
Python 3.8.0b2+ (3.8:0c076caaa8, Apr 20 2019, 21:55:00)
[GCC 6.2.0 20161005]
```

Novo na versão 3.6: A opção `-VV`.

1.1.3 Opções diversas

-b

Emite um aviso ao comparar `bytes` ou `bytearray` com `str` ou `bytes` com `int`. Emite um erro quando a opção é fornecida duas vezes (`-bb`).

Alterado na versão 3.5: Afeta comparações de `bytes` com `int`.

-B

Se fornecido, Python não tentará escrever arquivos `.pyc` na importação de módulos fonte. Veja também *PYTHONDONTWRITEBYTECODE*.

--check-hash-based-pycs `default|always|never`

Controla o comportamento de validação de arquivos `.pyc` baseados em hash. Veja pyc-invalidation. Quando definido como `default`, os arquivos de cache bytecode baseados em hash verificados e não verificados são validados de acordo com sua semântica padrão. Quando definido como `always`, todos os arquivos `.pyc` baseados em hash, sejam verificados ou não verificados, são validados para seus arquivos fonte correspondentes. Quando definido como `never`, os arquivos `.pyc` baseados em hash não são validados para seus arquivos fonte correspondentes.

A semântica dos arquivos `.pyc` baseados em marca de tempo não é afetada por esta opção.

-d

Ativa a saída de depuração do analisador sintático (apenas para especialistas, dependendo das opções de compilação). Veja também *PYTHONDEBUG*.

-E

Ignora todas as variáveis de ambiente `PYTHON*`, por exemplo *PYTHONPATH* e *PYTHONHOME*, que pode ser definido.

-i

Quando um script é passado como primeiro argumento ou a opção *-c* é usada, entre no modo interativo depois de executar o script ou o comando, mesmo quando `sys.stdin` não parece ser um terminal. O arquivo *PYTHONSTARTUP* não foi lido.

Isso pode ser útil para inspecionar variáveis globais ou um stack trace (situação da pilha de execução) quando um script levanta uma exceção. Veja também *PYTHONINSPECT*.

-I

Executa o Python no modo isolado. Isso também implica -E e -s. No modo isolado `sys.path` não contém o diretório do script nem o diretório de pacotes do site do usuário. Todas as variáveis de ambiente `PYTHON*` são ignoradas também. Outras restrições podem ser impostas para evitar que o usuário injete código malicioso.

Novo na versão 3.4.

-O

Remova as instruções de asserção e qualquer código condicional ao valor de `__debug__`. Aumenta o nome do arquivo para arquivos compilados (*bytecode*) adicionando `.opt-1` antes da extensão `.pyc` (veja **PEP 488**). Veja também *PYTHONOPTIMIZE*.

Alterado na versão 3.5: Modifica nomes de arquivos `.pyc` conforme a **PEP 488**.

-OO

Faz o mesmo que *-O* e também descarta docstrings. Aumenta o nome do arquivo para arquivos compilados (*bytecode*) adicionando `.opt-2` antes da extensão `.pyc` (veja **PEP 488**).

Alterado na versão 3.5: Modifica nomes de arquivos `.pyc` conforme a **PEP 488**.

-q

Não exibe as mensagens de direitos autorais e de versão nem mesmo no modo interativo.

Novo na versão 3.2.

-R

Habilita a aleatorização com hash. Esta opção só tem efeito se a variável de ambiente *PYTHONHASHSEED* estiver configurada para `0`, uma vez que a aleatorização com hash é habilitada por padrão.

Em versões anteriores do Python, esta opção ativa a aleatorização com hash, para que os valores `__hash__()` dos objetos str e bytes sejam "salgados" com um valor aleatório imprevisível. Embora permaneçam constantes em um processo Python individual, eles não são previsíveis entre invocações repetidas de Python.

A aleatorização com hash se destina a fornecer proteção contra uma negação de serviço causada por entradas cuidadosamente escolhidas que exploram o pior caso de desempenho de uma inserção de dicionário, complexidade O(n^2). Consulte http://www.ocert.org/advisories/ocert-2011-003.html para obter detalhes.

PYTHONHASHSEED permite que você defina um valor fixo para o segredo da semente de hash.

Alterado na versão 3.7: A opção não é mais ignorada.

Novo na versão 3.2.3.

-s

Não adiciona o `diretório site-packages de usuário` a `sys.path`.

Ver também:

PEP 370 – Diretório site-packages por usuário.

-S

Desabilita a importação do módulo `site` e as manipulações dependentes do site de `sys.path` que isso acarreta. Também desabilita essas manipulações se `site` for explicitamente importado mais tarde (chame `site.main()` se você quiser que eles sejam acionados).

-u

Força os fluxos stdout e stderr a serem sem buffer. Esta opção não tem efeito no fluxo stdin.

Veja também *PYTHONUNBUFFERED*.

Alterado na versão 3.7: A camada de texto dos fluxos stdout e stderr agora é sem buffer.

-v

Exibe uma mensagem cada vez que um módulo é inicializado, mostrando o local (nome do arquivo ou módulo embutido) de onde ele é carregado. Quando fornecido duas vezes (`-vv`), imprime uma mensagem para cada arquivo que é verificado durante a busca por um módulo. Também fornece informações sobre a limpeza do módulo na saída. Veja também *PYTHONVERBOSE*.

-W `arg`

Controle de aviso. O mecanismo de aviso do Python por padrão exibe mensagens de aviso para `sys.stderr`. Uma mensagem de aviso típica tem o seguinte formato:

```
file:line: category: message
```

Por padrão, cada aviso é exibido uma vez para cada linha fonte onde ocorre. Esta opção controla a frequência com que os avisos são exibidos.

Várias opções *-W* podem ser fornecidas; quando um aviso corresponde a mais de uma opção, a ação para a última opção correspondente é executada. As opções *-W* inválidas são ignoradas (embora, uma mensagem de aviso seja impressa sobre opções inválidas quando o primeiro aviso for emitido).

Os avisos também podem ser controlados usando a variável de ambiente *PYTHONWARNINGS* e de dentro de um programa Python usando o módulo `warnings`.

As configurações mais simples aplicam uma determinada ação incondicionalmente a todos os avisos emitidos por um processo (mesmo aqueles que são ignorados por padrão):

```
-Wdefault  # Warn once per call location
-Werror    # Convert to exceptions
-Walways   # Warn every time
-Wmodule   # Warn once per calling module
-Wonce     # Warn once per Python process
-Wignore   # Never warn
```

Os nomes das ações podem ser abreviados conforme desejado (por exemplo, `-Wi`, `-Wd`, `-Wa`, `-We`) e o interpretador irá resolvê-los para o nome de ação apropriado.

Veja warning-filter e describing-warning-filters para mais detalhes.

-x

Pula a primeira linha do código-fonte, permitindo o uso de formas não-Unix de `#!cmd`. Isso se destina apenas a um hack específico do DOS.

-X

Reservado para várias opções específicas de implementação. CPython atualmente define os seguintes valores possíveis:

- `-X faulthandler` para habilitar `faulthandler`;
- `-X showrefcount` para produzir a contagem de referências total e o número de blocos de memória usados quando o programa termina ou após cada instrução no interpretador interativo. Isso só funciona em compilações de depuração.
- `-X tracemalloc` para começar a rastrear alocações de memória do Python usando o módulo `tracemalloc`. Por padrão, apenas o quadro mais recente é armazenado no traceback (situação da pilha de execução) de um rastro. Use `-X tracemalloc=NFRAME` para iniciar o rastreamento com um limite de traceback de quadros *NFRAME*. Veja o `tracemalloc.start()` para mais informações.
- `-X showalloccount` to output the total count of allocated objects for each type when the program finishes. This only works when Python was built with `COUNT_ALLOCS` defined.
- `-X int_max_str_digits` configura a limitação de comprimento de string na conversão para inteiro. Veja também *PYTHONINTMAXSTRDIGITS*.
- `-X importtime` para mostrar quanto tempo leva cada importação. Mostra o nome do módulo, tempo cumulativo (incluindo importações aninhadas) e tempo próprio (excluindo importações aninhadas). Observe que sua saída pode ser interrompida em aplicações multithread. O uso típico é `python3 -X importtime -c 'import asyncio'`. Veja também *PYTHONPROFILEIMPORTTIME*.
- `-X dev`: enable CPython's "development mode", introducing additional runtime checks which are too expensive to be enabled by default. It should not be more verbose than the default if the code is correct: new warnings are only emitted when an issue is detected. Effect of the developer mode:
  - Add `default` warning filter, as *-W* `default`.
  - Install debug hooks on memory allocators: see the `PyMem_SetupDebugHooks()` C function.
  - Enable the `faulthandler` module to dump the Python traceback on a crash.
  - Enable asyncio debug mode.
  - Define o atributo `dev_mode` de `sys.flags` como `True`.
  - `io.IOBase` destructor logs `close()` exceptions.
- `-X utf8` habilita o modo UTF-8 para interfaces do sistema operacional, substituindo o modo com reconhecimento de localidade padrão. `-X utf8=0` desabilita explicitamente o modo UTF-8 (mesmo quando de outra forma seria ativado automaticamente). Veja *PYTHONUTF8* para mais detalhes.
- `-X pycache_prefix=PATH` permite a escrita de arquivos `.pyc` em uma árvore paralela enraizada em um determinado diretório em vez de na árvore de código. Veja também *PYTHONPYCACHEPREFIX*.

Também permite passar valores arbitrários e recuperá-los através do dicionário `sys._xoptions`.

Alterado na versão 3.2: A opção *-X* foi adicionada.

Novo na versão 3.3: A opção `-X faulthandler`.

Novo na versão 3.4: As opções `-X showrefcount` e `-X tracemalloc`.

Novo na versão 3.6: A opção `-X showalloccount`.

Novo na versão 3.7: As opções `-X importtime`, `-X dev` e `-X utf8`.

Novo na versão 3.8: A opção `-X pycache_prefix`. A opção `-X dev` agora registra exceções de `close()` no destruidor de `io.IOBase`.

Novo na versão 3.8.14: A opção `-X int_max_str_digits`.

1.1.4 Opções que você não deve usar

-J

Reservado para uso pelo Jython.

1.2 Variáveis de ambiente

Essas variáveis de ambiente influenciam o comportamento do Python, elas são processadas antes das opções de linha de comando diferentes de -E ou -I. É comum que as opções de linha de comando substituam as variáveis ambientais onde há um conflito.

PYTHONHOME

Altera a localização das bibliotecas Python padrão. Por padrão, as bibliotecas são pesquisadas em *prefix*`/lib/python`*version* e *exec_prefix*`/lib/python`*version*, onde *prefix* e *exec_prefix* são diretórios dependentes da instalação, ambos padronizando para `/usr/local`.

Quando *PYTHONHOME* é definido como um único diretório, seu valor substitui *prefix* e *exec_prefix*. Para especificar valores diferentes para estes, defina *PYTHONHOME* para *prefix*`:`*exec_prefix*.

PYTHONPATH

Aumenta o caminho de pesquisa padrão para arquivos de módulo. O formato é o mesmo `PATH` do shell: um ou mais caminhos de diretório separados por `os.pathsep` (por exemplo, dois pontos no Unix ou ponto e vírgula no Windows). Os diretórios inexistentes são ignorados silenciosamente.

Além dos diretórios normais, entradas individuais *PYTHONPATH* podem referir-se a arquivos zip contendo módulos Python puros (tanto no código-fonte quanto na forma compilada). Módulos de extensão não podem ser importados de arquivos zip.

O caminho de pesquisa padrão depende da instalação, mas geralmente começa com *prefix*`/lib/python`*version* (veja *PYTHONHOME* acima). É *sempre* anexado a *PYTHONPATH*.

Um diretório adicional será inserido no caminho de pesquisa antes de *PYTHONPATH* como descrito acima em *Opções de interface*. O caminho de pesquisa pode ser manipulado de dentro de um programa Python como a variável `sys.path`.

PYTHONSTARTUP

Se este for o nome de um arquivo legível, os comandos Python nesse arquivo serão executados antes que o primeiro prompt seja exibido no modo interativo. O arquivo é executado no mesmo espaço de nomes onde os comandos interativos são executados para que os objetos definidos ou importados nele possam ser usados sem qualificação na sessão interativa. Você também pode alterar os prompts `sys.ps1` e `sys.ps2` e o gancho `sys.__interactivehook__` neste arquivo.

Levanta um evento de auditoria `cpython.run_startup` com argumento `filename`.

PYTHONOPTIMIZE

Se for definido como uma string não vazia, é equivalente a especificar a opção *-O*. Se definido como um inteiro, é equivalente a especificar *-O* várias vezes.

PYTHONBREAKPOINT

Se estiver definida, ela nomeia um chamável usando a notação de caminho com pontos. O módulo que contém o chamável será importado e então o chamável será executado pela implementação padrão de `sys.breakpointhook()` que é chamado pelo `breakpoint()` embutido. Se não for definido, ou definido como uma string vazia, é equivalente ao valor "pdb.set_trace". Definir isso para a string "0" faz com que a implementação padrão de `sys.breakpointhook()` não faça nada além de retornar imediatamente.

Novo na versão 3.7.

PYTHONDEBUG
Se for definido como uma string não vazia, é equivalente a especificar a opção *-d*. Se definido como um inteiro, é equivalente a especificar *-d* várias vezes.

PYTHONINSPECT
Se for definido como uma string não vazia, é equivalente a especificar a opção *-i*.

Esta variável também pode ser modificada pelo código Python usando `os.environ` para forçar o modo de inspeção no encerramento do programa.

Levanta um evento de auditoria `cpython.run_stdin` sem argumentos.

Alterado na versão 3.8.20: Emits audit events.

PYTHONUNBUFFERED
Se for definido como uma string não vazia, é equivalente a especificar a opção *-u*.

PYTHONVERBOSE
Se for definido como uma string não vazia, é equivalente a especificar a opção *-v*. Se definido como um inteiro, é equivalente a especificar *-v* várias vezes.

PYTHONCASEOK
Se estiver definido, Python não diferencia letras maiúsculas e minúsculas nas instruções `import`. Isso só funciona no Windows e OS X.

PYTHONDONTWRITEBYTECODE
Se for definido como uma string não vazia, o Python não tentará escrever arquivos `.pyc` na importação de módulos fonte. Isso é equivalente a especificar a opção *-B*.

PYTHONPYCACHEPREFIX
Se estiver definido, o Python escreverá os arquivos `.pyc` em uma árvore de diretório espelho neste caminho, em vez de nos diretórios `__pycache__` dentro da árvore de fontes. Isso é equivalente a especificar a opção *-X* `pycache_prefix=PATH`.

Novo na versão 3.8.

PYTHONHASHSEED
Se esta variável não for definida ou definida como `random`, um valor aleatório é usado para semear os hashes de objetos str e bytes.

Se *PYTHONHASHSEED* for definido como um valor inteiro, ele é usado como uma semente fixa para gerar o hash() dos tipos cobertos pela aleatorização do hash.

Sua finalidade é permitir hash repetível, como autotestes do próprio interpretador, ou permitir que um cluster de processos Python compartilhe valores de hash.

O número inteiro deve ser um número decimal no intervalo [0,4294967295]. Especificar o valor 0 desabilitará a aleatorização de hash.

Novo na versão 3.2.3.

PYTHONINTMAXSTRDIGITS
Se esta variável estiver definida para um inteiro, é usada para configurar a limitação de comprimento de string na conversão para inteiro global do interpretador.

Novo na versão 3.8.14.

PYTHONIOENCODING
Se for definido antes de executar o interpretador, ele substitui a codificação usada para stdin/stdout/stderr, na sintaxe `encodingname:errorhandler`. Ambas as partes `encodingname` e `:errorhandler` são opcionais e têm o mesmo significado que em `str.encode()`.

Para stderr, a parte `:errorhandler` é ignorada; o tratador sempre será `'backslashreplace'`.

Alterado na versão 3.4: A parte `encodingname` é agora opcional.

Alterado na versão 3.6: No Windows, a codificação especificada por esta variável é ignorada para buffers de console interativo, a menos que *PYTHONLEGACYWINDOWSSTDIO* também seja especificado. Arquivos e canais redirecionados por meio de fluxos padrão não são afetados.

PYTHONNOUSERSITE
Se estiver definido, o Python não adicionará o diretório site-packages do usuário a sys.path.

Ver também:

PEP 370 – Diretório site-packages por usuário.

PYTHONUSERBASE
Define o diretório base do usuário, que é usado para calcular o caminho do diretório site-packages do usuário e caminhos de instalação do Distutils para python setup.py install --user.

Ver também:

PEP 370 – Diretório site-packages por usuário.

PYTHONEXECUTABLE
Se esta variável de ambiente for definida, sys.argv[0] será definido com seu valor em vez do valor obtido através do tempo de execução C. Funciona apenas no Mac OS X.

PYTHONWARNINGS
Isso é equivalente à opção *-W*. Se definido como uma string separada por vírgulas, é equivalente a especificar *-W* várias vezes, com os filtros posteriores na lista tendo precedência sobre os anteriores na lista.

As configurações mais simples aplicam uma determinada ação incondicionalmente a todos os avisos emitidos por um processo (mesmo aqueles que são ignorados por padrão):

