Python Setup and Usage

Release 3.14.0a1

Guido van Rossum and the Python development team

novembro 05, 2024

Python Software Foundation
Email: docs@python.org

Sumário

1 Linha de comando e ambiente **3**
1.1 Linha de comando . . . 3
1.1.1 Opções de interface . . . 3
1.1.2 Opções genéricas . . . 5
1.1.3 Opções diversas . . . 6
1.1.4 Controlando cores . . . 11
1.1.5 Opções que você não deve usar . . . 11
1.2 Variáveis de ambiente . . . 11
1.2.1 Variáveis de modo de depuração . . . 17

2 Utilizando Python em plataformas Unix **19**
2.1 Obtendo e instalando a versão mais recente do Python . . . 19
2.1.1 No Linux . . . 19
2.1.2 No FreeBSD e OpenBSD . . . 19
2.2 Compilando o Python . . . 20
2.3 Paths e arquivos relacionados com o Python . . . 20
2.4 Diversos . . . 20
2.5 OpenSSL personalizado . . . 21

3 Configurar o Python **23**
3.1 Requisitos de construção . . . 23
3.2 Arquivos gerados . . . 24
3.2.1 Script configure . . . 24
3.3 Opções de configuração . . . 24
3.3.1 Opções gerais . . . 24
3.3.2 Opções do compilador C . . . 27
3.3.3 Opções da ligação . . . 27
3.3.4 Opções para dependências de terceiros . . . 27
3.3.5 Opções de WebAssembly . . . 29
3.3.6 Opções de instalação . . . 29
3.3.7 Opções de desempenho . . . 30
3.3.8 Compilação de depuração do Python . . . 31
3.3.9 Opções de depuração . . . 32
3.3.10 Opções da ligação . . . 33
3.3.11 Opções da biblioteca . . . 33
3.3.12 Opções de segurança . . . 34
3.3.13 Opções do macOS . . . 35
3.3.14 Opções do iOS . . . 36
3.3.15 Opções de compilação cruzada . . . 36
3.4 Sistema de Construção Python . . . 37
3.4.1 Arquivos principais do sistema de construção . . . 37

3.4.2 Principais etapas de construção . . . 37
3.4.3 Alvos principais do Makefile . . . 37
3.4.4 Extensões C . . . 39
3.5 Sinalizadores do compilador e do vinculador . . . 39
3.5.1 Sinalizadores do pré-processador . . . 39
3.5.2 Sinalizadores do compilador . . . 40
3.5.3 Sinalizadores do vinculador . . . 41

4 Utilizando Python no Windows **43**
4.1 O instalador completo . . . 43
4.1.1 Etapas de instalação . . . 43
4.1.2 Removendo a Limitação do MAX_PATH . . . 45
4.1.3 Instalando sem UI . . . 45
4.1.4 Instalando Sem Download . . . 47
4.1.5 Modificando uma instalação . . . 47
4.1.6 Instalando binários com threads livres . . . 48
4.2 O pacote Microsoft Store . . . 49
4.2.1 Problemas conhecidos . . . 49
4.3 Os pacotes nuget.org . . . 50
4.3.1 Pacotes com threads livres . . . 51
4.4 O pacote embutível . . . 51
4.4.1 Aplicação Python . . . 51
4.4.2 Incorporando Python . . . 52
4.5 Pacotes Alternativos . . . 52
4.6 Configurando o Python . . . 52
4.6.1 Excursus: Configurando variáveis de ambiente . . . 52
4.6.2 Encontrando o executável do Python . . . 53
4.7 Modo UTF-8 . . . 53
4.8 Inicializador Python para Windows . . . 54
4.8.1 Começando . . . 54
4.8.2 Linhas Shebang . . . 56
4.8.3 Argumentos em linhas shebang . . . 57
4.8.4 Personalização . . . 57
4.8.5 Diagnósticos . . . 58
4.8.6 Teste de simulação . . . 58
4.8.7 Instalar sob demanda . . . 58
4.8.8 Códigos de retorno . . . 59
4.9 Encontrando módulos . . . 59
4.10 Módulos adicionais . . . 60
4.10.1 PyWin32 . . . 60
4.10.2 cx_Freeze . . . 61
4.11 Compilando Python no Windows . . . 61
4.12 Outras plataformas . . . 61

5 Usando Python em macOS **63**
5.1 Usando Python para macOS do `python.org` . . . 63
5.1.1 Etapas de instalação . . . 63
5.1.2 Como executar um script Python . . . 71
5.2 Distribuições alternativas . . . 71
5.3 Instalando pacotes adicionais ao python . . . 71
5.4 Programação de GUI . . . 72
5.5 Tópicos Avançados . . . 72
5.5.1 Instalando binários com threads livres . . . 72
5.5.2 Instalação usando a linha de comando . . . 74
5.5.3 Distribuindo aplicações Python . . . 75
5.5.4 Conformidade com a App Store . . . 75
5.6 Outros recursos . . . 76

6 Utilizando Python no Android **77**
6.1 Adicionando Python a uma aplicação Android . . . 77

7 Usando Python no iOS **79**
7.1 Python em tempo de execução no iOS . . . 79
7.1.1 Compatibilidade com a versão do iOS . . . 79
7.1.2 Identificação da plataforma . . . 79
7.1.3 Disponibilidade da biblioteca padrão . . . 80
7.1.4 Módulos de extensão binária . . . 80
7.1.5 Binários stub do compilador . . . 80
7.2 Instalando Python no iOS . . . 81
7.2.1 Ferramentas para construir aplicações de iOS . . . 81
7.2.2 Adicionando Python a um projeto iOS . . . 81
7.3 Conformidade com a App Store . . . 84

8 Editores e IDEs **85**

A Glossário **87**

B Sobre esses documentos **105**
B.1 Contribuidores da Documentação Python . . . 105

C História e Licença **107**
C.1 História do software . . . 107
C.2 Termos e condições para acessar ou usar Python . . . 108
C.2.1 ACORDO DE LICENCIAMENTO DA PSF PARA PYTHON 3.14.0a1 . . . 108
C.2.2 ACORDO DE LICENCIAMENTO DA BEOPEN.COM PARA PYTHON 2.0 . . . 109
C.2.3 CONTRATO DE LICENÇA DA CNRI PARA O PYTHON 1.6.1 . . . 109
C.2.4 ACORDO DE LICENÇA DA CWI PARA PYTHON 0.9.0 A 1.2 . . . 111
C.2.5 LICENÇA BSD DE ZERO CLÁUSULA PARA CÓDIGO NA DOCUMENTAÇÃO DO PYTHON 3.14.0a1 . . . 111
C.3 Licenças e Reconhecimentos para Software Incorporado . . . 111
C.3.1 Mersenne Twister . . . 111
C.3.2 Soquetes . . . 112
C.3.3 Serviços de soquete assíncrono . . . 113
C.3.4 Gerenciamento de cookies . . . 113
C.3.5 Rastreamento de execução . . . 114
C.3.6 Funções UUencode e UUdecode . . . 114
C.3.7 Chamadas de procedimento remoto XML . . . 115
C.3.8 test_epoll . . . 115
C.3.9 kqueue de seleção . . . 116
C.3.10 SipHash24 . . . 116
C.3.11 strtod e dtoa . . . 117
C.3.12 OpenSSL . . . 117
C.3.13 expat . . . 120
C.3.14 libffi . . . 121
C.3.15 zlib . . . 121
C.3.16 cfuhash . . . 122
C.3.17 libmpdec . . . 123
C.3.18 Conjunto de testes C14N do W3C . . . 123
C.3.19 mimalloc . . . 124
C.3.20 asyncio . . . 124
C.3.21 Global Unbounded Sequences (GUS) . . . 125

D Direitos autorais **127**

Índice **129**

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.