```
PYTHONWARNINGS=default  # Warn once per call location
PYTHONWARNINGS=error    # Convert to exceptions
PYTHONWARNINGS=always   # Warn every time
PYTHONWARNINGS=module   # Warn once per calling module
PYTHONWARNINGS=once     # Warn once per Python process
PYTHONWARNINGS=ignore   # Never warn
```

Veja warning-filter e describing-warning-filters para mais detalhes.

PYTHONFAULTHANDLER
Se esta variável de ambiente for definida como uma string não vazia, faulthandler.enable() é chamado na inicialização: instale um tratador para os sinais SIGSEGV, SIGFPE, SIGABRT, SIGBUS and SIGILL para despejar o traceback (situação da pilha de execução) do Python. Isso é equivalente à opção *-X* faulthandler.

Novo na versão 3.3.

PYTHONTRACEMALLOC
Se esta variável de ambiente for definida como uma string não vazia, começa a rastrear as alocações de memória Python usando o módulo tracemalloc. O valor da variável é o número máximo de quadros armazenados em um traceback de um rastreamento. Por exemplo, PYTHONTRACEMALLOC=1 armazena apenas o quadro mais recente. Veja o tracemalloc.start() para mais informações.

Novo na versão 3.4.

PYTHONPROFILEIMPORTTIME
Se esta variável de ambiente for definida como uma string não vazia, o Python mostrará quanto tempo leva cada importação. Isso é exatamente equivalente a definir -X importtime na linha de comando.

Novo na versão 3.7.

PYTHONASYNCIODEBUG
Se esta variável de ambiente for definida como uma string não vazia, habilita o modo de depuração do módulo asyncio.

Novo na versão 3.4.

PYTHONMALLOC
Define os alocadores de memória Python e/ou instale ganchos de depuração.

Define a família de alocadores de memória usados pelo Python:

- `default`: usa os alocadores padrão de memória.
- `malloc`: usa a função `malloc()` da biblioteca C para todos os domínios (`PYMEM_DOMAIN_RAW`, `PYMEM_DOMAIN_MEM`, `PYMEM_DOMAIN_OBJ`).
- `pymalloc`: usa o alocador pymalloc para domínios `PYMEM_DOMAIN_MEM` e `PYMEM_DOMAIN_OBJ` e usa a função `malloc()` para o domínio `PYMEM_DOMAIN_RAW`.

Instala os ganchos de depuração:

- `debug`: instala os ganchos de depuração sobre os alocadores padrão de memória.
- `malloc_debug`: o mesmo que `malloc`, mas também instala ganchos de depuração.
- `pymalloc_debug`: o mesmo que `pymalloc`, mas também instala ganchos de depuração.

Veja os alocadores de memória padrão e a função `PyMem_SetupDebugHooks()` (instala ganchos de depuração em alocadores de memória Python).

Alterado na versão 3.7: Adicionado o alocador `"default"`.

Novo na versão 3.6.

PYTHONMALLOCSTATS
Se definido como uma string não vazia, o Python exibe estatísticas do alocador de memória pymalloc toda vez que uma nova arena de objeto pymalloc for criada e ao no desligamento.

Esta variável é ignorada se a variável de ambiente *PYTHONMALLOC* é usada para forçar o alocador `malloc()` da biblioteca C, ou se Python está configurado sem suporte a `pymalloc`.

Alterado na versão 3.6: Esta variável agora também pode ser usada em Python compilado no modo de lançamento. Agora não tem efeito se definido como uma string vazia.

PYTHONLEGACYWINDOWSFSENCODING
Se definido como uma string não vazia, a codificação do sistema de arquivos padrão e o modo de erros serão revertidos para seus valores pré-3.6 de “mbcs” e “replace”, respectivamente. Caso contrário, os novos padrões “utf-8” e “surrogatepass” serão usados.

Isso também pode ser habilitado em tempo de execução com `sys._enablelegacywindowsfsencoding()`.

Disponibilidade: Windows.

Novo na versão 3.6: Veja **PEP 529** para mais detalhes.

PYTHONLEGACYWINDOWSSTDIO
Se definido como uma string não vazia, não usa o novo leitor e escritor de console. Isso significa que os caracteres Unicode serão codificados de acordo com a página de código do console ativo, em vez de usar utf-8.

Esta variável é ignorada se os fluxos padrão forem redirecionados (para arquivos ou canais) em vez de se referir aos buffers do console.

Disponibilidade: Windows.

Novo na versão 3.6.

PYTHONCOERCECLOCALE
Se definido com o valor `0`, faz com que a aplicação principal de linha de comando Python ignore a coerção dos códigos de idioma legados C e POSIX baseados em ASCII para uma alternativa baseada em UTF-8 mais capaz.

Se esta variável *não* estiver definida (ou estiver definida para um valor diferente de `0`), a variável de ambiente de substituição de localidade `LC_ALL` também não será definida, e a localidade atual relatada para a categoria `LC_CTYPE` é a localidade `C` padrão, ou então a localidade `POSIX` explicitamente baseada em ASCII, então a CLI do Python tentará configurar as seguintes localidades para a categoria `LC_CTYPE` na ordem listada antes de carregar o tempo de execução do interpretador:

- `C.UTF-8`

- `C.utf8`
- `UTF-8`

Se a configuração de uma dessas categorias de local for bem-sucedida, a variável de ambiente `LC_CTYPE` também será configurada de acordo no ambiente de processo atual antes que o tempo de execução do Python seja inicializado. Isso garante que, além de ser visto pelo próprio interpretador e outros componentes com reconhecimento de localidade em execução no mesmo processo (como a biblioteca GNU `readline`), a configuração atualizada também é vista em subprocessos (independentemente de ou não esses processos estão executando um interpretador Python), bem como em operações que consultam o ambiente em vez da localidade C atual (como o `locale.getdefaultlocale()` do próprio Python).

Configurar uma dessas localidades (explicitamente ou por meio da coerção de localidade implícita acima) habilita automaticamente o tratador de erros `surrogateescape` para `sys.stdin` e `sys.stdout` (`sys.stderr` continua a usar `backslashreplace` como faz em qualquer outra localidade). Este comportamento de tratamento de fluxo pode ser substituído usando *`PYTHONIOENCODING`* como de costume.

Para fins de depuração, definir `PYTHONCOERCECLOCALE=warn` fará com que o Python emita mensagens de aviso em `stderr` se a coerção de localidade for ativada ou se uma localidade que *teria* acionado a coerção ainda estiver ativa quando o Python o tempo de execução é inicializado.

Observe também que mesmo quando a coerção de localidade está desabilitada, ou quando não consegue encontrar uma localidade de destino adequada, *`PYTHONUTF8`* ainda será ativado por padrão em localidades baseadas em ASCII legadas. Ambos os recursos devem ser desabilitados para forçar o interpretador a usar `ASCII` ao invés de `UTF-8` para interfaces de sistema.

Disponível: *nix.

Novo na versão 3.7: Veja **PEP 538** para mais detalhes.

`PYTHONDEVMODE`
If this environment variable is set to a non-empty string, enable the CPython "development mode". See the *`-X`* `dev` option.

Novo na versão 3.7.

`PYTHONUTF8`
Se definido como `1`, habilita o modo UTF-8 do interpretador, onde `UTF-8` é usado como a codificação de texto para interfaces do sistema, independentemente da configuração da localidade atual.

Isso significa que:

- `sys.getfilesystemencoding()` retorna `'UTF-8'` (a codificação da localidade é ignorada).
- `locale.getpreferredencoding()` retorna `'UTF-8'` (a codificação da localidade é ignorada e o parâmetro `do_setlocale` da função não tem efeito).
- `sys.stdin`, `sys.stdout` e `sys.stderr` usam UTF-8 como codificação de texto, com o tratador de erros `surrogateescape` sendo habilitado para `sys.stdin` e `sys.stdout` (`sys.stderr` continua a usar `backslashreplace` como faz no modo padrão com reconhecimento de localidade)

Como consequência das mudanças nessas APIs de baixo nível, outras APIs de alto nível também exibem comportamentos padrões diferentes:

- Argumentos de linha de comando, variáveis de ambiente e nomes de arquivos são decodificados em texto usando a codificação UTF-8.
- `os.fsdecode()` e `os.fsencode()` usam a codificação UTF-8.
- `open()`, `io.open()` e `codecs.open()` usam a codificação UTF-8 por padrão. No entanto, elas ainda usam o tratador de erros estrito por padrão, de modo que a tentativa de abrir um arquivo binário no modo de texto provavelmente levantará uma exceção em vez de produzir dados sem sentido.

Observe que as configurações de fluxo padrão no modo UTF-8 podem ser substituídas por *`PYTHONIOENCODING`* (assim como podem estar no modo com reconhecimento de localidade padrão).

Se definido como `0`, o interpretador executa em seu modo com reconhecimento de localidade padrão.

Definir qualquer outra string não vazia causa um erro durante a inicialização do interpretador.

Se esta variável de ambiente não for definida, o interpretador assume como padrão o uso das configurações de localidade atuais, *a menos que* a localidade atual seja identificada como uma localidade baseada em ASCII legada (conforme descrito para *PYTHONCOERCECLOCALE*), e localidade a coerção está desabilitada ou falha. Nessas localidades legadas, o interpretador assumirá como padrão a ativação do modo UTF-8, a menos que seja explicitamente instruído a não fazê-lo.

Também disponível como a opção *-X* `utf8`.

Novo na versão 3.7: Veja **PEP 540** para mais detalhes.

1.2.1 Variáveis de modo de depuração

Definir essas variáveis só tem efeito em uma construção de depuração do Python.

PYTHONTHREADDEBUG
Se definido, Python exibirá informações de depuração de threads.

Necessita do Python configurado com a opção de construção `--with-pydebug`.

PYTHONDUMPREFS
Se definido, Python irá despejar objetos e contagens de referências ainda vivas após desligar o interpretador.

Necessita do Python configurado com a opção de construção `--with-trace-refs`.

CAPÍTULO 2

Utilizando Python em plataformas Unix

2.1 Obtendo e instalando a versão mais recente do Python

2.1.1 No Linux

O Python vem pré-instalado na maioria das distribuições Linux e está disponível como um pacote em todos as outras. No entanto, existem certos recursos que podemos querer utilizar e que não estão disponíveis no pacote da sua distro. Poderás compilar facilmente a versão mais recente do Python desde a origem.

Nas situações em que o Python não vier pré-instalado e também não estiver nos repositórios, poderás facilmente gerar os pacotes para a sua distro. Veja os seguintes links:

Ver também:

https://www.debian.org/doc/manuals/maint-guide/first.en.html para usuários Debian

https://en.opensuse.org/Portal:Packaging para usuários OpenSuse

https://docs-old.fedoraproject.org/en-US/Fedora_Draft_Documentation/0.1/html/RPM_Guide/ch-creating-rpms.html
para usuários Fedora

http://www.slackbook.org/html/package-management-making-packages.html para usuários do Slackware

2.1.2 No FreeBSD e OpenBSD

- usuários do FreeBSD, para adicionar a utilização do pacote utilize:

```
pkg install python3
```

- Usuários do OpenBSD, para adicionar pacotes use:

```
pkg_add -r python

pkg_add ftp://ftp.openbsd.org/pub/OpenBSD/4.2/packages/<insert your␣
↪architecture here>/python-<version>.tgz
```

Por exemplo, usuários i386 podem pegar a versão 2.5.1 do Python usando o comando:

```
pkg_add ftp://ftp.openbsd.org/pub/OpenBSD/4.2/packages/i386/python-2.5.1p2.tgz
```

2.1.3 No OpenSolaris

Podes baixar o Python desde OpenCSW. Várias versões do Python estão disponíveis e poderás instala-las, por exemplo `pkgutil -i python27`.

2.2 Compilando o Python

Se você quer compilar o CPython, a primeira coisa que você precisa fazer é baixar o código-fonte. Você pode baixar a última versão ou usar o git para fazer um clone. (Se você pretende contribuir modificações, você vai precisar um "clone".)

O processo de compilação consiste nos comandos usuais:

```
./configure
make
make install
```

Opções de configuração e advertências para plataformas específicas do Unix estão amplamente documentadas no arquivo README.rst na raiz da árvore de fontes Python.

Aviso: `make install` pode sobrescrever ou mascarar o arquivo binário `python3`. `make altinstall` é, portanto, recomendado ao invés de `make install` uma vez que o mesmo apenas instala o arquivo *exec_prefix*`/bin/python`*version*.

2.3 Paths e arquivos relacionados com o Python

Estes estão sujeitos a diferenças dependendo das convenções de instalação local; `prefix` (`${prefix}`) e `exec_prefix` (`${exec_prefix}`) dependem da instalação e devem ser interpretados da mesma forma que para o software GNU; eles poderão ser os mesmos.

Por exemplo, na maioria dos sistemas Linux, o padrão para ambos é `/usr`.

Arquivo/diretório	Significado
exec_prefix`/bin/python3`	Localização recomendada do interpretador.
prefix`/lib/python`*version*, *exec_prefix*`/lib/python`*version*	A localização recomendada dos diretórios contendo os módulos padrão.
prefix`/include/python`*version*, *exec_prefix*`/include/python`*version*	Localizações recomendadas dos diretórios contendo os arquivos de inclusão necessários para o desenvolvimento de extensões Python e incorporação do interpretador.

2.4 Diversos

Para usar facilmente scripts Python no Unix, você precisa torná-los executáveis, por exemplo, com