Detalhes da implementação do CPython: 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 [-bBdEhiIOPqRsSuvVWx?] [-c comando | -m nome-módulo | script | - ] [args]
```

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

```
python meuscript.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. Para mais sobre o modo interativo, veja tut-interac.
- 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 "configuração inicial aqui" "código a ser analisado aqui"
python -m timeit -h # para detalhes
```

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

Imprime uma breve descrição de todas as opções de linha de comando e variáveis de ambiente correspondentes e sai.

--help-env

Imprime uma breve descrição das variáveis de ambiente específicas do Python e sai.

Adicionado na versão 3.11.

--help-xoptions

Imprime uma descrição das opções `-X` específica da implementação e sai.

Adicionado na versão 3.11.

--help-all

Imprime as informações de uso completas e sai.

Adicionado na versão 3.11.

-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]
```

Adicionado na versão 3.6: A opção `-VV`.

1.1.3 Opções diversas

-b

Emite um aviso ao converter `bytes` ou `bytearray` para `str` sem especificar codificação ou 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 também 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 analisador sintático de depuração (uso avançado). Consulte também a variável de ambiente *PYTHONDEBUG*.

Essa opção requer uma *construção de depuração do Python*, caso contrário, será ignorada.

-E

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

Veja também as opções *-P* e *-I* (isolado).

-i

Enter interactive mode after execution.

Using the *-i* option will enter interactive mode in any of the following circumstances:

- When a script is passed as first argument
- When the *-c* option is used
- When the *-m* option is used

Interactive mode will start even when `sys.stdin` does not appear to be a terminal. The *PYTHONSTARTUP* file is not read.

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 nas opções *-E*, *-P* 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.

Adicionado 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**.

-P

Não anexa um caminho potencialmente inseguro a `sys.path`:

- A linha de comando `python -m módulo`: Não anexa o diretório atual.
- A linha de comando `python script.py`: não anexa o diretório do script. Se for um link simbólico, resolve os links simbólicos.
- Linhas de comando `python -c código` e `python` (REPL): Não anexa uma string vazia, o que significa o diretório de trabalho atual.

Veja também a variável de ambiente *PYTHONSAFEPATH* e as opções *-E* e *-I* (isolado).

Adicionado na versão 3.11.

-q

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

Adicionado 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://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.

Adicionado na versão 3.2.3.

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

-s

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

Veja também *PYTHONNOUSERSITE*.

> **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`), exibe uma mensagem para cada arquivo que é verificado durante a busca por um módulo. Também fornece informações sobre a limpeza do módulo na saída.

Alterado na versão 3.10: O módulo `site` relata os caminhos específicos do site e os arquivos `.pth` sendo processados.

Veja também *PYTHONVERBOSE*.

-W `arg`

Controle de advertência. O mecanismo de aviso do Python por padrão exibe mensagens de aviso para `sys.stderr`.

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  # Avisa uma vez por local de chamada
-Werror    # Converte para exceções
-Walways   # Avisa toda vez
-Wall      # Mesmo que -Walways
-Wmodule   # Avisa uma vez por módulo chamador
-Wonce     # Avisa uma vez por processo do Python
-Wignore   # Nunca avisar
```

Os nomes das ações podem ser abreviados conforme desejado e o interpretador irá resolvê-los para o nome da ação apropriado. Por exemplo, `-Wi` é o mesmo que `-Wignore`.

A forma completa de argumento é:

```
ação:mensagem:categoria:módulo:número-da-linha
```

Os campos vazios correspondem a todos os valores; campos vazios à direita podem ser omitidos. Por exemplo, `-W ignore::DeprecationWarning` ignora todos os avisos de DeprecationWarning.

O campo *action* é explicado acima, mas se aplica apenas a avisos que correspondem aos campos restantes.

O campo *message* deve corresponder a toda a mensagem de aviso; essa correspondência não diferencia maiúsculas de minúsculas.

O campo *category* corresponde à categoria de aviso (ex: `DeprecationWarning`). Deve ser um nome de classe; o teste de correspondência se a categoria de aviso real da mensagem é uma subclasse da categoria de aviso especificada.

O campo *module* corresponde ao nome do módulo (totalmente qualificado); esta correspondência diferencia maiúsculas de minúsculas.

O campo *lineno* corresponde ao número da linha, onde zero corresponde a todos os números da linha e, portanto, é equivalente a um número da linha omitido.

Múltiplas 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 exibida 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`. Por exemplo, a função `warnings.filterwarnings()` pode ser usada para usar uma expressão regular na mensagem de aviso.

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`. Veja também `PYTHONFAULTHANDLER`.

  Adicionado na versão 3.3.

- `-X showrefcount` para emitir 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 *construções de depuração*.

  Adicionado na versão 3.4.

- `-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()` e `PYTHONTRACEMALLOC` para mais informações.

  Adicionado na versão 3.4.

- `-X int_max_str_digits` configura a limitação de comprimento de string na conversão para inteiro. Veja também `PYTHONINTMAXSTRDIGITS`.

  Adicionado na versão 3.11.

- `-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`.

  Adicionado na versão 3.7.

- `-X dev`: habilita Modo de Desenvolvimento do Python, introduzindo verificações adicionais de tempo de execução que são muito custosas para serem habilitadas por padrão. Veja também `PYTHONDEVMODE`.

  Adicionado na versão 3.7.

- `-X utf8` habilita o Modo UTF-8 do Python. `-X utf8=0` explicitamente desabilita o Modo UTF-8 do Python (mesmo quando de outra forma seria ativado automaticamente). Veja também `PYTHONUTF8`.

  Adicionado na versão 3.7.

- `-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*.

  Adicionado na versão 3.8.

- `-X warn_default_encoding` emite uma `EncodingWarning` quando a codificação padrão específica da localidade é usada para abrir arquivos. Veja também *PYTHONWARNDEFAULTENCODING*.

  Adicionado na versão 3.10.

- `-X no_debug_ranges` desabilita a inclusão das tabelas que mapeiam informações de localização extra (linha final, deslocamento da coluna inicial e deslocamento da coluna final) para cada instrução em objetos código. Isso é útil quando objetos código menores e arquivos pyc são desejados, bem como suprimir os indicadores de localização visual extra quando o interpretador exibe tracebacks. Veja também *PYTHONNODEBUGRANGES*.

  Adicionado na versão 3.11.

- `-X frozen_modules` determina se os módulos congelados são ou não ignorados pelo maquinário de importação. Um valor de `on` significa que eles são importados e `off` significa que eles são ignorados. O padrão é `on` se este for um Python instalado (o caso normal). Se estiver em desenvolvimento (executando a partir da árvore de origem), o padrão é `off`. Observe que os módulos congelados `importlib_bootstrap` e `importlib_bootstrap_external` são sempre usados, mesmo que esse sinalizador esteja definido como `off`. Veja também *PYTHON_FROZEN_MODULES*.

  Adicionado na versão 3.11.

- `-X perf` habilita o suporte para o perfilador do Linux `perf`. Quando essa opção for fornecida, o perfilador `perf` poderá relatar chamadas Python. Essa opção está disponível somente em algumas plataformas e não fará nada se não for compatível com o sistema atual. A opção padrão valor é "off". Consulte também *PYTHONPERFSUPPORT* e perf_profiling.

  Adicionado na versão 3.12.

- `-X perf_jit` habilita o suporte para o perfilador do Linux `perf` com suporte a DWARF. Quando essa opção for fornecida, o perfilador `perf` poderá relatar chamadas do Python usando informações de DWARF. Essa opção está disponível somente em algumas plataformas e não fará nada se não for compatível com o sistema atual. A opção padrão valor é "off". Veja também *PYTHON_PERF_JIT_SUPPORT* e perf_profiling.

  Adicionado na versão 3.13.

- `-X cpu_count=`*n* substitui `os.cpu_count()`, `os.process_cpu_count()` e `multiprocessing.cpu_count()`. *n* deve ser maior ou igual a 1. Esta opção pode ser útil para usuários que precisam limitar os recursos da CPU de um sistema contêiner. Veja também *PYTHON_CPU_COUNT*. Se *n* for `default`, nada será substituído.

  Adicionado na versão 3.13.

- `-X presite=`*pacote.,módulo* especifica um módulo que deve ser importado antes do módulo `site` ser executado e antes do módulo `__main__` existir. Portanto, o módulo importado não é `__main__`. Isso pode ser usado para executar código antecipadamente durante a inicialização do Python. Python precisa ser *construído no modo de depuração* para que esta opção exista. Veja também *PYTHON_PRESITE*.

  Adicionado na versão 3.13.

- `-X gil=`*0,1* força a GIL a ser desabilitada ou habilitada, respectivamente. A definição para `0` está disponível apenas em construções configuradas com *--disable-gil*. Veja também *PYTHON_GIL* e whatsnew313-free-threaded-cpython.

  Adicionado na versão 3.13.

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

Adicionado na versão 3.2.

Alterado na versão 3.9: Removida a opção `-X showalloccount`.

Alterado na versão 3.10: Removida a opção `-X oldparser`.

1.1.4 Controlando cores

O interpretador Python é configurado por padrão para usar cores para destacar a saída em determinadas situações, como ao exibir tracebacks. Este comportamento pode ser controlado definindo diferentes variáveis de ambiente.

Definir a variável de ambiente `TERM` como `dumb` desativará cores.

Se a variável de ambiente `FORCE_COLOR` estiver definida, a cor será habilitada independentemente do valor de TERM. Isso é útil em sistemas CI que não são terminais, mas ainda podem exibir sequências de escape ANSI.

Se a variável de ambiente `NO_COLOR` estiver definida, o Python desativará todas as cores na saída. Isto tem precedência sobre `FORCE_COLOR`.

Todas essas variáveis de ambiente também são usadas por outras ferramentas para controlar a saída de cores. Para controlar a saída de cores apenas no interpretador Python, a variável de ambiente *`PYTHON_COLORS`* pode ser usada. Esta variável tem precedência sobre `NO_COLOR`, que por sua vez tem precedência sobre `FORCE_COLOR`.

1.1.5 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`.

`PYTHONSAFEPATH`

Se for definido como uma string não vazia, não anexa um caminho potencialmente inseguro para `sys.path`: consulte a opção *`-P`* para obter detalhes.

Adicionado na versão 3.11.

`PYTHONPLATLIBDIR`

Se for definido como uma string não vazia, ela substitui o valor `sys.platlibdir`.

Adicionado na versão 3.9.

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 o nome de arquivo como argumento quando chamado na inicialização.

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.

Adicionado 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.

Essa variável de ambiente requer uma *construção de depuração do Python*, caso contrário, será ignorada.

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 3.12.5: (também 3.11.10, 3.10.15, 3.9.20 e 3.8.20) Emite eventos de auditoria.

Alterado na versão 3.13: Usa PyrePL, se possível, nesse caso *PYTHONSTARTUP* também é executado. Emite eventos de auditoria.

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`.

Adicionado 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.

Adicionado 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.

Adicionado na versão 3.11.

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 para `python -m pip 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 macOS.

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  # Avisa uma vez por local de chamada
PYTHONWARNINGS=error    # Converte para exceções
PYTHONWARNINGS=always   # Avisa toda vez
PYTHONWARNINGS=all      # Mesmo que PYTHONWARNINGS=always
PYTHONWARNINGS=module   # Avisa uma vez por módulo chamador
PYTHONWARNINGS=once     # Avisa uma vez por processo do Python
PYTHONWARNINGS=ignore   # Nunca avisar
```