```
$ chmod +x script
```

e colocar uma linha Shebang apropriada no topo do script. Uma boa escolha normalmente é

```
#!/usr/bin/env python3
```

que procura o interpretador do Python no conjunto `PATH`. No entanto, alguns sistemas Unix podem não ter o comando **env**, então você pode precisar codificar `/usr/bin/python3` como o caminho do interpretador.

Para usar comandos Shell em seus scripts Python, veja o módulo `subprocess`.

CAPÍTULO 3

Utilizando Python no Windows

Este documento pretende dar uma visão geral do comportamento específico do Windows que você deve conhecer quando fores utilizar o Python no sistema operacional Microsoft Windows.

Diferente da maioria dos sistemas e serviços Unix, o Windows não inclui uma instalação suportada do Python. Para deixar o Python disponível, o time CPython compilou os instaladores do Windows (pacotes MSI) com cada versão por vários anos. Esses instaladores têm a intenção primária de adicionar uma instalação de Python por usuário, com o interpretador e as bibliotecas núcleo sendo utilizadas por um único usuário. O instalador também é capaz de instalar para todos os usuários de uma única máquina, e um arquivo ZIP separado está disponível para distribuições locais de aplicação.

Como especificado na **PEP 11**, uma versão Python suporta apenas uma plataforma Windows enquanto a Microsoft considera a plataforma sob suporte estendido. Isso significa que o Python 3.8 suporta Windows Vista ou superiores. Se você requer suporte à Windows XP, então por favor instale o Python 3.4.

Há uma quantidade de instaladores diferentes disponíveis para Windows, cada um com algumas vantagens e desvantagens.

O instalador completo contém todos os componentes e é a melhor opção para desenvolvedores usando Python para qualquer tipo de projeto.

O pacote Microsoft Store é uma instalação simples do Python que é adequada para executar scripts e pacotes, e para usar a IDLE ou outros ambientes de desenvolvimento. Ela requer Windows 10, mas pode ser instalada de forma segura sem corromper outros programas. Ela também fornece vários comandos convenientes para iniciar o Python e suas ferramentas.

Os pacotes nuget.org são instalações leves criadas para integração contínua de sistemas. Elas podem ser usadas para construir pacotes Python ou executar scripts, mas não são atualizáveis e não possuem ferramentas de interface de usuário.

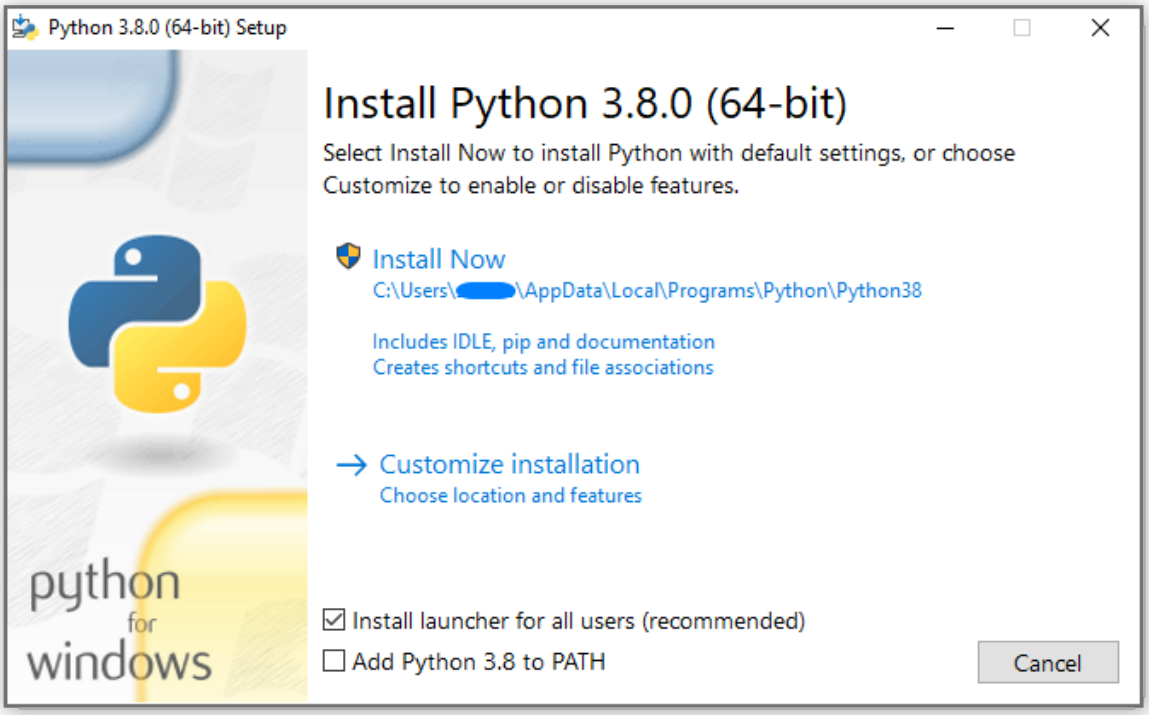
O pacote embutível é um pacote mínimo de Python adequado para ser incorporado em uma aplicação maior.

3.1 O instalador completo

3.1.1 Etapas de instalação

Quatro instaladores Python 3.8 estão disponíveis para download - dois de cada para as versões 32-bit e 64-bit do interpretador. O *instalador web* é um download inicial menor, e ele irá automaticamente fazer o download dos componentes solicitados na medida do necessário. O *instalador offline* inclui os componentes necessários para uma instalação padrão e requer apenas uma conexão de internet para recursos opcionais. Veja *Instalando Sem Download* para outras formas de evitar o download durante a instalação.

Após iniciar o instalador, uma de duas opções deve ser selecionada:



Se você selecionar “Install Now”:

- Você *não* precisará ser um administrador (a menos que uma atualização de sistema para a Biblioteca em Tempo de Execução C seja necessária ou que você instale o *Inicializador Python para Windows* para todos os usuários)
- Python será instalado dentro do seu diretório de usuário
- O *Inicializador Python para Windows* será instalado de acordo com a opção ao final da primeira página
- A biblioteca padrão, a suíte de testes, o inicializador e o pip serão instalados
- Se selecionado, o diretório de instalação será adicionado no seu `PATH`
- Atalhos serão visíveis apenas para o usuário atual

Selecionando “Customize installation” irá permitir que você selecione os recursos a serem instalados, o local da instalação e outras opções ou ações pós-instalação. Para instalar símbolos de depuração ou binários, você precisará usar essa opção.

Para realizar uma instalação para todos os usuários, você deve selecionar “Customize installation”. Neste caso:

- Você pode ser solicitado a providenciar credenciais administrativas ou aprovação
- Python será instalado dentro do diretório Program Files (Arquivos de Programa)
- O *Inicializador Python para Windows* será instalado dentro do diretório Windows

- Recursos opcionais podem ser selecionados durante a instalação
- A biblioteca padrão pode ser pré-compilada em bytecode
- Se selecionado, o diretório de instalação será adicionado ao `PATH` do sistema
- Atalhos estão disponíveis para todos os usuários

3.1.2 Removendo a Limitação do MAX_PATH

O Windows historicamente tem limitado os comprimentos dos caminhos de arquivos em 260 caracteres. Isso significava que caminhos maiores que isso não seriam resolvidos e resultariam em erros.

Nas últimas versões do Windows, essa limitação pode ser expandida para aproximadamente 32.000 caracteres. Seu administrador irá precisar ativar a política de grupo "Enable Win32 long paths", ou definir `LongPathsEnabled` para `1` na chave de registro `HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Control\FileSystem`.

Isso permite que a função `open()`, o módulo `os` e a maior parte das outras funcionalidades de caminho aceitem e retornem caminhos maiores que 260 caracteres quando usando strings.

Após alterar a opção acima, nenhuma configuração adicional é necessária.

Alterado na versão 3.6: Suporte para caminhos longos foi ativado no Python.

3.1.3 Instalando sem UI

Todas as opções disponíveis na IU do instalador também podem ser especificadas a partir da linha de comando, permitindo que instaladores por script repliquem uma instalação em várias máquinas sem interação do usuário. Essas opções também podem ser definidas sem suprimir a IU para alterar alguns dos padrões.

Para esconder completamente a IU do instalador e instalar o Python silenciosamente, passe a opção `/quiet`. Para pular a interação de usuário mas ainda exibir o progresso e os erros, passe a opção `/passive`. A opção `/uninstall` pode ser passada para imediatamente começar a remover o Python - nenhuma confirmação será solicitada.

Todas as outras opções são passadas como `name=value`, onde o valor é usualmente `0` para desabilitar o recurso, `1` para ativar o recurso, ou um caminho. A lista completa de opções disponíveis é mostrada abaixo.

Nome	Descrição	Default (padrão)
InstallAllUsers (Instalar para Todos usuários)	Execute uma instalação em todo o sistema.	0
TargetDir (Diretório Alvo)	O diretório de instalação	Selecionado com base em "Instalar para Todos os Usuários"
DefaultAllUsersTargetDir (Diretório de Destino Padrão para Todos os usuários)	O diretório de instalação padrão para instalações de todos os usuários	`%ProgramFiles%\Python X.Y` or `%ProgramFiles(x86)%\Python X.Y`
DefaultJustForMeTargetDir (Diretório Alvo Padrão Apenas Para Mim)	O diretório de instalação padrão para instalações just-for-me	`%LocalAppData%\Programs\PythonXY` or `%LocalAppData%\Programs\PythonXY-32` or `%LocalAppData%\Programs\PythonXY-64`
DefaultCustomTargetDir (Diretório de destino personalizado padrão)	Diretório de instalação personalizado padrão exibido na interface do usuário	(vazio)
AssociateFiles (Arquivos Associados)	Criar associações de arquivos se o launcher também estiver instalado.	1
CompileAll (Compilar Tudo)	Compile todos os arquivos `.py` para `.pyc`.	0
PrependPath (path a ser percorrido)	Adicione o diretório de instalação e o de Scripts para `PATH` e `.PY` para `PATHEXT`	0
Shortcuts (atalhos)	Crie atalhos para o interpretador, documentação e IDLE se instalado.	1
Include_doc	Install Python manual (Instalação Manual do Python)	1
Include_debug (Incluir o Modo de Depuração)	Instalar binários de Depuração	0
Include_dev	Instalar cabeçalhos e bibliotecas de desenvolvedores	1
Include_exe	Instalar `python.exe` e arquivos relacionados	1
Include_launcher	Instalar *Inicializador Python para Windows*.	1
InstallLauncherAllUsers	Instalar para *Inicializador Python para Windows* todos os usuários.	1
Include_lib	Instalar biblioteca padrão e módulos de extensão	1
Include_pip	Instale o pacote pip e setuptools	1
Include_symbols	Instalar símbolos de depuração (*.pdb)	0
Include_tcltk	Instale o suporte Tcl/Tk e o IDLE	1
Include_test	Instalar o conjunto de testes da biblioteca padrão	1
Include_tools	Instalar scripts com utilitários	1
LauncherOnly	Instala apenas o launcher. Isso substituirá a maioria das outras opções.	0
SimpleInstall	Desativar a maioria das UIs de instalação	0
SimpleInstallDescription	Uma mensagem personalizada para exibir quando a IU de instalação simplificada é usada.	(vazio)

Por exemplo, para instalar silenciosamente uma instalação de Python padrão e em todo o sistema, você pode usar o seguinte comando (a partir de um terminal de comando autorizado):

```
python-3.8.0.exe /quiet InstallAllUsers=1 PrependPath=1 Include_test=0
```

Para permitir que usuários instalem facilmente uma cópia do Python sem a suíte de testes, você pode fornecer um atalho com o seguinte comando. Isso irá exibir uma página inicial simplificado e bloquear a personalização:

```
python-3.8.0.exe InstallAllUsers=0 Include_launcher=0 Include_test=0
    SimpleInstall=1 SimpleInstallDescription="Just for me, no test suite."
```

(Observe que omitir o inicializador também omite associações de arquivos, e só é recomendado para instalações por usuários quando também existe uma instalação por todo o sistema que inclui o inicializador.)

As opções listadas acima também podem ser fornecidas em um arquivo chamado `unattend.xml` juntamente com o executável. Esse arquivo especifica uma lista de opções e valores. Quando um valor é fornecido como um atributo, ele será convertido para um número se possível. Valores fornecidos como elementos de texto são sempre deixados como strings. Esse arquivo de exemplo define as mesmas opções que o exemplo anterior:

```
<Options>
    <Option Name="InstallAllUsers" Value="no" />
    <Option Name="Include_launcher" Value="0" />
    <Option Name="Include_test" Value="no" />
    <Option Name="SimpleInstall" Value="yes" />
    <Option Name="SimpleInstallDescription">Just for me, no test suite</Option>
</Options>
```

3.1.4 Instalando Sem Download

Como alguns recursos do Python não estão inclusos no download inicial do instalador, selecionar esses recursos pode exigir uma conexão com a internet. Para evitar isso, todos os possíveis componentes podem ser baixados sob demanda para criar um *layout* completo que não irá precisar de uma conexão de internet independentemente dos recursos selecionados. Note que este download pode ser maior que o necessário, mas onde um grande número de instalações serão realizadas é bem útil ter uma cópia em cache local.

Execute o seguinte comando a partir do Prompt de Comando para fazer o download de todos os possíveis arquivos necessários. Lembre-se de substituir `python-3.8.0.exe` pelo nome real do seu instalador, e de criar layouts nos respectivos diretórios para evitar colisão entre arquivos com o mesmo nome.