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` e `SIGILL` para despejar o traceback (situação da pilha de execução) do Python. Isso é equivalente à opção `-X faulthandler`.

Adicionado 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 a função `tracemalloc.start()` para mais informações. Isso é o equivalente a definir a opção `-X tracemalloc`.

Adicionado 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 o equivalente a definir a opção `-X importtime`.

Adicionado 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`.

Adicionado 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`.
- `mimalloc`: usa o alocador mimalloc para domínios `PYMEM_DOMAIN_MEM` e `PYMEM_DOMAIN_OBJ` e usa a função `malloc()` para o domínio `PYMEM_DOMAIN_RAW`.

Instala 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.
- `mimalloc_debug`: o mesmo que `mimalloc`, mas também instala ganchos de depuração.

Adicionado na versão 3.6.

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

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, o modo padrão do *tratador de erros e codificação do sistema de arquivos* irá reverter 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

Adicionado 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

Adicionado 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.

Disponibilidade

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

PYTHONDEVMODE

Se esta variável de ambiente for definida como uma string não vazia, habilita Modo de Desenvolvimento do Python, introduzindo verificações adicionais de tempo de execução que são muito caras para serem habilitadas por padrão. Isso é o equivalente a definir a opção *-X* `dev`.

Adicionado na versão 3.7.

PYTHONUTF8

Se definido para `1`, habilita o modo UTF-8 do Python.

Se definido para `0`, desabilita o modo UTF-8 do Python.

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

Adicionado na versão 3.7.

PYTHONWARNDEFAULTENCODING

Se esta variável de ambiente for definida como uma string não vazia, emite uma `EncodingWarning` quando a codificação padrão específica da localidade é usada.

Veja io-encoding-warning para detalhes.

Adicionado na versão 3.10.

PYTHONNODEBUGRANGES

Se esta variável estiver definida, ela desabilita a inclusão das tabelas que mapeiam informações de localização extra (linha final, deslocamento da coluna inicial e deslocamento da coluna final) para cada instrução em objetos código. Isso é útil quando objetos código menores e arquivos pyc são desejados, bem como suprimir os indicadores de localização visual extra quando o interpretador exibe tracebacks.

Adicionado na versão 3.11.

PYTHONPERFSUPPORT

Se essa variável for definida como valor diferente de zero, ela habilitará suporte para o perfilador do Linux `perf` para que as chamadas Python possam ser detectadas por ele.

Se definido como `0`, desativa o suporte ao perfilador do Linux `perf`.

Consulte também a opção de linha de comando *-X perf* e perf_profiling.

Adicionado na versão 3.12.

PYTHON_PERF_JIT_SUPPORT

Se essa variável for definida como valor diferente de zero, ela habilitará suporte para o perfilador do Linux `perf` para que as chamadas Python possam ser detectadas por ele usando informações de DWARF.

Se definido como `0`, desativa o suporte ao perfilador do Linux `perf`.

Veja também a opção de linha de comando *-X perf_jit* e perf_profiling.

Adicionado na versão 3.13.

PYTHON_CPU_COUNT

Se esta variável for definida como um número inteiro positivo, ela substitui os valores de retorno de `os.cpu_count()` e `os.process_cpu_count()`.

Veja também a opção de linha de comando *-X cpu_count*.

Adicionado na versão 3.13.

PYTHON_FROZEN_MODULES

Se esta variável for definida como `on` ou `off`, ela determina se os módulos congelados serão ou não ignorados pela maquinário de importação. Um valor `on` significa que eles serão importados e `off` significa que serão ignorados. O padrão é `on` para construções sem depuração (o caso normal) e `off` para construções com depuração. Observe que os módulos congelados `importlib_bootstrap` e `importlib_bootstrap_external` são sempre usados, mesmo se este sinalizador estiver definido como `off`.

Veja também a opção de linha de comando *`-X frozen_modules`*.

Adicionado na versão 3.13.

PYTHON_COLORS

Se esta variável for definida como `1`, o interpretador irá colorir vários tipos de saída. Definir como `0` desativa este comportamento. Veja também *Controlando cores*.

Adicionado na versão 3.13.

PYTHON_BASIC_REPL

Se esta variável for definida como `1`, o interpretador não tentará carregar o *REPL* baseado em Python que requer `curses` e `readline`, e em vez disso usará o baseado em analisador tradicional *REPL*.

Adicionado na versão 3.13.

PYTHON_HISTORY

Esta variável de ambiente pode ser usada para definir a localização de um arquivo `.python_history` (por padrão, é `.python_history` no diretório inicial do usuário).

Adicionado na versão 3.13.

PYTHON_GIL

Se esta variável for definida como `1`, a trava global do interpretador (GIL) será forçada. Definir como `0` força a GIL a desligar (precisa do Python configurado com a opção de construção *`--disable-gil`*).

Veja também a opção de linha de comando *`-X gil`*, que tem precedência sobre esta variável, e whatsnew313--free-threaded-cpython.

Adicionado na versão 3.13.

1.2.1 Variáveis de modo de depuração

PYTHONDUMPREFS

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

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

PYTHONDUMPREFSFILE

Se definido, o Python irá despejar objetos e contagens de referências ainda ativas após desligar o interpretador em um arquivo no caminho fornecido como valor para esta variável de ambiente.

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

Adicionado na versão 3.11.