```
python-3.8.0.exe /layout [optional target directory]
```

Você também pode especificar a opção `/quiet` para esconder o acompanhamento de progresso.

3.1.5 Modificando uma instalação

Uma vez que o Python foi instalado, você pode adicionar ou remover recursos através da ferramenta Programas e Recursos que é parte do Windows. Selecione o registro do Python e escolha "Uninstall/Change" para abrir o instalador no modo de manutenção.

"Modify" permite que você adicione ou remova recursos modificando as caixas de entrada - caixas de entrada não modificadas não irão instalar ou remover nada. Algumas opções não podem ser modificadas dessa forma, como o diretório de instalação; para modificá-las, você precisará remover e então reinstalar o Python completamente.

"Repair" irá verificar todos os arquivos que devem ser instalados usando as configurações atuais e substituir qualquer um que tiver sido removido ou modificado.

"Uninstall" irá remover o Python completamente, com a exceção do *Inicializador Python para Windows*, que tem seu próprio registro nos Programas e Recursos.

3.2 O pacote Microsoft Store

Novo na versão 3.7.2.

O pacote Microsoft Store é um interpretador Python facilmente instalável criado principalmente para uso interativo, por exemplo, por estudantes.

Para instalar o pacote, certifique-se de que você possui as últimas 10 atualizações do Windows e procure o app "Python 3.8" na Microsoft Store. Certifique-se que o app que você selecionou foi publicado pela Python Software Foundation, e instale-o.

> **Aviso:** Python sempre estará disponível gratuitamente na Microsoft Store. Se você for solicitado a pagar por ele, você não selecionou o pacote correto.

Após a instalação, o Python pode ser inicializado por pesquisa no menu Iniciar. Alternativamente, ele estará disponível a partir de qualquer Prompt de Comando ou sessão PowerShell apenas digitando `python`. Além disso, o pip e a IDLE podem ser usados digitando `pip` ou `idle`. A IDLE também pode ser encontrada no menu Iniciar.

Todos os três comandos também estão disponíveis com sufixos do número da versão, por exemplo, como em `python3.exe` e `python3.x.exe` bem como `python.exe` (onde `3.x` é a versão específica que você quer iniciar, como a 3.8). Abra o "Gerenciar aliases de execução de aplicativo" através do menu Iniciar para selecionar qual versão do Python está associada com cada comando. É recomendado que você se certifique que `pip` e `idle` são consistentes com qualquer versão do `python` que seja selecionada.

Ambientes virtuais podem ser criados com `python -m venv` e ativados e usados normalmente.

Se você instalou outra versão do Python e adicionou ela à sua variável `PATH`, ela estará disponível com `python.exe` ao invés de uma instalada pela Microsoft Store. Para acessar a nova instalação, use `python3.exe` ou `python3.x.exe`.

O instalador `py.exe` irá detectar essa instalação do Python, mas irá preferir instalações feitas pelo instalador tradicional.

Para remover o Python, abra as Configurações e use Aplicativos e Recursos, ou encontre o Python no menu Iniciar e clique com o botão direito para selecionar Desinstalar. A desinstalação irá remover todos os pacotes que você instalou diretamente dentro dessa instalação do Python, mas não irá remover nenhum ambiente virtual.

3.2.1 Problemas Conhecidos

Por causa de restrições nos aplicativos da Microsoft Store, scripts Python podem não ter acesso de escrita completo em locais compartilhados como `TEMP` ou o registro. Ao invés disso, ele irá escrever uma cópia privada. Se seus scripts precisam modificar locais compartilhados, você terá que instalar o instalador completo.

Para obter mais detalhes sobre a base técnica dessas limitações, consulte a documentação da Microsoft sobre aplicativos de confiança total empacotados, atualmente disponíveis em docs.microsoft.com/pt-br/windows/msix/desktop/desktop-to-uwp-behind-the-scenes

3.3 Os pacotes nuget.org

Novo na versão 3.5.2.

O pacote nuget.org é um ambiente Python de tamanho reduzido criado para uso em integração contínua e construção de sistemas que não precisam de uma instalação de Python por todo o sistema da máquina. Enquanto nuget é o "gerenciador de pacotes para .NET", ele também funciona perfeitamente bem para pacotes contendo ferramentas em tempo de construção.

Visite nuget.org para informações mais atualizadas sobre utilização do nuget. A seguir está um sumário que é suficiente para desenvolvedores Python.

A ferramenta de linha de comando `nuget.exe` pode ser baixada diretamente de `https://aka.ms/nugetclidl`, por exemplo, usando o curl ou PowerShell. Com a ferramenta, a versão mais recente do Python para máquinas 64-bit ou 32-bit é instalada usando:

```
nuget.exe install python -ExcludeVersion -OutputDirectory .
nuget.exe install pythonx86 -ExcludeVersion -OutputDirectory .
```

Para selecionar uma versão específica, adicione `-Version 3.x.y`. O diretório de saída pode ser mudado de `.`, e o pacote será instalado em um subdiretório. Por padrão, o subdiretório é nomeado igual ao pacote, e sem a opção `-ExcludeVersion` esse nome irá incluir a versão específica instalada. Dentro do subdiretório está um diretório `tools` que contém a instalação do Python:

```
# Without -ExcludeVersion
> .\python.3.5.2\tools\python.exe -V
Python 3.5.2

# With -ExcludeVersion
> .\python\tools\python.exe -V
Python 3.5.2
```

Em geral, pacotes nuget não são atualizáveis, e novas versões devem ser instaladas lado-a-lado e referenciadas usando o caminho completo. Alternativamente, delete o diretório do pacote manualmente e instale novamente. Muitos sistemas CI irão fazer isso automaticamente se eles não preservam arquivos entre construções de projetos.

Juntamente com o diretório `tools` está o diretório `build\native`. Ele contém um arquivo de propriedades MSBuild `python.props` que pode ser usado em um projeto C++ para referenciar a instalação do Python. Incluir as configurações irá automaticamente usar os cabeçalhos e importar as bibliotecas na sua construção de projeto.

As páginas de informação dos pacotes em nuget.org são www.nuget.org/packages/python para a versão 64-bit e www.nuget.org/packages/pythonx86 para a versão 32-bit.

3.4 O pacote embutível

Novo na versão 3.5.

A distribuição embutida é um arquivo ZIP contendo um ambiente Python mínimo. Ela foi criada para atuar como parte de outra aplicação, ao invés de ser diretamente acessada por usuários finais.

Quando extraída, a distribuição embutida é (quase) completamente isolada do sistema do usuário, incluindo variáveis de ambiente, configurações de registro de sistema, e pacotes instalados. A biblioteca padrão está inclusa como arquivos `.pyc` pré-compilados e otimizados em um ZIP, e `python3.dll`, `python37.dll`, `python.exe` e `pythonw.exe` estão todos disponíveis. Tcl/tk (incluindo todas as dependências, como a Idle), pip e a documentação do Python não estão inclusos.

Nota: A distribuição embutida não inclui o Microsoft C Runtime e é de responsabilidade do instalador da aplicação providenciar isso. O aplicativo de tempo de execução pode já ter sido instalado no sistema de um usuário previamente ou automaticamente via Windows Update, e pode ser detectado procurando por `ucrtbase.dll` no diretório do sistema.

Nota: When running on Windows 7, Python 3.8 requires the KB2533623 update to be installed. The embeddable distribution does not detect this update, and may fail at runtime. Later versions of Windows include this update.

Pacotes de terceiros devem ser instalados pelo instalador da aplicação juntamente com a distribuição embutida. Usar o pip para gerenciar as dependências como em uma instalação regular do Python não é suportado nessa distribuição, apesar de que com algum cuidado pode ser possível incluir e usar o pip para atualizações automáticas. Em geral, pacotes de terceiros devem ser tratados como parte da aplicação ("vendoring") para que o desenvolvedor consiga garantir compatibilidade com versões mais recentes antes de fornecer atualizações para os usuários.

Os dois casos de uso recomendados para essa distribuição são descritos abaixo.

3.4.1 Aplicação Python

Uma aplicação escrita em Python não requer necessariamente que os usuários estejam cientes deste fato. A distribuição embutida pode ser usada neste caso para incluir uma versão privada do pacote de instalação do Python. Dependendo de quão transparente deve ser (ou pelo contrário, quão profissional deve parecer), existem duas opções.

Usar um executável especializado como inicializador requer um pouco de código, mas fornece a experiência mais transparente para os usuários. Com um inicializador personalizado, não existem indicações óbvias que o programa está rodando em Python: ícones podem ser personalizados, informações da empresa e versão podem ser especificadas, e associações de arquivo se comportam de forma apropriada. Na maioria dos casos, um inicializador personalizado deve simplesmente ser capaz de chamar `Py_Main` com uma linha de comando predefinida (hard-coded).

A abordagem mais simples é fornecer um arquivo batch ou um atalho gerado que chama diretamente o `python.exe` ou `pythonw.exe` com os argumentos de linha de comando necessários. Neste caso, a aplicação irá aparecer como Python e não seu nome real, e os usuários podem ter problemas em distinguir ela de outros processos ou associações de arquivo em Python.

Com a abordagem anterior, pacotes devem ser instalados como diretórios juntamente do executável do Python para garantir que estarão disponíveis no caminho do ambiente. Com o instalador especializado, pacotes podem ser armazenados em outras localidades já que há uma oportunidade de especificar o caminho de busca antes de executar a aplicação.

3.4.2 Incorporando Python

Aplicações escritas em código nativo frequentemente precisam de alguma forma de linguagem de script, e a distribuição embutida do Python pode ser usada para esse propósito. Em geral, a maior parte da aplicação é em código nativo, e uma parte irá ou invocar `python.exe` ou usar diretamente `python3.dll`. Para ambos os casos, extrair a distribuição embutida em um subdiretório da instalação da aplicação é suficiente para providenciar um interpretador Python carregável.

Para uso da aplicação, pacotes podem ser instalados em qualquer local já que há uma oportunidade de especificar caminhos de busca antes de inicializar o interpretador. De outra forma, não existem diferenças fundamentais entre usar a distribuição embutida ou a instalação regular.

3.5 Pacotes Alternativos

À parte da distribuição padrão CPython, existem pacotes modificados incluindo funcionalidades adicionais. A seguir está uma lista de versões populares e seus recursos chave:

ActivePython Instalador com compatibilidade multi-plataforma, documentação, PyWin32

Anaconda Módulos científicos populares (como o numpy, scipy e pandas) e o gerenciador de pacotes `conda`.

Canopy Um "ambiente de análise Python compreensivo" com editores e outras ferramentas de desenvolvimento.

WinPython Distribuição específica do Windows com pacotes científicos pré-construídos e ferramentas para construir pacotes.

Note que esses pacotes podem não incluir as últimas versões do Python ou outras bibliotecas, e não são mantidos ou suportados pelo time do núcleo do Python.

3.6 Configurando o Python

Para executar o Python convenientemente de um prompt de comando, você pode considerar mudar algumas variáveis de ambiente padrão do Windows. Ainda que o instalador forneça uma opção para configurar as variáveis PATH e PATHEXT para você, isso só é confiável para uma instalação única e global no sistema. Se você usa regularmente múltiplas versões do Python, considere usar o *Inicializador Python para Windows*.

3.6.1 Excursus: Configurando variáveis de ambiente

O Windows permite que variáveis de ambiente sejam configuradas de forma permanente em ambos os níveis de Usuário e de Sistema, ou temporariamente em um prompt de comando.

Para definir as variáveis de ambiente temporariamente, abra um Prompt de Comando e use o comando **`set`**:

```
C:\>set PATH=C:\Program Files\Python 3.8;%PATH%
C:\>set PYTHONPATH=%PYTHONPATH%;C:\My_python_lib
C:\>python
```

Essas mudanças serão aplicadas em quaisquer comandos posteriores que forem executados neste console, e serão herdadas por quaisquer aplicações iniciadas pelo console.

Incluir o nome da variável com sinais de porcentagem irá expandir o valor existente, permitindo que você adicione seu novo valor ao início ou final. Modificar o `PATH` adicionando o diretório contendo **`python.exe`** ao início é o modo mais comum de garantir que a versão correta do Python seja iniciada.

Para modificar permanentemente as variáveis de ambiente padrão, clique em Iniciar e procure por 'Editar as variáveis de ambiente do sistema', ou abra as propriedades do Sistema, *Configurações avançadas do sistema* e clique no botão *Variáveis de Ambiente*. Neste diálogo, você pode adicionar ou modificar as variáveis de Usuário ou Sistema. Para mudar as variáveis do Sistema, você precisa de acesso não-restrito à sua máquina (isto é, direitos de Administrador).

Nota: O Windows irá concatenar as variáveis de Usuário *após* as variáveis de Sistema, o que pode causar resultados inesperados quando modificando o `PATH`.

A variável *`PYTHONPATH`* é usada por todas as versões do Python 2 e Python 3, então você não deve configurar permanentemente esta variável a menos que ela apenas inclua código que seja compatível com todas as suas versões instaladas do Python.

Ver também:

https://www.microsoft.com/en-us/wdsi/help/folder-variables Variáveis de ambiente no Windows NT

https://technet.microsoft.com/en-us/library/cc754250.aspx O comando SET, para modificar temporariamente as variáveis de ambiente

https://technet.microsoft.com/en-us/library/cc755104.aspx O comando SETX, para modificar permanentemente as variáveis de ambiente

https://support.microsoft.com/pt-br/help/310519/how-to-manage-environment-variables-in-windows-xp Como Gerenciar Variáveis de Ambiente no Windows XP

https://www.chem.gla.ac.uk/~louis/software/faq/q1.html Configurando variáveis de ambiente, Louis J. Farrugia

3.6.2 Encontrando o executável do Python

Alterado na versão 3.5.

Além de usar o registro do menu Iniciar criado automaticamente para o interpretador do Python, você pode querer iniciar o Python no prompt de comando. O instalador possui uma opção para configurar isso pra você.

Na primeira página do instalador, uma opção chamada "Add Python to PATH" pode ser selecionada para que o instalador adicione o local de instalação na sua variável `PATH`. O local da pasta de `Scripts\` também é adicionado. Isso permite que você digite **`python`** para executar o interpretador, e **`pip`** para o instalador de pacotes. Além disso, você pode também executar seus scripts com as opções de linha de comando, veja a documentação *Linha de comando*.

Se você não habilitar essa opção no momento de instalação, você sempre pode re-executar o instalador, selecionar Modify, e habilitá-la. Alternativamente, você pode modificar manualmente a variável `PATH` usando os direcionamentos em *Excursus: Configurando variáveis de ambiente*. Você precisa definir sua variável de ambiente `PATH` para incluir o diretório da sua instalação Python, delimitado por um ponto e vírgula de outras entradas. Uma variável exemplo pode parecer com isso (presumindo que as duas primeiras entradas já existem):

```
C:\WINDOWS\system32;C:\WINDOWS;C:\Program Files\Python 3.8
```

3.7 Modo UTF-8

Novo na versão 3.7.

O Windows ainda usa codificações legadas para a codificação do sistema (a Página de Código ANSI). O Python usa-o para a codificação padrão de arquivos de texto (por exemplo, `locale.getpreferredencoding()`).

Isso pode causar problemas, porque o UTF-8 é amplamente usado na Internet e na maioria dos sistemas Unix, incluindo o WSL (Subsistema Windows para Linux).

Você pode usar o modo UTF-8 para alterar a codificação de texto padrão para UTF-8. Você pode ativar o modo UTF-8 através da opção de linha de comando `-X utf8` ou da variável de ambiente `PYTHONUTF8=1`. Veja *`PYTHONUTF8`* para habilitar o modo UTF-8, e *Excursus: Configurando variáveis de ambiente* para saber como modificar variáveis de ambiente.

Quando o modo UTF-8 está habilitado:

- `locale.getpreferredencoding()` retorna `'UTF-8'` em vez da codificação do sistema. Esta função é usada para a codificação de texto padrão em muitos locais, incluindo `open()`, `Popen`, `Path.read_text()`, etc.
- `sys.stdin`, `sys.stdout` e `sys.stderr`, todos usam UTF-8 como codificação de texto.
- Você ainda pode usar a codificação do sistema através do codec "mbcs".

Observe que adicionar `PYTHONUTF8=1` às variáveis de ambiente padrão afetará todas as aplicações Python 3.7+ em seu sistema. Se você tiver aplicações Python 3.7+ que dependem da codificação do sistema legado, é recomendável definir a variável de ambiente temporariamente ou usar a opção de linha de comando `-X utf8`.

Nota: Mesmo quando o modo UTF-8 está desativado, o Python usa o UTF-8 por padrão no Windows para:

- E/S do console, incluindo E/S padrão (consulte **PEP 528** para detalhes).
- A codificação do sistema de arquivos (consulte **PEP 529** para detalhes)

3.8 Inicializador Python para Windows

Novo na versão 3.3.

O inicializador Python para Windows é um utilitário que auxilia na localização e execução de diferentes versões do Python. Ele permite que scripts (ou a linha de comando) indiquem uma preferência por uma versão do Python específica, e irá localizar e executar essa versão.

Ao contrário da variável `PATH`, o inicializador irá corretamente selecionar a versão mais apropriada do Python. Ele irá preferir instalações por usuário ao invés de instalações globais no sistema, e ordenará por versão da linguagem ao invés de usar a versão instalada mais recentemente.

O inicializador foi originalmente especificado na **PEP 397**.

3.8.1 Começando

Pela linha de comando

Alterado na versão 3.6.

Instalações globais no sistema do Python 3.3 ou posterior irão colocar o inicializador no seu `PATH`. O inicializador é compatível com todas as versões disponíveis do Python, então não importa qual versão está instalada. Para verificar se o inicializador está disponível, execute o seguinte comando no Prompt de Comando:

```
py
```

Você deve perceber que a última versão do Python que você tem é iniciada - ela pode ser fechada normalmente, e qualquer argumento da linha de comando adicional especificado será enviado diretamente para o Python.

Se você tem múltiplas versões do Python instaladas (por exemplo, 2.7 e 3.8) você deve ter notado que o Python 3.8 foi iniciado - para iniciar o Python 2.7, use o comando:

```
py -2.7
```

Se você quer a versão mais recente do Python 2.x que você tem instalada, tente o comando:

```
py -2
```

Você deve perceber que a versão mais recente do Python 2.x iniciou.

Se você ver o seguinte erro, você não tem o inicializador instalado:

```
'py' is not recognized as an internal or external command,
operable program or batch file.
```

Instalações do Python por usuário não permitem adicionar o inicializador ao `PATH` a não ser que a opção seja selecionada na instalação.

Ambientes virtuais

Novo na versão 3.5.

Se o inicializador é executado sem versão explícita do Python especificada, e um ambiente virtual (criado com o módulo da biblioteca padrão `venv` ou da ferramenta externa `virtualenv`) está ativo, o inicializador irá executar o interpretador do ambiente virtual ao invés do global. Para executar o interpretador global, ou desative o ambiente virtual, ou explicitamente especifique a versão global do Python.

Por um script

Vamos criar um script teste de Python - crie um arquivo chamado `hello.py` com os seguintes conteúdos:

```
#! python
import sys
sys.stdout.write("hello from Python %s\n" % (sys.version,))
```

A partir do diretório onde hello.py está, execute o comando:

```
py hello.py
```

Você deve notar que o número da versão da sua instalação mais recente do Python 2.x é exibido. Agora tente mudar a primeira linha para ser:

```
#! python3
```

Re-executar o comando agora deve exibir informações da última versão do Python 3.x. Como nos exemplos da linha de comando acima, você pode especificar um qualificador de versão mais explícito. Assumindo que você tem o Python 2.6 instalado, tente mudar a primeira linha para `#! python2.6` e você deve ver as informações da versão 2.6 sendo exibidas.

Note que diferentemente do uso interativo, um simples "python" irá usar a última versão do Python 2.x que você tem instalada. Isso é para retrocompatibilidade e para compatibilidade com Unix, onde o comando `python` tipicamente se refere ao Python 2.

Por associação de arquivos

O inicializador deve ter sido associado com arquivos Python (isto é, arquivos `.py`, `.pyw`, `.pyc`) quando foi instalado. Isso significa que quando você clica duas vezes em um desses arquivos a partir do Explorador de Arquivos do Windows o inicializador será usado, e assim você pode usar as mesmas facilidades descritas acima para que o script especifique qual versão deve ser usada.

O benefício chave disso é que um único inicializador pode suportar múltiplas versões do Python ao mesmo tempo dependendo dos conteúdos da primeira linha.

3.8.2 Linhas Shebang

Se a primeira linha de um arquivo de script começa com `#!`, ela é conhecida como linha "shebang". Linux e outros tipos de sistemas operacionais Unix têm suporte nativo para essas linhas e elas são comumente usadas nesses sistemas para indicar como um script deve ser executado. Esse inicializador permite que as mesmas facilidades sejam usadas com scripts Python no Windows e os exemplos acima demonstram seu uso.

Para permitir que linhas shebang em scripts Python sejam portáveis entre Unix e Windows, este inicializador suporta um número de comandos 'virtuais' para especificar qual interpretador deve ser usado. Os comandos virtuais suportados são:

- `/usr/bin/env python`
- `/usr/bin/python`
- `/usr/local/bin/python`
- `python`

Por exemplo, se a primeira linha do seu script começa com

```
#! /usr/bin/python
```

O Python padrão será localizado e utilizado. Como muitos scripts Python escritos para funcionar no Unix já terão essa linha, você deve perceber que esses scripts podem ser usados pelo inicializador sem modificação. Se você está

escrevendo um novo script no Windows que você espera que seja útil no Unix, você deve usar uma dessas linhas shebang começando com `/usr`.

Qualquer um dos comandos virtuais acima pode ser sufixado com uma versão explícita (ou apenas a maior versão, ou a maior e a menor versão). Além disso, a versão de 32-bit pode ser solicitada adicionando "-32" após a menor versão. Isto é, `/usr/bin/python2.7-32` irá solicitar o uso do python 2.7 32-bit.

Novo na versão 3.7: A partir do inicializador do python 3.7 é possível solicitar a versão 64-bit adicionando o sufixo "-64". Além disso é possível especificar uma versão maior e arquitetura sem a menor (isto é, `/usr/bin/python3-64`).

A forma de linha shebang `/usr/bin/env` tem uma propriedade especial adicional. Antes de procurar pela versão dos interpretadores de Python instalados, essa forma irá pesquisar no `PATH` por um executável Python. Isso corresponde ao comportamento do programa `env` do Unix, que realiza a pesquisa no `PATH`.

3.8.3 Argumentos em linhas shebang

As linhas shebang também podem especificar opções adicionais a serem passadas ao interpretador Python. Por exemplo, se você tem uma linha shebang:

```
#! /usr/bin/python -v
```

O Python será iniciado com a opção `-v`

3.8.4 Personalização

Personalização via arquivos INI

Dois arquivos .ini serão pesquisados pelo inicializador - `py.ini` no diretório "dados de aplicativos" do usuário atual (isto é, o diretório retornado ao chamar a função do Windows `SHGetFolderPath` com `CSIDL_LOCAL_APPDATA`) e `py.ini` no mesmo diretório que o inicializador. Os mesmos arquivos .ini são usados para ambas a versão 'console' do inicializador (isto é, py.exe) e a versão 'windows' (isto é, pyw.exe).

Personalização especificada no "diretório da aplicação" terão precedência sobre àquela especificada junto do executável, portanto um usuário, que pode não ter acesso de escrita ao arquivo .ini junto do inicializador, pode sobrescrever comandos naquele arquivo .ini global.

Personalizando versões padrão do Python

Em alguns casos, um qualificador de versão pode ser incluído em um comando para ditar qual versão do Python deve ser usada pelo comando. Um qualificador de versão começa com um número maior de versão e pode opcionalmente ser seguido por um ponto ('.') e uma especificação de versão menor. Além disso, é possível especificar se uma implementação de 32 ou 64 bit deve ser solicitada adicionando "-32" ou "-64".

Por exemplo, uma linha shebang `#!python` não tem qualificador de versão, enquanto `#!python3` tem um qualificador de versão que especifica apenas uma versão maior.

Se nenhum qualificador de versão é encontrado em um comando, a variável de ambiente `PY_PYTHON` pode ser definida para especificar o qualificador de versão padrão. Se ela não está definida, o padrão é "3". A variável pode especificar qualquer valor que pode ser passado para a linha de comando, como "3", "3.7", "3.7-32" ou "3.7-64". (Note que a opção "-64" está disponível apenas com o inicializador incluso no Python 3.7 ou mais recente.)

Se nenhum qualificador de versão menor é encontrado, a variável de ambiente `PY_PYTHON{major}` (onde `{major}` é o qualificador da versão maior atual, como determinado acima) pode ser definida para especificar a versão completa. Se nenhuma opção é encontrada, o inicializador irá enumerar as versões do Python instaladas e usar a última versão menor encontrada como versão maior, o que é provavelmente, ainda que não garantido, a versão instalada mais recentemente naquela família.

Em um Windows 64-bit com ambas as implementações de 32-bit e 64-bit da mesma (maior.menor) versão do Python instaladas, a versão 64-bit sempre será preferida. Isso será verdadeiro para ambas as implementações de 32-bit

e 64-bit do inicializador - um inicializador 32-bit irá preferir executar uma instalação 64-bit do Python da versão especificada, se disponível. Isso é para que o comportamento do inicializador possa ser previsto sabendo apenas quais versões estão instaladas no PC e sem considerar a ordem com que elas foram instaladas (isto é, sem saber se a última a ser instalada foi a versão 32 ou 64-bit do Python e do instalador correspondente). Como observado acima, um sufixo opcional "-32" ou "-64" pode ser usado como especificador de versão para mudar esse comportamento.

Exemplos:

- Se nenhuma opção relevante for definida, os comandos `python` e `python2` irão usar a última versão instalada do Python 2.x e o comando `python3` irá usar o último Python 3.x instalado.
- Os comandos `python3.1` e `python2.7` não irão consultar nenhuma opção já que as versões estão completamente especificadas.
- Se `PY_PYTHON=3`, os comandos `python` e `python3` irão ambos usar a última versão do Python 3 instalada.
- Se `PY_PYTHON=3.1-32`, o comando `python` irá usar a implementação 32-bit do 3.1 enquanto o comando `python3` irá usar o último Python instalado (PY_PYTHON não foi considerado porque uma versão maior foi especificada).
- Se `PY_PYTHON=3` e `PY_PYTHON3=3.1`, os comandos `python` e `python3` irão ambos usar especificamente 3.1.

Em adição às variáveis de ambiente, as mesmas configurações podem ser feitas no arquivo .INI usado pelo inicializador. A seção nos arquivos INI é chamada `[defaults]` e o nome da chave será o mesmo que as variáveis de ambiente sem o prefixo `PY_` (e observe que os nomes das chaves no arquivo INI não diferem maiúsculas e minúsculas). Os conteúdos de uma variável de ambiente irão sobrescrever coisas especificadas em um arquivo INI.

Por exemplo:

- Configurar `PY_PYTHON=3.1` é o equivalente ao arquivo INI contendo:

```
[defaults]
python=3.1
```

- Configurar `PY_PYTHON=3` e `PY_PYTHON3=3.1` é o equivalente ao arquivo INI contendo:

```
[defaults]
python=3
python3=3.1
```

3.8.5 Diagnósticos

Se uma variável de ambiente `PYLAUNCH_DEBUG` é configurada (para qualquer valor), o inicializador irá exibir a informação de diagnóstico do stderr (isto é, no console). Enquanto essa informação consegue ser simultaneamente verbosa *e* concisa, ela deve permitir que você veja quais versões do Python foram localizadas, porquê uma versão particular foi escolhida e qual linha de comando exatamente foi usada para executar o Python alvo.

3.9 Encontrando módulos

O Python geralmente armazena sua biblioteca (e assim sua pasta de site-packages) no diretório de instalação. Então, se você instalou o Python em `C:\Python\`, a biblioteca padrão irá residir em `C:\Python\Lib\` e módulos de terceiros serão armazenados em `C:\Python\Lib\site-packages\`.

Para completamente sobrescrever `sys.path´, crie um arquivo `._pth" com o mesmo nome que a DLL (`python37._pth`) ou o executável (`python._pth`) e especifique uma linha para cada caminho a adicionar ao `sys.path`. O arquivo baseado no nome da DLL sobrescreve o arquivo baseado no executável, o que permite que caminhos sejam restritos para qualquer programa carregando o tempo de execução, se desejado.

Quando o arquivo existe, todos os registros e variáveis de ambiente são ignorados, o modo isolado é ativado, e `site` não é importado a menos que uma linha do arquivo especifique `import site`. Caminhos em branco e linhas

começando com # são ignorados. Cada caminho pode ser absoluto ou relativo ao local do arquivo. Instruções de importação diferente de site não são permitidas, e código arbitrário não pode ser especificado.

Note que arquivos ._pth (sem o sublinhado no início) serão processados normalmente pelo módulo site quando import site tiver sido especificado.

Quando nenhum arquivo ._pth é encontrado, assim é como sys.path é populado no Windows:

- Uma entrada em branco é adicionada ao início, que corresponde ao diretório atual.
- Se a variável de ambiente *PYTHONPATH* existe, como descrito em *Variáveis de ambiente*, suas entradas são adicionadas em seguida. Note que no Windows, caminhos nessa variável devem ser separados por ponto e vírgula, para distinguir eles dos dois pontos usados nos identificadores de drivers (C:\ etc.).
- "Caminhos da aplicação" adicionais podem ser adicionados ao registro como subchaves de \SOFTWARE\Python\PythonCore{version}\PythonPath sob ambas HKEY_CURRENT_USER e HKEY_LOCAL_MACHINE. Subchaves que possuem string de caminhos delimitados por ponto e vírgula como seu valor padrão farão com que cada caminho seja adicionado ao sys.path. (Note que todos os instaladores conhecidos usam apenas HKLM, portanto HKCU está tipicamente vazio.)
- Se a variável de ambiente *PYTHONHOME* está definida, ela é presumida como "Python Home". Caso contrário, o caminho do principal executável do Python é usado para localizar um "arquivo de referência" (ou Lib\os.py ou pythonXY.zip) para deduzir o "Python Home". Se um Python Home é encontrado, os subdiretórios relevantes adicionados ao sys.path (Lib, plat-win, etc) são baseados naquela pasta. Se não, o caminho núcleo do Python é construído a partir do PythonPath armazenado no registro.
- Se o Python Home não puder ser localizado, nenhum *PYTHONPATH* está especificado no ambiente, e nenhuma entrada de registro pôde ser encontrada, um caminho padrão com entradas relativas é usado (por exemplo, .\Lib;.\plat-win, etc).

Se um arquivo pyvenv.cfg for encontrado juntamente com o principal executável ou no diretório um nível acima do executável, as seguintes variações se aplicam:

- Se home é um caminho absoluto e *PYTHONHOME* não está definido, o caminho é usado ao invés do caminho ao principal executável quando deduzindo a localização do Home.

O resultado final de tudo isso é:

- Quando executando python.exe, ou qualquer outro .exe no diretório principal do Python (ou uma versão instalada, ou diretamente do diretório PCbuild), o caminho núcleo é deduzido, e os caminhos núcleo no registro são ignorados. Outros "caminhos da aplicação" no registro são sempre lidos.
- Quando Python é hospedado em outro .exe (diretório diferente, embutido via COM, etc), o "Python Home" não será deduzido, então o caminho núcleo do registro é usado. Outros "caminhos da aplicação" no registro sempre são lidos.
- Se o Python não puder encontrar seu Home e não existem valores no registro (.exe imutáveis, algumas configurações de instalação bem estranhas) você recebe um caminho com alguns caminhos padrão, porém relativos.

Para aqueles que querem empacotar o Python em suas aplicações ou distribuições, o seguinte conselho irá prevenir conflitos com outras instalações:

- Inclua um arquivo ._pth juntamente do executável contendo os diretórios a serem incluídos. Isso irá ignorar caminhos listados no registro e variáveis de ambiente, e também ignorar sites a não ser que import site seja listado.
- Se você está carregando o python3.dll ou python37.dll no seu próprio executável, explicitamente chame Py_SetPath() ou (pelo menos) Py_SetProgramName() antes de Py_Initialize().
- Limpe e/ou sobrescreva *PYTHONPATH* e defina *PYTHONHOME* antes de iniciar o python.exe a partir da sua aplicação.
- Se você não puder usar as sugestões anteriores (por exemplo, você é uma distribuição que permite que as pessoas executem o arquivo python.exe diretamente), certifique-se de que o arquivo de referência (Lib\os.py) existe no seu diretório de instalação. (Note que ele não será detectado dentro de um arquivo ZIP, mas um arquivo ZIP corretamente nomeado será detectado ao invés disso.)

Isso irá garantir que seus arquivos em uma instalação global no sistema não terão precedência sobre uma cópia da biblioteca padrão empacotada com a sua aplicação. Caso contrário, seus usuários podem experienciar problemas usando sua aplicação. Note que a primeira sugestão é a melhor, pois as outras podem ainda estar suscetíveis a caminhos não-padrão no registro e no site-packages do usuário.

Alterado na versão 3.6:

- Adiciona suporte a arquivos `._pth` e remove a opção `applocal` do `pyvenv.cfg`.
- Adiciona `pythonXX.zip` como uma possível referência quando diretamente adjacente ao executável.

Obsoleto desde a versão 3.6: Módulos especificados no registro sob `Modules` (não `PythonPath`) podem ser importados por `importlib.machinery.WindowsRegistryFinder`. Este localizador está ativo no Windows no 3.6.0 e anteriores, mas pode precisar ser explicitamente adicionado ao `sys.meta_path` no futuro.

3.10 Módulos adicionais

Mesmo que o Python tenha como objetivo ser portável através de todas as plataformas, existem recursos que são únicos para o Windows. Alguns módulos, em ambas as bibliotecas padrão e externa, e trechos de código existem para usar esses recursos.

Os módulos padrão específicos para Windows estão documentados em mswin-specific-services.

3.10.1 PyWin32

O módulo PyWin32 de Mark Hammond é uma coleção de módulos para suporte avançado específico para Windows. Isso inclui utilitários para:

- Component Object Model (COM)
- Chamadas à API Win32
- Registro
- Log de Eventos
- Microsoft Foundation Classes (MFC) interface de usuário

PythonWin é uma aplicação MFC de exemplo enviada com o PyWin32. É uma IDE embutível com um depurador embutido.

Ver também:

Win32 How Do I...? por Tim Golden

Python and COM by David and Paul Boddie

3.10.2 cx_Freeze

cx_Freeze é uma extensão `distutils` (veja extending-distutils) que encapsula scripts Python em programas executáveis do Windows (arquivos `*.exe`). Quando você tiver feito isso, você pode distribuir sua aplicação sem solicitar que os seus usuários instalem o Python.

3.10.3 WConio

Dado que a camada de manipulação de terminal avançada do Python, `curses`, é restrita a sistemas tipo Unix, existe uma biblioteca exclusiva para o Windows: Windows Console I/O para Python.

WConio é um encapsulador para o `CONIO.H` do Turbo-C usado para criar texto sob interfaces de usuário.

3.11 Compilando Python no Windows

Se você quer compilar o CPython por conta própria, a primeira coisa que você precisa ter é a fonte. Você pode fazer o download tanto da fonte da última versão quanto pegar um novo checkout.

A árvore de fontes contém uma solução de construção e arquivos de projeto para o Microsoft Visual Studio 2015, que é o compilador usado para construir as versões do Python oficiais. Esses arquivos estão no diretório `PCbuild`.

Confira o `PCbuild/readme.txt` para informações gerais sobre o processo de construção.

Para módulos de extensão, consulte building-on-windows.

Ver também:

Python + Windows + distutils + SWIG + gcc MinGW ou "Creating Python extensions in C/C++ with SWIG and compiling them with MinGW gcc under Windows" ou "Installing Python extension with distutils and without Microsoft Visual C++" por Sébastien Sauvage, 2003

3.12 Outras plataformas

Com o desenvolvimento do Python em andamento, algumas plataformas que costumavam ser suportadas anteriormente não são mais suportadas (devido à falta de usuário ou desenvolvedores). Confira a **PEP 11** para detalhes de todas as plataformas não suportadas.

- Windows CE ainda é suportado.
- O instalador Cygwin também oferece instalação do interpretador do Python (cf. Cygwin package source, Maintainer releases)

Veja Python for Windows para informação detalhada sobre as plataformas com instaladores pré-compilados.

CAPÍTULO 4

Utilizando Python em um Mac

Autor Bob Savage <bobsavage@mac.com>

O Python em um Macintosh executando o Mac OS X é, em princípio, muito semelhante ao Python em qualquer outra plataforma Unix, mas há vários recursos adicionais, como o IDE e o Gerenciador de Pacotes, que merecem destaque.

4.1 Obtendo e instalando MacPython

O Mac OS X 10.8 vem com o Python 2.7 pré-instalado pela Apple. Se desejar, você está convidado a instalar a versão mais recente do Python 3 no site do Python (https://www.python.org). Uma versão atual do "binário universal" do Python, que funciona nativamente nas novas CPUs Intel e PPC herdadas do Mac, está disponível lá.

O que você obtém após a instalação é uma série de coisas:

- A pasta `Python 3.8` na sua pasta `Applications`. Aqui você encontra o IDLE, o ambiente de desenvolvimento que é parte padrão das distribuições oficiais do Python; e PythonLauncher, que lida com scripts de Python clicando duas vezes no Finder.
- Um framework `/Library/Frameworks/Python.framework`, que inclui o executável e as bibliotecas do Python. O instalador adiciona esse local ao seu caminho do console. Para desinstalar o MacPython, você pode simplesmente remover essas três coisas. Um link simbólico para o executável Python é colocado em /usr/local/bin/.

A compilação do Python fornecida pela Apple é instalada em `/System/Library/Frameworks/Python.framework` e `/usr/bin/python`, respectivamente. Você nunca deve modificá-las ou excluí-las, pois elas são controladas pela Apple e são usadas por software da Apple ou de terceiros. Lembre-se de que, se você optar por instalar uma versão mais recente do Python a partir do python.org, terá duas instalações Python diferentes, mas funcionais, no seu computador, por isso será importante que seus caminhos e usos sejam consistentes com o que você deseja fazer.

O IDLE inclui um menu de ajuda que permite acessar a documentação do Python. Se você é completamente novo no Python, comece a ler a introdução do tutorial nesse documento.

Se você está familiarizado com o Python em outras plataformas Unix, leia a seção sobre a execução de scripts Python no shell do Unix.

4.1.1 Como executar um script Python

A melhor maneira de você começar o uso do Python no Mac OS X é através do ambiente de desenvolvimento integrado IDLE, consulte a seção *A IDE* e use o menu Help quando o IDE estiver em execução.

Se você deseja executar scripts Python na linha de comando da janela Terminal ou no Finder, primeiro precisa de um editor para criar seu script. O Mac OS X vem com vários editores de linha de comando padrão do Unix, entre os quais **vim** e **emacs**. Se você deseja um editor mais ao estilo do Mac, **BBEdit** ou **TextWrangler** da Bare Bones Software (consulte http://www.barebones.com/products/bbedit/index.html) são boas escolhas , como é **TextMate** (consulte https://macromates.com/). Outros editores incluem **Gvim** (http://macvim-dev.github.io/macvim/) e **Aquamacs** (http://aquamacs.org/).

Para executar seu script a partir da janela do Terminal, você deve se certificar de que `/usr/local/bin` esteja no seu caminho de pesquisa do shell.

Para executar seu script no Finder, você tem duas opções:

- Arrastá-lo para o **PythonLauncher**
- Selecionar **PythonLauncher** como aplicação padrão para abrir seu script (ou qualquer script .py) através da janela de Informações do Finder e clique duas vezes nele. **PythonLauncher** possui várias preferências para controlar como o script é iniciado. Arrastar com opções permite alterar esses itens para uma chamada ou usar o menu Preferências para alterar as coisas globalmente.

4.1.2 Executando scripts como uma GUI

Nas versões mais antigas do Python, há uma peculiaridade do Mac OS X que você precisa conhecer: os programas que conversam com o gerenciador de janelas Aqua (em outras palavras, qualquer coisa que tenha uma GUI) precisam ser executados de uma maneira especial. Use **pythonw** em vez de **python** para iniciar esses scripts.

Com o Python 3.8, você pode usar **python** ou **pythonw**.

4.1.3 Configuração

O Python no OS X honra todas as variáveis de ambiente padrão do Unix, como *PYTHONPATH*, mas definir essas variáveis para programas iniciados no Finder não é padrão, pois o Finder não lê o seu `.profile` ou `.cshrc` na inicialização. Você precisa criar um arquivo `~/.MacOSX/environment.plist`. Consulte o Documento Técnico da Apple QA1067 para obter detalhes.

Para obter mais informações sobre a instalação de pacotes Python no MacPython, consulte a seção *Instalando pacotes adicionais ao python*.

4.2 A IDE

O MacPython é fornecido com o ambiente de desenvolvimento IDLE padrão. Uma boa introdução ao uso do IDLE pode ser encontrada em http://www.hashcollision.org/hkn/python/idle_intro/index.html.

4.3 Instalando pacotes adicionais ao python

Existem vários métodos para instalar pacotes Python adicionais:

- Os pacotes podem ser instalados através do modo distutils padrão do Python (`python setup.py install`).
- Muitos pacotes também podem ser instalados através da extensão **`setuptools`** ou do wrapper **`pip`**. Consulte https://pip.pypa.io/.

4.4 Programação de GUI no Mac

Existem várias opções para criar aplicações GUI no Mac com Python.

PyObjC é uma ligação do Python para o framework Objective-C/Cocoa da Apple, que é a base do desenvolvimento mais moderno do Mac. Informações sobre PyObjC estão disponíveis em https://pypi.org/project/pyobjc/.

O kit de ferramentas de GUI padrão do Python é `tkinter`, baseado no kit de ferramentas plataforma cruzada Tk (https://www.tcl.tk). Uma versão nativa do Aqua do Tk é fornecida com o OS X da Apple, e a versão mais recente pode ser baixada e instalada em https://www.activestate.com; também pode ser construído a partir do código-fonte.

wxPython é outro popular kit de ferramentas de GUI multiplataforma que funciona nativamente no Mac OS X. Pacotes e documentação estão disponíveis em https://www.wxpython.org.

PyQt é outro popular kit de ferramentas de GUI multiplataforma que funciona nativamente no Mac OS X. Mais informações podem ser encontradas em https://riverbankcomputing.com/software/pyqt/intro.

4.5 Distribuindo aplicações Python no Mac

A ferramenta padrão para implantar aplicações Python independentes no Mac é **`py2app`**. Mais informações sobre a instalação e o uso do py2app podem ser encontradas em http://undefined.org/python/#py2app.

4.6 Outros recursos

A lista de discussão do MacPython é um excelente recurso de suporte para usuários e desenvolvedores de Python no Mac:

https://www.python.org/community/sigs/current/pythonmac-sig/

Outro recurso útil é o wiki do MacPython:

https://wiki.python.org/moin/MacPython

CAPÍTULO 5

Editores e IDEs

Há um grande número de IDEs que suportam a linguagem de programação Python. Vários editores e IDEs provêem destaques coloridos de sintaxe, ferramentas de depuração, e checagem do código frente à **PEP 8**.

Por favor, vá para Python Editors and Integrated Development Environments para obter uma lista completa.

APÊNDICE A

Glossário

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

. . . Pode se referir a:

- 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.
- A constante embutida `Ellipsis`.

2to3 Uma ferramenta que tenta converter código Python 2.x em código Python 3.x tratando a maioria das incompatibilidades que podem ser 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 bases 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). CBAs introduzem subclasses virtuais, classes que não herdam de uma classe mas ainda são reconhecidas por `isinstance()` e `issubclass()`; veja a documentação do módulo `abc`. Python vem com muitas CBAs embutidas 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 CBAs com o módulo `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 *dica de tipo*.

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 `__annotations__` de módulos, classes e funções, respectivamente.

Veja *anotação de variável*, *anotação de função*, **PEP 484** e **PEP 526**, que descrevem esta funcionalidade.

argumento Um valor passado para uma *função* (ou *método*) ao chamar a função. Existem dois tipos de argumento:

- *argumento nomeado*: um argumento precedido por um identificador (por exemplo, `name=`) 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 no FAQ sobre a diferença entre argumentos e parâmetros e **PEP 362**.

gerenciador de contexto assíncrono Um objeto que controla o ambiente visto numa instrução `async with` por meio da definição dos métodos `__aenter__()` e `__aexit__()`. Introduzido pela **PEP 492**.

gerador assíncrono Uma função que retorna um *iterador gerador assíncrono*. É parecida com uma função de corrotina definida com `async def` exceto pelo fato de conter instruções `yield` para produzir uma série de valores que podem ser usados em um laço `async for`.

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 as instruções `async for` e `async with`.

iterador gerador assíncrono Um objeto criado por uma função *geradora assíncrona*.

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

Cada `yield` suspende temporariamente o processamento, lembrando o estado de execução do local (incluindo variáveis locais e instruções `try` pendentes). Quando o *iterador gerador assíncrono* é efetivamente retomado com outro aguardável retornado 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 *corrotina* 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 objeto arquivo capaz de ler e gravar em objetos `str`.

objeto byte ou similar Um objeto com suporte ao o bufferobjects e que pode exportar um buffer C *contíguo*. Isso inclui todos os objetos `bytes`, `bytearray` e `array.array`, além de muitos objetos `memoryview` comuns. 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 e 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 O código-fonte Python é compilado para bytecode, a representação interna de um programa em Python no interpretador CPython. O bytecode também é mantido em cache em arquivos `.pyc` e `.pyo`, de forma que executar um mesmo arquivo é mais rápido na segunda vez (a recompilação dos fontes para bytecode não é necessária). Esta "linguagem intermediária" é adequada para execução em uma *máquina virtual*, que executa o código de máquina correspondente para cada bytecode. Tenha em mente que não se espera que bytecodes sejam executados entre máquinas virtuais Python diferentes, nem que se mantenham estáveis entre versões de Python.

Uma lista de instruções bytecode pode ser encontrada na documentação para o módulo dis.

função de retorno Também conhecida como callback, é uma função sub-rotina que é passada como um argumento a ser executado em algum ponto no futuro.

classe Um modelo para criação de objetos definidos pelo usuário. Definições de classe normalmente contém definições de métodos que operam sobre instâncias da classe.

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 Uma extensão ao familiar sistema de números reais em que todos os números são expressos como uma soma de uma parte real e uma parte imaginária. Números imaginários são múltiplos reais da unidade imaginária (a raiz quadrada de `-1`), normalmente escrita como `i` em matemática ou `j` em engenharia. O Python tem suporte nativo para números complexos, que são escritos com esta última notação; a parte imaginária escrita com um sufixo `j`, p.ex., `3+1j`. Para ter acesso aos equivalentes para números complexos do módulo `math`, utilize `cmath`. O uso de números complexos é uma funcionalidade matemática bastante avançada. Se você não sabe se irá precisar deles, é quase certo que você pode ignorá-los sem problemas.

gerenciador de contexto Um objeto que controla o ambiente visto numa instrução `with` por meio da definição dos métodos `__enter__()` e `__exit__()`. Veja **PEP 343**.

variável de contexto Uma variável que pode ter valores diferentes, dependendo do seu contexto. Isso é semelhante ao armazenamento local de threads, no qual cada thread pode ter um valor diferente para uma variável. No entanto, com variáveis de contexto, pode haver vários contextos em uma thread 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 vetores unidimensionais, os itens devem ser dispostos na memória próximos um do outro, em ordem crescente de índices, começando do zero. Em vetores multidimensionais contíguos C, o último índice varia mais rapidamente ao visitar itens em ordem de endereço de memória. No entanto, nos vetores contíguos do Fortran, o primeiro índice varia mais rapidamente.

corrotina Corrotinas são uma forma mais generalizada de sub-rotinas. Sub-rotinas tem a entrada iniciada em um ponto, e a saída em outro ponto. Corrotinas podem entrar, sair, e continuar em muitos pontos diferentes. Elas podem ser implementadas com a instrução `async def`. Veja também **PEP 492**.

função de corrotina Uma função que retorna um objeto do tipo *corrotina*. Uma função de corrotina 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 A implementação canônica da linguagem de programação Python, como disponibilizada pelo python.org. O termo "CPython" é usado quando necessário distinguir esta implementação de outras como Jython ou 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 definições de função e definições de classe para obter mais informações sobre decoradores.

descritor Qualquer objeto que define os métodos `__get__()`, `__set__()` ou `__delete__()`. Quando um atributo de classe é um descritor, seu comportamento de associação especial é acionado no acesso a um atributo. Normalmente, ao se utilizar *a.b* para se obter, definir ou excluir, um atributo dispara uma busca no objeto chamado *b* no dicionário de classe de *a*, mas se *b* for um descritor, o respectivo método descritor é chamado. Compreender descritores é a chave para um profundo entendimento de Python pois eles são a base de muitas funcionalidades incluindo funções, métodos, propriedades, métodos de classe, métodos estáticos e referências para superclasses.

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

dicionário Um vetor 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.

compreensão de dicionário Uma maneira compacta de processar todos ou parte dos elementos de um iterável e retornar um dicionário com os resultados. `results = {n: n ** 2 for n in range(10)}` gera um dicionário contendo a chave `n` mapeada para o valor `n ** 2`. Veja comprehensions.

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

docstring Abreviatura de "documentation string" (string de documentação). 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.

tipagem pato Também conhecida como *duck-typing*, é um estilo de programação que não verifica o tipo do objeto para determinar se ele possui a interface correta; em vez disso, o método ou atributo é simplesmente chamado ou utilizado ("Se se parece com um pato e grasna como um pato, então deve ser um pato.") Enfatizando interfaces ao invés de tipos específicos, o código bem desenvolvido aprimora sua flexibilidade por permitir substituição polimórfica. Tipagem pato evita necessidade de testes que usem `type()` ou `isinstance()`. (Note, porém, que a tipagem pato pode ser complementada com o uso de *classes base abstratas*.) Ao invés disso, são normalmente empregados testes `hasattr()` ou programação *EAFP*.

EAFP Iniciais da expressão em inglês "easier to ask for forgiveness than permission" que significa "é mais fácil pedir perdão que permissão". Este estilo de codificação comum no Python presume a existência de chaves ou atributos válidos e captura exceções caso essa premissa se prove falsa. Este estilo limpo e rápido se caracteriza pela presença de várias instruções `try` e `except`. A técnica diverge do estilo *LBYL*, comum em outras linguagens como C, por exemplo.

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 *instruções*, as 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 do 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**.

objeto arquivo 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, soquetes, pipes, etc.). Objetos arquivo também são chamados de *objetos arquivo ou similares* ou *fluxos*.

Atualmente há três categorias de objetos arquivo: *arquivos binários* brutos, *arquivos binários* em buffer e *arquivos textos*. Suas interfaces estão definidas no módulo `io`. A forma canônica para criar um objeto arquivo é usando a função `open()`.

objeto arquivo ou similar Um sinônimo do termo *objeto arquivo*.

localizador Um objeto que tenta encontrar o *carregador* para um módulo que está sendo importado.

Desde o Python 3.3, existem dois tipos de localizador: *localizadores de metacaminho* para uso com `sys.meta_path`, e *localizadores de entrada de caminho* para uso com `sys.path_hooks`.

Veja **PEP 302**, **PEP 420** e **PEP 451** para mais informações.

divisão pelo piso Divisão matemática que arredonda para baixo para o inteiro mais próximo. O operador de divisão pelo piso é `//`. Por exemplo, a expressão `11 // 4` retorna o valor `2` ao invés de `2.75`, que seria retornado pela divisão de ponto flutuante. Note que `(-11) // 4` é `-3` porque é `-2.75` arredondado *para baixo*. Consulte a **PEP 238**.

função Uma série de instruções que retorna algum valor para um chamador. Também pode ser passado zero ou mais *argumentos* que podem ser usados na execução do corpo. Veja também *parâmetro*, *método* e a seção function.

anotação de função Uma *anotação* de um parâmetro de função ou valor de retorno.

Anotações de função são comumente usados por *dicas de tipo*: 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 função é explicada na seção function.

Veja *anotação de variável* e **PEP 484**, que descrevem essa funcionalidade.

__future__ Um pseudo-módulo o qual os programadores podem usar para habilitar novas funcionalidades da linguagem que não são compatíveis com o interpretador atual.

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)
```