PYTHON_PRESITE

Se esta variável for definida como um módulo, esse módulo será importado no início do ciclo de vida do interpretador, antes do módulo `site` ser executado, e antes do módulo `__main__` ser criado. Portanto, o módulo importado não é tratado como `__main__`.

Isso pode ser usado para executar código antecipadamente durante a inicialização do Python.

Para importar um submódulo, use `pacote.módulo` como valor, como em uma instrução de importação.

Veja também a opção de linha de comando *`-X presite`*, que tem precedência sobre esta variável.

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

Adicionado na versão 3.13.

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 todas as outras. No entanto, há certos recursos que você pode querer usar que não estão disponíveis no pacote da sua distribuição. Você pode compilar a versão mais recente do Python a partir do código-fonte.

No caso de a versão mais recente do Python não vir pré-instalada e não estar nos repositórios também, você pode criar pacotes para sua própria distro. Dê uma olhada nos 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.fedoraproject.org/en-US/package-maintainers/Packaging_Tutorial_GNU_Hello/
para usuários Fedora

https://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
```

2.2 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* e *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`.

2.5 OpenSSL personalizado

1. Para usar a configuração OpenSSL do seu fornecedor e armazenamento confiável do sistema, localize o diretório com o arquivo `openssl.cnf` ou link simbólico em `/etc`. Na maioria das distribuições o arquivo está em `/etc/ssl` ou em `/etc/pki/tls`. O diretório também deve conter um arquivo `cert.pem` e/ou um diretório `certs`.

```
$ find /etc/ -name openssl.cnf -printf "%h\n"
/etc/ssl
```

2. Baixe, construa e instale o OpenSSL. Certifique-se de usar `install_sw` e não `install`. O destino `install_sw` não substitui o `openssl.cnf`.

```
$ curl -O https://www.openssl.org/source/openssl-VERSÃO.tar.gz
$ tar xzf openssl-VERSÃO
$ pushd openssl-VERSÃO
$ ./config \
    --prefix=/usr/local/custom-openssl \
    --libdir=lib \
    --openssldir=/etc/ssl
$ make -j1 depend
$ make -j8
$ make install_sw
$ popd
```

3. Construa o Python com o OpenSSL personalizado (veja as opções configure `--with-openssl` e `--with-openssl-rpath`)

```
$ pushd python-3.x.x
$ ./configure -C \
    --with-openssl=/usr/local/custom-openssl \
    --with-openssl-rpath=auto \
    --prefix=/usr/local/python-3.x.x
$ make -j8
$ make altinstall
```

Nota

As versões de patches do OpenSSL têm uma ABI compatível com versões anteriores. Você não precisa recompilar o Python para atualizar o OpenSSL. É suficiente substituir a instalação personalizada do OpenSSL por uma versão mais recente.

CAPÍTULO 3

Configurar o Python

3.1 Requisitos de construção

Recursos e versões mínimas necessários para construir o CPython:

- Um compilador C11. Não são necessários recursos opcionais do C11.
- No Windows, é necessário o Microsoft Visual Studio 2017 ou posterior.
- Suporte para números de ponto flutuante do IEEE 754 e Not-a-Number (NaN) de ponto flutuante.
- Suporte a threads.
- OpenSSL 1.1.1 é a versão mínima e OpenSSL 3.0.9 é a versão mínima recomendada para os módulos de extensão `ssl` e `hashlib`.
- SQLite 3.15.2 para o módulo de extensão `sqlite3`.
- Tcl/Tk 8.5.12 para o módulo `tkinter`.
- Autoconf 2.71 e aclocal 1.16.5 são necessários para regenerar o script `configure`.

Alterado na versão 3.1: Agora é necessária a versão 8.3.1 do Tcl/Tk.

Alterado na versão 3.5: No Windows, agora é necessário o Visual Studio 2015 ou posterior. Agora é necessária a versão 8.4 do Tcl/Tk.

Alterado na versão 3.6: Recursos selecionados do C99 agora são necessários, como funções `<stdint.h>` e `static inline`.

Alterado na versão 3.7: Suporte a threads e OpenSSL 1.0.2 agora são necessários.

Alterado na versão 3.10: OpenSSL 1.1.1 agora é necessário. Requer SQLite 3.7.15.

Alterado na versão 3.11: Compilador C11, suporte a IEEE 754 e NaN agora são necessários. No Windows, é necessário o Visual Studio 2017 ou posterior. A versão 8.5.12 do Tcl/Tk agora é necessária para o módulo `tkinter`.

Alterado na versão 3.13: Autoconf 2.71, aclocal 1.16.5 e SQLite 3.15.2 agora são necessários.

Veja também **PEP 7** "Guia de estilo para código C" e **PEP 11** "Suporte do CPython a plataformas".

3.2 Arquivos gerados

Para reduzir as dependências de construção, o código-fonte do Python contém vários arquivos gerados. Comandos para regenerar todos os arquivos gerados:

```
make regen-all
make regen-stdlib-module-names
make regen-limited-abi
make regen-configure
```

O arquivo `Makefile.pre.in` documenta os arquivos gerados, suas entradas e ferramentas usadas para regenerá-los. Procure por alvos `regen-*` de make.

3.2.1 Script configure

O comando `make regen-configure` regera o arquivo `aclocal.m4` e o script `configure` usando o shell script `Tools/build/regen-configure.sh`, o qual usa um contêiner Ubuntu para obter as mesmas versões de ferramentas e ter uma saída reproduzível.

O contêiner é opcional, o seguinte comando pode ser executado localmente:

```
autoreconf -ivf -Werror
```

Os arquivos gerados podem mudar dependendo das versões exatas do `autoconf-archive`, `aclocal` e `pkg-config`.

3.3 Opções de configuração

Liste todas as opções do `configure` usando:

```
./configure --help
```

Veja também o `Misc/SpecialBuilds.txt` na distribuição de código-fonte do Python.

3.3.1 Opções gerais

--enable-loadable-sqlite-extensions

Suporte a extensões carregáveis no módulo de extensão `_sqlite` (o padrão é não) do módulo `sqlite3`.

Veja o método `sqlite3.Connection.enable_load_extension()` do módulo `sqlite3`.

Adicionado na versão 3.6.

--disable-ipv6

Desabilita suporte a IPv6 (habilitado por padrão se houver suporte), veja o módulo `socket`.

--enable-big-digits=[15|30]

Define o tamanho em bits dos dígitos de `int` do Python: 15 ou 30 bits.

Por padrão, o tamanho dos dígitos é 30.

Define o `PYLONG_BITS_IN_DIGIT` para `15` ou `30`.

Veja `sys.int_info.bits_per_digit`.

--with-suffix=SUFFIX

Define o sufixo do executável do Python para *SUFFIX*.

O sufixo padrão é `.exe` no Windows e macOS (executável `python.exe`), `.js` em nó Emscripten, `.html` em navegador Emscripten, `.wasm` em WASI e uma string vazia em outras plataformas (executável `python`).

Alterado na versão 3.11: O sufixo padrão na plataforma WASM é um entre `.js`, `.html` ou `.wasm`

--with-tzpath=<list of absolute paths separated by pathsep>

Seleciona o caminho de pesquisa de fuso horário padrão para `zoneinfo.TZPATH`. Veja a Configuração de tempo de compilação do módulo `zoneinfo`.

Padrão: `/usr/share/zoneinfo:/usr/lib/zoneinfo:/usr/share/lib/zoneinfo:/etc/zoneinfo`.

Veja o separador de caminhos `os.pathsep`.

Adicionado na versão 3.9.

--without-decimal-contextvar

Constrói o módulo de extensão `_decimal` usando um contexto local de thread ao invés de um contexto local de corrotina (padrão), veja o módulo `decimal`.

Veja `decimal.HAVE_CONTEXTVAR` e o módulo `contextvars`.

Adicionado na versão 3.9.

--with-dbmliborder=<list of backend names>

Substitui a ordem de verificação de backends de banco de dados para o módulo `dbm`

Um valor válido é uma string separada por dois pontos (`:`) com os nomes de backend:

- `ndbm`;
- `gdbm`;
- `bdb`.

--without-c-locale-coercion

Desabilita a coerção de localidade C para uma localidade baseada em UTF-8 (ativada por padrão).

Não define a macro `PY_COERCE_C_LOCALE`.

Consulte *`PYTHONCOERCECLOCALE`* e a **PEP 538**.

--with-platlibdir=DIRNAME

Nome do diretório da biblioteca Python (o padrão é `lib`).

Fedora e SuSE usam `lib64` em plataformas de 64 bits.

Veja `sys.platlibdir`.

Adicionado na versão 3.9.

--with-wheel-pkg-dir=PATH

Diretório de pacotes de wheel usados pelo módulo `ensurepip` (nenhum por padrão).

Algumas políticas de empacotamento de distribuição do Linux recomendam contra o empacotamento de dependências. Por exemplo, o Fedora instala pacotes wheel no diretório `/usr/share/python-wheels/` e não instala o pacote `ensurepip._bundled`.

Adicionado na versão 3.10.

--with-pkg-config=[check|yes|no]

Se o configure deve usar **pkg-config** para detectar dependências de construção.

- `check` (padrão): **pkg-config** é opcional
- `yes`: **pkg-config** é obrigatório
- `no`: configure não usa **pkg-config** mesmo quando presente

Adicionado na versão 3.11.

--enable-pystats

Ativa a coleta de estatísticas internas de desempenho do Python.

Por padrão, a coleta de estatísticas está desativada. Use o comando `python3 -X pystats` ou defina a variável de ambiente `PYTHONSTATS=1` para ativar a coleta de estatísticas na inicialização do Python.

Na saída do Python, despeja as estatísticas se a coleta de estatísticas estiver ativada e não apagada.

Efeitos:

- Adiciona a opção de linha de comando *`-X pystats`*.
- Adiciona a variável de ambiente `PYTHONSTATS`.
- Define a macro `Py_STATS`.
- Adiciona funções ao módulo `sys`:
  - `sys._stats_on()`: Ativa a coleta de estatísticas.
  - `sys._stats_off()`: Desativa a coleta de estatísticas.
  - `sys._stats_clear()`: Apaga as estatísticas.
  - `sys._stats_dump()`: Despeja as estatísticas no arquivo e apaga as estatísticas.

As estatísticas serão despejadas em um arquivo arbitrário (provavelmente único) em `/tmp/py_stats/` (Unix) ou `C:\temp\py_stats\` (Windows). Se aquele diretório não existir, os resultados serão enviados para stderr.

Use `Tools/scripts/summarize_stats.py` para ler as estatísticas.

Estatísticas:

- Código de operação:
  - Especialização: sucesso, fracasso, acerto, adiado, erro, deopt, falhas;
  - Contagem de execuções;
  - Contagem em pares.
- Chamada:
  - Chamadas Python em linha;
  - Chamadas a PyEval;
  - Quadros enviados;
  - Objetos quadro criados;
  - Chamadas de Eval: vetor, gerador, legado, função VECTORCALL, classe de construção, slot, função "ex", API, método.
- Objeto:
  - incref e decref;
  - incref e decref do interpretador;
  - alocações: tudo, 512 bytes, 4 kiB, big;
  - memória livre;
  - listas de memória livre para/de;
  - Dicionário materializado/desmaterializado;
  - cache de tipo;
  - tentativas de otimização;
  - rastros de otimização criados/executados;
  - uops executados.

- Coletor de lixo:
  - Coletas de lixo;
  - Objetos visitados;
  - Objetos coletados.

Adicionado na versão 3.11.

--disable-gil

Habilita suporte **experimental** para execução de Python sem a *trava global do interpretador* (GIL): construção de com threads livres.

Define a macro `Py_GIL_DISABLED` e adiciona `"t"` a `sys.abiflags`.

Veja whatsnew313-free-threaded-cpython para mais detalhes.

Adicionado na versão 3.13.

PKG_CONFIG

Caminho para o utilitário `pkg-config`.

PKG_CONFIG_LIBDIR

PKG_CONFIG_PATH

Opções do `pkg-config`.

3.3.2 Opções do compilador C

CC

Comando do compilador C.

CFLAGS

Sinalizadores do compilador C.

CPP

Comando do pré-processador C.

CPPFLAGS

Sinalizadores do pré-processador C, p.ex., `-I`*`include_dir`*.

3.3.3 Opções da ligação

LDFLAGS

Sinalizadores do vinculador. p.ex., `-L`*`library_directory`*.

LIBS

Bibliotecas para passar para o vinculador, p.ex. `-l`*`library`*.

MACHDEP

Nome para arquivos de biblioteca dependentes de máquina.

3.3.4 Opções para dependências de terceiros

Adicionado na versão 3.11.

BZIP2_CFLAGS

BZIP2_LIBS

Sinalizadores de compilador C e vinculador para vincular Python a `libbz2`, usados pelo módulo `bz2`, substituindo `pkg-config`.

CURSES_CFLAGS

CURSES_LIBS

Sinalizadores de compilador C e vinculador para `libncurses` ou `libncursesw`, usados pelo módulo `curses`, substituindo `pkg-config`.

GDBM_CFLAGS

GDBM_LIBS

Sinalizadores de compilador C e vinculador para `gdbm`.

LIBB2_CFLAGS

LIBB2_LIBS

Sinalizadores de compilador C e vinculador para `libb2` (BLAKE2), usados pelo módulo `hashlib`, substituindo `pkg-config`.

LIBEDIT_CFLAGS

LIBEDIT_LIBS

Sinalizadores de compilador C e vinculador para `libedit`, usados pelo módulo `readline`, substituindo `pkg-config`.

LIBFFI_CFLAGS

LIBFFI_LIBS

Sinalizadores de compilador C e vinculador para `libffi`, usados pelo módulo `ctypes`, substituindo `pkg-config`.

LIBMPDEC_CFLAGS

LIBMPDEC_LIBS

Sinalizadores de compilador C e vinculador para `libmpdec`, usados pelo módulo `decimal`, substituindo `pkg-config`.

> **Nota**
>
> Estas variáveis de ambiente não têm efeito a menos que *`--with-system-libmpdec`* seja especificado.

LIBLZMA_CFLAGS

LIBLZMA_LIBS

Sinalizadores de compilador C e vinculador para `liblzma`, usados pelo módulo `lzma`, substituindo `pkg-config`.

LIBREADLINE_CFLAGS

LIBREADLINE_LIBS

Sinalizadores de compilador C e vinculador para `libreadline`, usados pelo módulo `readline`, substituindo `pkg-config`.

LIBSQLITE3_CFLAGS

LIBSQLITE3_LIBS

Sinalizadores de compilador C e vinculador para `libsqlite3`, usados pelo módulo `sqlite3`, substituindo `pkg-config`.

LIBUUID_CFLAGS

LIBUUID_LIBS

Sinalizadores de compilador C e vinculador para `libuuid`, usados pelo módulo `uuid`, substituindo `pkg-config`.

PANEL_CFLAGS

PANEL_LIBS

Sinalizadores de compilador C e vinculador para PANEL, substituindo `pkg-config`.

Sinalizadores de compilador C e vinculador para `libpanel` ou `libpanelw`, usados pelo módulo `curses.panel`, substituindo `pkg-config`.

TCLTK_CFLAGS

TCLTK_LIBS

Sinalizadores de compilador C e vinculador para TCLTK, substituindo `pkg-config`.;

ZLIB_CFLAGS

ZLIB_LIBS

Sinalizadores de compilador C e vinculador para `libzlib`, usados pelo módulo `gzip`, substituindo `pkg-config`.

3.3.5 Opções de WebAssembly

--with-emscripten-target=[browser|node]

Define o "sabor" de construção para `wasm32-emscripten`.

- `browser` (padrão): pré-carrega stdlib mínima, MEMFS padrão.
- `node`: suporte a NODERAWFS e pthread.

Adicionado na versão 3.11.

--enable-wasm-dynamic-linking

Ativa o suporte de vinculação dinâmica para WASM.

A vinculação dinâmica permite `dlopen`. O tamanho do arquivo executável aumenta devido à eliminação limitada de código morto e recursos adicionais.

Adicionado na versão 3.11.

--enable-wasm-pthreads

Ativa o suporte a pthreads para WASM.

Adicionado na versão 3.11.

3.3.6 Opções de instalação

--prefix=PREFIX

Instala arquivos independentes de arquitetura em PREFIX. No Unix, o padrão é `/usr/local`.

Este valor pode ser recuperado em tempo de execução usando `sys.prefix`.

Como exemplo, pode-se usar `--prefix="$HOME/.local/"` para instalar um Python em seu diretório pessoal (home).

--exec-prefix=EPREFIX

Instala arquivos dependentes de arquitetura no EPREFIX, o padrão é *--prefix*.

Este valor pode ser recuperado em tempo de execução usando `sys.exec_prefix`.

--disable-test-modules

Não constrói nem instala módulos de teste, como o pacote `test` ou o módulo de extensão `_testcapi` (construído e instalado por padrão).

Adicionado na versão 3.10.

--with-ensurepip=[upgrade|install|no]

Seleciona o comando `ensurepip` executado na instalação do Python:

- `upgrade` (padrão): executa o comando `python -m ensurepip --altinstall --upgrade`.
- `install`: executa o comando `python -m ensurepip --altinstall`;
- `no`: não executa ensurepip;

Adicionado na versão 3.6.

3.3.7 Opções de desempenho

Configurar o Python usando `--enable-optimizations --with-lto` (PGO + LTO) é o recomendado para melhor desempenho. O sinalizador experimental `--enable-bolt` também pode ser usado para melhorar o desempenho.

--enable-optimizations

Habilita a otimização guiada por perfil (PGO, do inglês Profile Guided Optimization) usando *PROFILE_TASK* (desabilitado por padrão).

O compilador C Clang requer o programa `llvm-profdata` para PGO. No macOS, o GCC também exige: o GCC é apenas um apelido para o Clang no macOS.

Desabilita também a interposição semântica no libpython se `--enable-shared` e GCC forem usados: adiciona `-fno-semantic-interposition` aos sinalizadores do compilador e do vinculador.

> **Nota**
>
> Durante a construção, você poderá encontrar avisos do compilador sobre a indisponibilidade de dados de perfil para alguns arquivos fonte. Esses avisos são inofensivos, pois apenas um subconjunto do código é exercido durante a aquisição de dados de perfil. Para desativar esses avisos no Clang, suprima-os manualmente adicionando `-Wno-profile-instr-unprofiled` a *CFLAGS*.

Adicionado na versão 3.6.

Alterado na versão 3.10: Usa `-fno-semantic-interposition` no GCC.

PROFILE_TASK

Variável de ambiente usada no Makefile: argumentos de linha de comando do Python para a tarefa de geração de PGO.

Padrão: `-m test --pgo --timeout=$(TESTTIMEOUT)`.

Adicionado na versão 3.8.

Alterado na versão 3.13: A falha da tarefa não é mais ignorada silenciosamente.

--with-lto=[full|thin|no|yes]

Habilita o otimização em tempo de vinculação (LTO, do inglês Link Time Optimization) em qualquer compilação (desabilitado por padrão).

O compilador C Clang requer `llvm-ar` para LTO (`ar` no macOS), bem como um vinculador compatível com LTO (`ld.gold` ou `lld`).

Adicionado na versão 3.6.

Adicionado na versão 3.11: Para usar o recurso ThinLTO, use `--with-lto=thin` no Clang.

Alterado na versão 3.12: Usa ThinLTO como política de otimização padrão no Clang se o compilador aceitar o sinalizador.

--enable-bolt

Habilita o uso do otimizador binário pós-vinculação BOLT (desabilitado por padrão).

BOLT faz parte do projeto LLVM, mas nem sempre está incluído em suas distribuições binárias. Este sinalizador requer que `llvm-bolt` e `merge-fdata` estejam disponíveis.