coleta de lixo Também conhecido como *garbage collection*, é 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 Uma função que retorna um *iterador gerador*. É parecida com uma função normal, exceto pelo fato de conter expressões `yield` para produzir uma série de valores que podem ser usados em um laço "for" ou que podem ser obtidos um de cada vez com a função `next()`.

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 *geradora*.

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).

expressão geradora 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
```

função genérica Uma função composta por várias 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 *despacho único* no glossário, o decorador `functools.singledispatch()`, e a **PEP 443**.

GIL Veja *trava global do interpretador*.

trava global do interpretador O mecanismo utilizado pelo interpretador *CPython* para garantir que apenas uma thread execute o *bytecode* Python por vez. Isto simplifica a implementação do CPython ao fazer com que o modelo de objetos (incluindo tipos embutidos críticos como o `dict`) ganhem segurança implícita contra acesso concorrente. Travar todo o interpretador facilita que o interpretador em si seja multitarefa, às custas de muito do paralelismo já provido por máquinas multiprocessador.

No entanto, alguns módulos de extensão, tanto da biblioteca padrão quanto de terceiros, são desenvolvidos de forma a liberar o GIL ao realizar tarefas computacionalmente muito intensas, como compactação ou cálculos de hash. Além disso, o GIL é sempre liberado nas operações de E/S.

No passado, esforços para criar um interpretador que lidasse plenamente com threads (travando dados compartilhados numa granularidade bem mais fina) não foram bem sucedidos devido a queda no desempenho ao serem executados em processadores de apenas um núcleo. Acredita-se que superar essa questão de desempenho acabaria tornando a implementação muito mais complicada e bem mais difícil de manter.

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.

hasheável Um objeto é *hasheável* se tem um valor de hash que nunca muda durante seu ciclo de vida (precisa ter um método `__hash__()`) e pode ser comparado com outros objetos (precisa ter um método `__eq__()`). Objetos hasheáveis que são comparados como iguais devem ter o mesmo valor de hash.

A hasheabilidade faz com que um objeto possa ser usado como uma chave de dicionário e como um membro de conjunto, pois estas estruturas de dados utilizam os valores de hash internamente.

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.

caminho de importação Uma lista de localizações (ou *entradas de caminho*) que são buscadas pelo *localizador baseado no caminho* por módulos para importar. Durante a importação, esta lista de localizações usualmente vem a partir de `sys.path`, mas para subpacotes ela também pode vir do atributo `__path__` de pacotes-pai.

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

importador Um objeto que localiza e carrega um módulo; Tanto um *localizador* e o objeto *carregador*.

interativo Python tem um interpretador interativo, o que significa que você pode digitar instruções e expressões no prompt do interpretador, executá-los imediatamente e ver seus resultados. Apenas execute `python` sem argumentos (possivelmente selecionando-o a partir do menu de aplicações de seu sistema operacional). O interpretador interativo é uma maneira poderosa de testar novas ideias ou aprender mais sobre módulos e pacotes (lembre-se do comando `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*.

desligamento do interpretador 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 *coletor de lixo*. Isto pode disparar a execução de código em destrutores definidos pelo usuário ou função de retorno 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 não sequenciais como `dict`, *objeto arquivo*, e objetos de qualquer classe que você definir com um método `__iter__()` ou com um método `__getitem__()` que implemente 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 embutida `next()`) vão retornar itens sucessivos do fluxo. Quando não houver mais dados disponíveis uma exceção `StopIteration` 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 Uma função chave ou função colação é um chamável que retorna um valor usado para ordenação ou classificação. Por exemplo, `locale.strxfrm()` é usada para produzir uma chave de ordenação que leva o locale em consideração para fins de ordenação.

Uma porção de ferramentas no Python aceitam funções chave para controlar como os elementos são ordenados ou agrupados. Algumas delas incluem `min()`, `max()`, `sorted()`, `list.sort()`, `heapq.merge()`, `heapq.nsmallest()`, `heapq.nlargest()` e `itertools.groupby()`.

Há várias maneiras de se criar funções chave. Por exemplo, o método `str.lower()` pode servir como uma função chave para ordenações insensíveis à caixa. Alternativamente, uma função chave ad-hoc pode ser construída a partir de uma expressão `lambda`, como `lambda r: (r[0], r[2])`. Além disso, o módulo `operator` dispõe de três construtores para funções chave: `attrgetter()`, `itemgetter()` e

o `methodcaller()`. Consulte o HowTo de Ordenação para ver exemplos de como criar e utilizar funções chave.

argumento nomeado Veja *argumento*.

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

LBYL Iniciais da expressão em inglês "look before you leap", que significa algo como "olhe antes de pisar". Este estilo de codificação testa as pré-condições explicitamente antes de fazer chamadas ou buscas. Este estilo contrasta com a abordagem *EAFP* e é caracterizada pela presença de muitas instruções `if`.

Em um ambiente multithread, a abordagem LBYL pode arriscar a introdução de uma condição de corrida entre "o olhar" e "o pisar". Por exemplo, o código `if key in mapping: return mapping[key]` pode falhar se outra thread remover *key* do *mapping* após o teste, mas antes da olhada. Esse problema pode ser resolvido com travas ou usando a abordagem EAFP.

lista Uma *sequência* 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).

compreensão de lista Uma maneira compacta de processar todos ou parte dos elementos de uma sequência e retornar os resultados em uma lista. `result = ['{:#04x}'.format(x) for x in range(256) if x % 2 == 0]` gera uma lista de strings contendo números hexadecimais (0x..) no intervalo de 0 a 255. A cláusula `if` é opcional. Se omitida, todos os elementos no `range(256)` serão processados.

carregador Um objeto que carrega um módulo. Deve definir um método chamado `load_module()`. Um carregador é normalmente devolvido por um *localizador*. Veja a **PEP 302** para detalhes e `importlib.abc.Loader` para um *classe base abstrata*.

método mágico Um sinônimo informal para um *método especial*.

mapeamento Um objeto contêiner que suporta buscas por chaves arbitrárias e implementa os métodos especificados em classes base abstratas `Mapping` ou `MutableMapping`. Exemplos incluem `dict`, `collections.defaultdict`, `collections.OrderedDict` e `collections.Counter`.

localizador de metacaminho Um *localizador* retornado por uma busca de `sys.meta_path`. Localizadores de metacaminho são relacionados a, mas diferentes de, *localizadores de entrada de caminho*.

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

metaclasse A classe de uma classe. Definições de classe criam um nome de classe, um dicionário de classe e uma lista de classes base. A metaclasse é responsável por receber estes três argumentos e criar a classe. A maioria das linguagens de programação orientadas a objetos provê uma implementação default. O que torna o Python especial é o fato de ser possível criar metaclasses personalizadas. A maioria dos usuários nunca vai precisar deste recurso, mas quando houver necessidade, metaclasses possibilitam soluções poderosas e elegantes. Metaclasses têm sido utilizadas para gerar registros de acesso a atributos, para incluir proteção contra acesso concorrente, rastrear a criação de objetos, implementar singletons, dentre muitas outras tarefas.

Mais informações podem ser encontradas em metaclasses.

método Uma função que é definida dentro do corpo de uma classe. Se chamada como um atributo de uma instância daquela classe, o método receberá a instância do objeto como seu primeiro *argumento* (que comumente é chamado de `self`). Veja *função* e *escopo aninhado*.

ordem de resolução de métodos Ordem de resolução de métodos é a ordem em que os membros de uma classe base são buscados durante a pesquisa. Veja A ordem de resolução de métodos do Python 2.3 para detalhes do algoritmo usado pelo interpretador do Python desde a versão 2.3.

módulo Um objeto que serve como uma unidade organizacional de código Python. Os módulos têm um espaço de nomes contendo objetos Python arbitrários. Os módulos são carregados pelo Python através do processo de *importação*.

Veja também *pacote*.

módulo spec Um espaço de nomes que contém as informações relacionadas à importação usadas para carregar um módulo. Uma instância de `importlib.machinery.ModuleSpec`.

MRO Veja *ordem de resolução de métodos*.

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

tupla nomeada O termo "tupla nomeada" é aplicado a qualquer tipo ou classe que herda de tupla 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 de uma definição de 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 fábrica `collections.namedtuple()`. A segunda técnica também adiciona alguns métodos extras, que podem não ser encontrados quando foi escrita manualmente, ou em tuplas nomeadas embutidas.

espaço de nomes O lugar em que uma variável é armazenada. Espaços de nomes são implementados como dicionários. Existem os espaços de nomes local, global e nativo, bem como espaços de nomes aninhados em objetos (em métodos). Espaços de nomes suportam modularidade ao prevenir conflitos de nomes. Por exemplo, as funções `__builtin__.open()` e `os.open()` são diferenciadas por seus espaços de nomes. Espaços de nomes também auxiliam na legibilidade e na manutenibilidade ao torar 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.

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

Veja também *módulo*.

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 espaço de nomes global. O `nonlocal` permite escrita para escopos externos.

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

objeto Qualquer dado que tenha estado (atributos ou valores) e comportamento definidos (métodos). Também a última classe base de qualquer *classe estilo novo*.

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

Veja também *pacote regular* e *pacote de espaço de nomes*.

parâmetro Uma entidade nomeada na definição de uma *função* (ou método) que específica 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 fornecido apenas por posição. Parâmetros somente-posicionais podem ser definidos incluindo o caractere `/` na lista de parâmetros da definição da função após eles, por exemplo *posonly1* e *posonly2* a seguir:

```
def func(posonly1, posonly2, /, positional_or_keyword): ...
```

- *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 que 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 do parâmetro, 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 definido colocando-se `**` antes do nome, por exemplo *kwargs* no exemplo acima.

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

Veja o termo *argumento* no glossário, a pergunta sobre a diferença entre argumentos e parâmetros, a classe `inspect.Parameter`, a seção function e a **PEP 362**.

entrada de caminho Um local único no *caminho de importação* que o *localizador baseado no caminho* consulta para encontrar módulos a serem importados.

localizador de entrada de caminho Um *localizador* retornado por um chamável em `sys.path_hooks` (ou seja, um *gancho de entrada de caminho*) que sabe como localizar os módulos *entrada de caminho*.

Veja `importlib.abc.PathEntryFinder` para os métodos que localizadores de entrada de caminho implementam.

gancho de entrada de caminho Um chamável na lista `sys.path_hook` que retorna um *localizador de entrada de caminho* caso saiba como localizar módulos em uma *entrada de caminho* específica.

localizador baseado no caminho Um dos *localizadores de metacaminho* padrão que procura por um *caminho de importação* de módulos.

objeto caminho ou similar Um objeto representando um caminho de sistema de arquivos. 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 têm 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**.

porção Um conjunto de arquivos em um único diretório (possivelmente armazenado em um arquivo zip) que contribuem para um pacote de espaço de nomes, conforme definido em **PEP 420**.

argumento posicional Veja *argumento*.

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 *API provisória*.

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

Pythônico 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`. Muitas outras linguagens 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, Pythônico:

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

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

Quando usado para se referir a módulos, o *nome totalmente qualificado* 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'
```

contagem de referências 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.

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

Veja também *pacote de espaço de nomes*.

__slots__ Uma declaração dentro de uma classe que economiza memória pré-declarando espaço para atributos de 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 *iterável* 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 embutidos 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()`.

compreensão de conjunto Uma maneira compacta de processar todos ou parte dos elementos em iterável e retornar um conjunto com os resultados. `results = {c for c in 'abracadabra' if c not in 'abc'}` gera um conjunto de strings `{'r', 'd'}`. Veja comprehensions.

despacho único Uma forma de despacho de *função genérica* onde a implementação é escolhida com base no tipo de um único argumento.

fatia Um objeto geralmente contendo uma parte de uma *sequência*. 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.

instrução Uma instrução é parte de uma suíte (um "bloco" de código). Uma instrução é ou uma *expressão* ou uma de várias construções com uma palavra reservada, tal como `if`, `while` ou `for`.

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

arquivo texto Um *objeto arquivo* apto a ler e escrever objetos `str`. Geralmente, um arquivo texto, na verdade, acessa um fluxo de dados de bytes e captura o *codificador de texto* 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 *arquivo binário* para um objeto arquivo apto a ler e escrever *objetos byte ou similar*.

string entre aspas triplas Uma string que está definida com três ocorrências de aspas duplas (") ou apóstrofos ('). 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 escapadas dentro de uma string, e elas podem utilizar múltiplas linhas sem o uso de caractere 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; cada objeto tem um tipo. Um tipo de objeto é acessível pelo atributo `__class__` ou pode ser recuperado com `type(obj)`.

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

Tipos alias são úteis para simplificar *dicas de tipo*. 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**, a qual descreve esta funcionalidade.

dica de tipo Uma *anotação* 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 de tipos estático, 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**, a qual descreve esta funcionalidade.

novas linhas universais Uma maneira de interpretar fluxos de textos, na qual todos estes são reconhecidos como caracteres de fim 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 de variável Uma *anotação* de uma variável ou um atributo de classe.

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

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

Anotações de variáveis são normalmente usadas para *dicas de tipo*: por exemplo, espera-se que esta variável receba valores do tipo `int`:

```
count: int = 0
```

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

Veja *anotação de função*, **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`.

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

Zen do 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.

APÊNDICE B

Sobre esses documentos

Esses documentos são gerados a partir de reStructuredText pelo Sphinx, um processador de documentos especificamente escrito para documentação Python.

O desenvolvimento da documentação e de suas ferramentas é um esforço totalmente voluntário, como 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 Python e escritor de boa parte do conteúdo;
- O projeto Docutils por criar reStructuredText e o pacote Docutils;
- Fredrik Lundh por seu projeto Referência Alternativa para Python do qual Sphinx teve muitas ideias boas.

B.1 Contribuidores da Documentação Python

Muitas pessoas tem contribuído para a linguagem Python, sua biblioteca padrão e sua documentação. Veja Misc/ACKS na distribuição do código 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!

APÊNDICE C

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 principal de desenvolvimento do Python mudaram-se para o BeOpen.com para formar a equipe BeOpen PythonLabs. Em outubro do mesmo ano, a equipe da PythonLabs mudou para a Digital Creations (agora Zope Corporation; veja https://www.zope.org/). Em 2001, formou-se a Python Software Foundation (PSF, veja https://www.python.org/psf/), uma organização sem fins lucrativos criada especificamente para possuir propriedade intelectual relacionada a 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.

Versão	Derivada de	Ano	Proprietário	Compatível com a GPL?
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 a 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 a 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

O software e a documentação do Python são licenciados sob o *Acordo de Licenciamento PSF*.

A partir do Python 3.8.6, exemplos, receitas e outros códigos na documentação são licenciados duplamente sob o Acordo de Licenciamento PSF e a *Licença BSD de Zero Cláusula*.

Alguns softwares incorporados ao Python estão sob licenças diferentes. As licenças são listadas com o código abrangido por essa licença. Veja *Licenças e Reconhecimentos para Software Incorporado* para uma lista incompleta dessas licenças.

C.2.1 ACORDO DE LICENCIAMENTO DA PSF PARA PYTHON 3.8.20

```
1. This LICENSE AGREEMENT is between the Python Software Foundation␣
↪("PSF"), and
   the Individual or Organization ("Licensee") accessing and otherwise␣
↪using Python
   3.8.20 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 3.8.20 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 3.8.20 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 3.8.20 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
   3.8.20.

4. PSF is making Python 3.8.20 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 3.8.20 WILL NOT INFRINGE ANY THIRD PARTY RIGHTS.

5. PSF SHALL NOT BE LIABLE TO LICENSEE OR ANY OTHER USERS OF PYTHON 3.8.20
   FOR ANY INCIDENTAL, SPECIAL, OR CONSEQUENTIAL DAMAGES OR LOSS AS A␣
↪RESULT OF
   MODIFYING, DISTRIBUTING, OR OTHERWISE USING PYTHON 3.8.20, 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 3.8.20, Licensee agrees
   to be bound by the terms and conditions of this License Agreement.
```

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

ACORDO DE LICENCIAMENTO DA BEOPEN DE FONTE ABERTA DO 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
```

(continua na próxima página)

(continuação da página anterior)

```
   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
   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
   under the GNU General Public License (GPL), the law of the Commonwealth of
```

(continua na próxima página)

(continuação da página anterior)

```
   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.2.5 LICENÇA BSD DE ZERO CLÁUSULA PARA CÓDIGO NA DOCUMENTAÇÃO DO PYTHON 3.8.20

```
Permission to use, copy, modify, and/or distribute this software for any
purpose with or without fee is hereby granted.

THE SOFTWARE IS PROVIDED "AS IS" AND THE AUTHOR DISCLAIMS ALL WARRANTIES WITH
REGARD TO THIS SOFTWARE INCLUDING ALL IMPLIED WARRANTIES OF MERCHANTABILITY
AND FITNESS. IN NO EVENT SHALL THE AUTHOR BE LIABLE FOR ANY SPECIAL, DIRECT,
INDIRECT, OR CONSEQUENTIAL DAMAGES OR ANY DAMAGES WHATSOEVER RESULTING FROM
LOSS OF USE, DATA OR PROFITS, WHETHER IN AN ACTION OF CONTRACT, NEGLIGENCE OR
OTHER TORTIOUS ACTION, ARISING OUT OF OR IN CONNECTION WITH THE USE OR
PERFORMANCE OF THIS SOFTWARE.
```

C.3 Licenças e Reconhecimentos para Software Incorporado

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

C.3.1 Mersenne Twister

O módulo `_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).

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 Soquetes

O módulo `socket` usa as funções `getaddrinfo()` e `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.

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 `asynchat` e `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 `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.

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 `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 `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
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 `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 MERCHANT-
ABILITY AND FITNESS.  IN NO EVENT SHALL SECRET LABS AB OR THE AUTHOR
BE LIABLE FOR ANY SPECIAL, INDIRECT OR CONSEQUENTIAL DAMAGES OR ANY
DAMAGES WHATSOEVER RESULTING FROM LOSS OF USE, DATA OR PROFITS,
WHETHER IN AN ACTION OF CONTRACT, NEGLIGENCE OR OTHER TORTIOUS
ACTION, ARISING OUT OF OR IN CONNECTION WITH THE USE OR PERFORMANCE
OF THIS SOFTWARE.
```

C.3.8 test_epoll

O módulo `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 kqueue de seleção

O módulo `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 `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 `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 `hashlib`, `posix`, `ssl`, `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
   * 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)
```

(continua na próxima página)

(continuação da página anterior)

```
    * ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED
    * OF THE POSSIBILITY OF SUCH DAMAGE.
    * ====================================================================
    *
    * This product includes cryptographic software written by Eric Young
    * (eay@cryptsoft.com).  This product includes software written by Tim
    * Hudson (tjh@cryptsoft.com).
    *
    */

Original SSLeay License
-----------------------

   /* Copyright (C) 1995-1998 Eric Young (eay@cryptsoft.com)
    * All rights reserved.
    *
    * This package is an SSL implementation written
    * by Eric Young (eay@cryptsoft.com).
    * The implementation was written so as to conform with Netscapes SSL.
    *
    * This library is free for commercial and non-commercial use as long as
    * the following conditions are aheared to.  The following conditions
    * apply to all code found in this distribution, be it the RC4, RSA,
    * lhash, DES, etc., code; not just the SSL code.  The SSL documentation
    * included with this distribution is covered by the same copyright terms
    * except that the holder is Tim Hudson (tjh@cryptsoft.com).
    *
    * Copyright remains Eric Young's, and as such any Copyright notices in
    * the code are not to be removed.
    * If this package is used in a product, Eric Young should be given attribution
    * as the author of the parts of the library used.
    * This can be in the form of a textual message at program startup or
    * in documentation (online or textual) provided with the package.
    *
    * Redistribution and use in source and binary forms, with or without
    * modification, are permitted provided that the following conditions
    * are met:
    * 1. Redistributions of source code must retain the copyright
    *    notice, this list of conditions and the following disclaimer.
    * 2. Redistributions in binary form must reproduce the above copyright
    *    notice, this list of conditions and the following disclaimer in the
    *    documentation and/or other materials provided with the distribution.
    * 3. All advertising materials mentioning features or use of this software
    *    must display the following acknowledgement:
    *    "This product includes cryptographic software written by
    *     Eric Young (eay@cryptsoft.com)"
    *    The word 'cryptographic' can be left out if the rouines from the library
    *    being used are not cryptographic related :-).
    * 4. If you include any Windows specific code (or a derivative thereof) from
    *    the apps directory (application code) you must include an acknowledgement:
    *    "This product includes software written by Tim Hudson (tjh@cryptsoft.com)"
    *
    * THIS SOFTWARE IS PROVIDED BY ERIC YOUNG ``AS IS'' AND
    * ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
    * IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
    * ARE DISCLAIMED.  IN NO EVENT SHALL THE AUTHOR OR CONTRIBUTORS BE LIABLE
    * FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL
    * DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS
    * OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION)
    * HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT
    * LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY
```

(continua na próxima página)

(continuação da página anterior)

```
    * 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 `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 `_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
```

(continua na próxima página)

(continuação da página anterior)

```
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 `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 `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
```

(continua na próxima página)

(continuação da página anterior)

```
    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 `_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.

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.3.18 Conjunto de testes C14N do W3C

O conjunto de testes C14N 2.0 no pacote `test` (`Lib/test/xmltestdata/c14n-20/`) foi recuperado do site do W3C em https://www.w3.org/TR/xml-c14n2-testcases/ e é distribuído sob a licença BSD de 3 cláusulas:

> Copyright (c) 2013 W3C(R) (MIT, ERCIM, Keio, Beihang), All Rights Reserved.
>
> A redistribuição e uso nas formas de fonte e binárias, com ou sem modificação, são permitidas desde que as seguintes condições sejam atendidas:
>
> - As redistribuições de obras devem manter o aviso de direitos autorais original, esta lista de condições e o aviso de isenção de responsabilidade a seguir.

- As redistribuições em formato binário devem reproduzir o aviso de direitos autorais original, esta lista de condições e o aviso de isenção de responsabilidade a seguir na documentação e/ou outros materiais fornecidos com a distribuição.
- Nem o nome do W3C nem os nomes de seus colaboradores podem ser usados para endossar ou promover produtos derivados deste trabalho sem permissão prévia por escrito específica.

ESTE SOFTWARE É FORNECIDO PELOS DETENTORES DE DIREITOS AUTORAIS E CONTRIBUIDORES "COMO ESTÁ" E QUALQUER GARANTIA EXPRESSA OU IMPLÍCITA, INCLUINDO, MAS NÃO SE LIMITANDO A, AS GARANTIAS IMPLÍCITAS DE COMERCIALIZAÇÃO E ADEQUAÇÃO A UM PROPÓSITO ESPECÍFICO. EM NENHUM CASO O DETENTOR DE DIREITOS AUTORAIS OU OS CONTRIBUIDORES SERÃO RESPONSÁVEIS POR QUALQUER DANO DIRETO, INDIRETO, INCIDENTAL, ESPECIAL, EXEMPLAR OU CONSEQUENCIAL (INCLUINDO, MAS NÃO SE LIMITANDO A, PROCURAÇÃO DE BENS OU SERVIÇOS SUBSTITUTOS; PERDA DE USO, DADOS, LUCROS DE USO; OU INTERRUPÇÃO DE NEGÓCIOS), CAUSADO E QUALQUER TEORIA DE RESPONSABILIDADE, CONTRATADA, RESPONSABILIDADE RÍGIDA OU ATRIBUIÇÃO (INCLUINDO NEGLIGÊNCIA OU DE OUTRA FORMA), SURGINDO DE ALGUMA FORMA FORA DO USO DESTE SOFTWARE, MESMO QUE SEJA ACONSELHÁVEL A POSSIBILIDADE DE TAL CONTEÚDO.

APÊNDICE D

Direitos autorais

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

Índice

Não alfabético

A

B

C

D

E

F

G

H

I

J

L

M

N

O

P

Q

R

S

T

U

V

W

X

Z