BOLT ainda é um projeto relativamente novo, então este sinalizador deve ser considerado experimental por enquanto. Como esta ferramenta opera em código de máquina, seu sucesso depende de uma combinação do ambiente de construção + os outros argumentos de configuração de otimização + a arquitetura da CPU, e nem todas as combinações são suportadas. Versões do BOLT anteriores ao LLVM 16 são conhecidas por travar o BOLT em alguns cenários. O uso do LLVM 16 ou mais recente para otimização do BOLT é fortemente incentivado.

As variáveis `BOLT_INSTRUMENT_FLAGS` e `BOLT_APPLY_FLAGS` do **configure** podem ser definidas para substituir o conjunto padrão de argumentos para **llvm-bolt** para instrumentar e aplicar dados BOLT aos binários , respectivamente.

Adicionado na versão 3.12.

BOLT_APPLY_FLAGS

Argumentos para `llvm-bolt` ao criar um binário otimizado com BOLT.

Adicionado na versão 3.12.

BOLT_INSTRUMENT_FLAGS

Argumentos para `llvm-bolt` ao instrumentar binários.

Adicionado na versão 3.12.

--with-computed-gotos

Habilita "gotos" computados no laço de avaliação (habilitado por padrão em compiladores suportados).

--without-mimalloc

Desativa o alocador rápido mimalloc (habilitado por padrão).

Veja também a variável de ambiente *PYTHONMALLOC*.

--without-pymalloc

Desabilita o alocador de memória especializado do Python pymalloc (habilitado por padrão).

Veja também a variável de ambiente *PYTHONMALLOC*.

--without-doc-strings

Desabilita as strings de documentação estática para reduzir o consumo de memória (habilitado por padrão). As strings de documentação definidas em Python não são afetadas.

Não define a macro `WITH_DOC_STRINGS`.

Veja a macro `PyDoc_STRVAR()`.

--enable-profiling

Habilita o perfil de código a nível C com `gprof` (desabilitado por padrão).

--with-strict-overflow

Adiciona `-fstrict-overflow` aos sinalizadores do compilador C (por padrão adicionamos `-fno-strict-overflow`).

3.3.8 Compilação de depuração do Python

Uma compilação de depuração é Python compilada com a opção de configuração *--with-pydebug*.

Efeitos de uma compilação de depuração:

- Exibe todos os avisos por padrão: a lista de filtros de aviso padrão está vazia no módulo `warnings`.
- Adiciona `d` a `sys.abiflags`.

- Adiciona a função `sys.gettotalrefcount()`.
- Adiciona a opção de linha de comando *-X showrefcount*.
- Adiciona a opção de linha de comando *-d* e a variável de ambiente *PYTHONDEBUG* para depurar o analisador sintático.
- Adiciona suporte para a variável `__lltrace__`: habilita o rastreamento de baixo nível no laço de avaliação de bytecode se a variável estiver definida.
- Instala ganchos de depuração nos alocadores de memória para detectar estouro de buffer e outros erros de memória.
- Define as macros `Py_DEBUG` e `Py_REF_DEBUG`.
- Adiciona verificações de tempo de execução: código cercado por `#ifdef Py_DEBUG` e `#endif`. Habilita as asserções `assert(...)` e `_PyObject_ASSERT(...)`: não define a macro `NDEBUG` (veja também a configuração *--with-assertions* opção). Principais verificações de tempo de execução:
  - Adiciona verificações de sanidade nos argumentos da função.
  - Objetos Unicode e int são criados com sua memória preenchida com um padrão para detectar o uso de objetos não inicializados.
  - Garante que as funções que podem limpar ou substituir a exceção atual não sejam chamadas com uma exceção levantada.
  - Verifica se as funções desalocadoras não alteram a exceção atual.
  - O coletor de lixo (função `gc.collect()`) executa algumas verificações básicas na consistência dos objetos.
  - A macro `Py_SAFE_DOWNCAST()` verifica o underflow e o overflow de inteiros ao fazer o downcast de tipos largos para tipos estreitos.

Veja também o Modo de Desenvolvimento do Python e a opção de configuração *--with-trace-refs*.

Alterado na versão 3.8: Construções de lançamento e construções de depuração agora são compatíveis com ABI: definir a macro `Py_DEBUG` não implica mais na macro `Py_TRACE_REFS` (consulte a opção *--with-trace-refs*).

3.3.9 Opções de depuração

--with-pydebug

Construção de depuração do Python: define a macro `Py_DEBUG` (desabilitada por padrão).

--with-trace-refs

Habilita referências de rastreamento para fins de depuração (desabilitado por padrão).

Efeitos:

- Define a macro `Py_TRACE_REFS`.
- Add `sys.getobjects()` function.
- Adiciona a variável de ambiente *PYTHONDUMPREFS*.

A variável de ambiente *PYTHONDUMPREFS* pode ser usada para despejar objetos e contagens de referências ainda ativas na saída do Python.

Objetos alocados estaticamente não são rastreados.

Adicionado na versão 3.8.

Alterado na versão 3.13: Esta construção agora é compatibilidade de ABI com a construção de lançamento e *construção de depuração*.

--with-assertions

Constrói com asserções C habilitadas (o padrão é não): `assert(...);` e `_PyObject_ASSERT(...);`.

Se definido, a macro `NDEBUG` não é definida na variável do compilador *`OPT`*.

Veja também a opção *`--with-pydebug`* (*construção de depuração*) que também habilita asserções.

Adicionado na versão 3.6.

--with-valgrind

Habilita suporte ao Valgrind (o padrão é não).

--with-dtrace

Habilita suporte ao DTrace (o padrão é não).

Veja Instrumentando o CPython com DTrace e SystemTap.

Adicionado na versão 3.6.

--with-address-sanitizer

Habilita o detector de erros de memória AddressSanitizer, `asan` (o padrão é não).

Adicionado na versão 3.6.

--with-memory-sanitizer

Habilita o detector de erros de alocação do MemorySanitizer, `msan` (o padrão é não).

Adicionado na versão 3.6.

--with-undefined-behavior-sanitizer

Habilita o detector de comportamento indefinido UndefinedBehaviorSanitizer, `ubsan` (o padrão é não).

Adicionado na versão 3.6.

--with-thread-sanitizer

Habilita o detector de corrida de dados ThreadSanitizer, `tsan` (o padrão é não).

Adicionado na versão 3.13.

3.3.10 Opções da ligação

--enable-shared

Habilita a construção de uma biblioteca Python compartilhada: `libpython` (o padrão é não).

--without-static-libpython

Não constrói `libpythonMAJOR.MINOR.a` e não instala `python.o` (construído e habilitado por padrão).

Adicionado na versão 3.10.

3.3.11 Opções da biblioteca

--with-libs`='lib1 ...'`

Vincula bibliotecas adicionais (o padrão é não).

--with-system-expat

Constrói o módulo `pyexpat` usando uma biblioteca `expat` instalada (o padrão é não).

--with-system-libmpdec

Constrói o módulo de extensão `_decimal` usando uma biblioteca `mpdecimal` instalada, veja o módulo `decimal` (o padrão é sim).

Adicionado na versão 3.3.

Alterado na versão 3.13: O padrão é usar a biblioteca `mpdecimal` instalada.

Deprecated since version 3.13, will be removed in version 3.15: Uma cópia dos fontes da biblioteca `mpdecimal` não será mais distribuída com Python 3.15.

> **Ver também**
>
> *LIBMPDEC_CFLAGS* e *LIBMPDEC_LIBS*.

--with-readline=readline|editline

Designa uma biblioteca backend para o módulo `readline`.

- readline: Usa readline como o backend.
- editline: Usa editline como o backend.

Adicionado na versão 3.10.

--without-readline

Não constrói o módulo `readline` (construído por padrão).

Não define a macro `HAVE_LIBREADLINE`.

Adicionado na versão 3.10.

--with-libm=STRING

Substitui a biblioteca matemática `libm` por *STRING* (o padrão depende do sistema).

--with-libc=STRING

Substitui a biblioteca C `libc` por *STRING* (o padrão depende do sistema).

--with-openssl=DIR

Raiz do diretório OpenSSL.

Adicionado na versão 3.7.

--with-openssl-rpath=[no|auto|DIR]

Define o diretório da biblioteca de tempo de execução (rpath) para bibliotecas OpenSSL:

- `no` (padrão): não define o rpath;
- `auto`: detecta automaticamente o rpath de *--with-openssl* e `pkg-config`;
- *DIR*: define um rpath explícito.

Adicionado na versão 3.10.

3.3.12 Opções de segurança

--with-hash-algorithm=[fnv|siphash13|siphash24]

Seleciona o algoritmo de hash para usar em `Python/pyhash.c`:

- `siphash13` (padrão);
- `siphash24`;
- `fnv`.

Adicionado na versão 3.4.

Adicionado na versão 3.11: `siphash13` é adicionado e é o novo padrão.

--with-builtin-hashlib-hashes=md5,sha1,sha256,sha512,sha3,blake2

Módulos embutidos de hash

- `md5`;
- `sha1`;

- `sha256`;
- `sha512`;
- `sha3` (com shake);
- `blake2`.

Adicionado na versão 3.9.

--with-ssl-default-suites=[python|openssl|STRING]

Substitui a string dos conjuntos de criptografia padrão do OpenSSL:

- `python` (padrão): use a seleciona preferida do Python;
- `openssl`: mantém inalterados os padrões do OpenSSL;
- *STRING*: usa uma string personalizada

Veja o módulo `ssl`.

Adicionado na versão 3.7.

Alterado na versão 3.10: As configurações `python` e *STRING* também definem TLS 1.2 como versão mínima do protocolo.

--disable-safety

Disable compiler options that are recommended by OpenSSF for security reasons with no performance overhead. If this option is not enabled, CPython will be built based on safety compiler options with no slow down. When this option is enabled, CPython will not be built with the compiler options listed below.

The following compiler options are disabled with `--disable-safety`:

- -fstack-protector-strong: Enable run-time checks for stack-based buffer overflows.
- -Wtrampolines: Enable warnings about trampolines that require executable stacks.

Adicionado na versão 3.14.

--enable-slower-safety

Enable compiler options that are recommended by OpenSSF for security reasons which require overhead. If this option is not enabled, CPython will not be built based on safety compiler options which performance impact. When this option is enabled, CPython will be built with the compiler options listed below.

The following compiler options are enabled with `--enable-slower-safety`:

- -D_FORTIFY_SOURCE=3: Fortify sources with compile- and run-time checks for unsafe libc usage and buffer overflows.

Adicionado na versão 3.14.

3.3.13 Opções do macOS

Veja Mac/README.rst.

--enable-universalsdk

--enable-universalsdk=SDKDIR

Cria uma construção binária universal. *SDKDIR* especifica qual SDK do macOS deve ser usado para executar a construção (o padrão é não).

--enable-framework

--enable-framework=INSTALLDIR

Cria um Python.framework em vez de uma instalação tradicional do Unix. O *INSTALLDIR* opcional especifica o caminho de instalação (o padrão é não).

--with-universal-archs=ARCH

Especifica o tipo de binário universal que deve ser criado. Esta opção só é válida quando *--enable-universalsdk* está definido.

Opções:

- universal2;
- 32-bit;
- 64-bit;
- 3-way;
- intel;
- intel-32;
- intel-64;
- all.

--with-framework-name=FRAMEWORK

Especifica o nome do framework python no macOS válido apenas quando *--enable-framework* está definido (padrão: Python).

--with-app-store-compliance

--with-app-store-compliance=PATCH-FILE

A biblioteca padrão do Python contém strings que são conhecidas por acionar erros de ferramentas de inspeção automatizadas quando enviadas para distribuição pelas App Stores do macOS e do iOS. Se ativada, esta opção aplicará a lista de patches que corrigem a conformidade da app store. Um arquivo de patch personalizado também pode ser especificado. Esta opção está desativada por padrão.

Adicionado na versão 3.13.

3.3.14 Opções do iOS

Veja iOS/README.rst.

--enable-framework=INSTALLDIR

Cria um Python.framework. Ao contrário do macOS, o argumento *INSTALLDIR* que especifica o caminho de instalação é obrigatório.

--with-framework-name=FRAMEWORK

Especifica o nome do framework (padrão: Python).

3.3.15 Opções de compilação cruzada

A compilação cruzada, também conhecida como construção cruzada, pode ser usada para construir Python para outra arquitetura ou plataforma de CPU. A compilação cruzada requer um interpretador Python para a plataforma de construção. A versão do Python para construção deve corresponder à versão do Python da compilação cruzada do host.

--build=BUILD

configura para construir em BUILD, geralmente adivinhado por **config.guess**.

--host=HOST

faz compilação cruzada para construir programas para executar no HOST (plataforma de destino)

--with-build-python=path/to/python

caminho para construir o binário python para compilação cruzada

Adicionado na versão 3.11.

CONFIG_SITE=file

Uma variável de ambiente que aponta para um arquivo com substituições de configuração.

Exemplo de arquivo *config.site*:

```
# config.site-aarch64
ac_cv_buggy_getaddrinfo=no
ac_cv_file__dev_ptmx=yes
ac_cv_file__dev_ptc=no
```

HOSTRUNNER

Programa para executar CPython para a plataforma host para compilação cruzada.

Adicionado na versão 3.11.

Exemplo de compilação cruzada:

```
CONFIG_SITE=config.site-aarch64 ../configure \
    --build=x86_64-pc-linux-gnu \
    --host=aarch64-unknown-linux-gnu \
    --with-build-python=../x86_64/python
```

3.4 Sistema de Construção Python

3.4.1 Arquivos principais do sistema de construção

- `configure.ac` => `configure`;
- `Makefile.pre.in` => `Makefile` (criado por `configure`);
- `pyconfig.h` (criado por `configure`);
- `Modules/Setup`: Extensões C construídas pelo Makefile usando o shell script `Module/makesetup`;

3.4.2 Principais etapas de construção

- Arquivos C (`.c`) são construídos como arquivos objeto (`.o`).
- Uma biblioteca estática `libpython` (`.a`) é criada a partir de arquivos de objetos.
- `python.o` e a biblioteca estática `libpython` estão vinculadas ao programa final `python`.
- Extensões C são construídas pelo Makefile (veja `Modules/Setup`).

3.4.3 Alvos principais do Makefile

make

Na maioria das vezes, ao reconstruir após editar algum código ou atualizar seu checkout do upstream, tudo que você precisa fazer é executar `make`, que (pela semântica do Make) constrói o alvo padrão, o primeiro definido no Makefile. Por tradição (inclusive no projeto CPython) este é geralmente o alvo `all`. O script `configure` expande uma variável `autoconf`, `@DEF_MAKE_ALL_RULE@` para descrever precisamente quais alvos `make all` serão construídos. As três opções são:

- `profile-opt` (configurado com `--enable-optimizations`)
- `build_wasm` (configurado com `--with-emscripten-target`)
- `build_all` (configurado sem usar explicitamente nenhum dos outros)

Dependendo das alterações mais recentes no arquivo fonte, o Make irá reconstruir quaisquer alvos (arquivos objetos e executáveis) considerados desatualizados, incluindo executar `configure` novamente se necessário. As dependências de origem/alvo são muitas e mantidas manualmente, portanto, às vezes, o Make não tem todas as informações

necessárias para detectar corretamente todos os alvos que precisam ser reconstruídos. Dependendo de quais destinos não são reconstruídos, você poderá enfrentar vários problemas. Se você tiver problemas de construção ou teste que você não pode explicar de outra forma, `make clean && make` deve resolver a maioria dos problemas de dependência, às custas de tempos de construção mais longos.

make platform

Constrói o programa `python`, mas não constrói os módulos de extensão da biblioteca padrão. Isto gera um arquivo chamado `platform` que contém uma única linha descrevendo os detalhes da plataforma de construção, por exemplo, `macosx-14.3-arm64-3.12` ou `linux-x86_64-3.13`.

make profile-opt

Constrói Python usando otimização guiada por perfil (PGO). Você pode usar a opção *`--enable-optimizations`* do configure para tornar este o alvo padrão do comando `make` (`make all` ou apenas `make`).

make clean

Remove arquivos construídos.

make distclean

Além do trabalho feito por `make clean`, remove os arquivos criados pelo script configure. `configure` terá que ser executado antes de construir novamente.[1]

make install

Constrói o alvo `all` e instala o Python.

make test

Build the `all` target and run the Python test suite with the `--fast-ci` option without GUI tests. Variables:

- `TESTOPTS`: opções adicionais de linha de comando regrtest.
- `TESTPYTHONOPTS`: opções adicionais de linha de comando do Python.
- `TESTTIMEOUT`: tempo limite em segundos (padrão: 10 minutos).

make ci

This is similar to `make test`, but uses the `-ugui` to also run GUI tests.

Adicionado na versão 3.14.

make buildbottest

Isto é semelhante ao `make test`, mas usa a opção `--slow-ci` e o tempo limite padrão de 20 minutos, em vez da opção `--fast-ci`.

make regen-all

Gera novamente (quase) todos os arquivos gerados. Isso inclui (mas não está limitado a) casos de bytecode e arquivo gerador de analisador sintático. `make regen-stdlib-module-names` e `autoconf` devem ser executados separadamente para os restantes *arquivos gerados*.

[1] `git clean -fdx` é uma maneira ainda mais extrema de "limpar" seu checkout. Ele remove todos os arquivos não conhecidos pelo Git. Ao procurar bugs usando `git bisect`, isso é recomendado entre testes para garantir uma construção completamente limpa. **Use com cuidado**, pois isso excluirá todos os arquivos não registrados no Git, incluindo seu trabalho novo e não confirmado.

3.4.4 Extensões C

Algumas extensões C são construídas como módulos embutidos, como o módulo `sys`. Eles são construídos com a macro `Py_BUILD_CORE_BUILTIN` definida. Módulos embutidos não possuem nenhum atributo `__file__`:

```
>>> import sys
>>> sys
<module 'sys' (built-in)>
>>> sys.__file__
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
AttributeError: module 'sys' has no attribute '__file__'
```

Outras extensões C são construídas como bibliotecas dinâmicas, como o módulo `_asyncio`. Eles são construídos com a macro `Py_BUILD_CORE_MODULE` definida. Exemplo no Linux x86-64:

```
>>> import _asyncio
>>> _asyncio
<module '_asyncio' from '/usr/lib64/python3.9/lib-dynload/_asyncio.cpython-39-x86_
↪64-linux-gnu.so'>
>>> _asyncio.__file__
'/usr/lib64/python3.9/lib-dynload/_asyncio.cpython-39-x86_64-linux-gnu.so'
```

`Modules/Setup` é usado para gerar alvos Makefile para construir extensões C. No início dos arquivos, as extensões C são construídas como módulos embutidos. Extensões definidas após o marcador `*shared*` são construídas como bibliotecas dinâmicas.

As macros `PyAPI_FUNC()`, `PyAPI_DATA()` e `PyMODINIT_FUNC` de `Include/exports.h` são definidas de forma diferente dependendo se a macro `Py_BUILD_CORE_MODULE` está definida:

- Usa `Py_EXPORTED_SYMBOL` se `Py_BUILD_CORE_MODULE` estiver definido
- Do contrário, usa `Py_IMPORTED_SYMBOL`.

Se a macro `Py_BUILD_CORE_BUILTIN` for usada por engano em uma extensão C construída como uma biblioteca compartilhada, sua função `PyInit_xxx()` não será exportada, causando um `ImportError` na importação.

3.5 Sinalizadores do compilador e do vinculador

Opções definidas pelo script `./configure` e variáveis de ambiente e usadas por `Makefile`.

3.5.1 Sinalizadores do pré-processador

CONFIGURE_CPPFLAGS

Valor da variável *CPPFLAGS* passado para o script `./configure`.

Adicionado na versão 3.6.

CPPFLAGS

Sinalizadores de pré-processador C++ / (Objective) C, p.ex. `-Iinclude_dir` se você tiver cabeçalhos em um diretório não padrão *include_dir*.

Tanto *CPPFLAGS* quanto *LDFLAGS* precisam conter o valor do shell para poder construir módulos de extensão usando os diretórios especificados nas variáveis de ambiente.

BASECPPFLAGS

Adicionado na versão 3.4.

PY_CPPFLAGS

Sinalizadores extras de pré-processador adicionados para construir os arquivos de objeto do interpretador.

Padrão: `$(BASECPPFLAGS) -I. -I$(srcdir)/Include $(CONFIGURE_CPPFLAGS) $(CPPFLAGS)`.

Adicionado na versão 3.2.

3.5.2 Sinalizadores do compilador

CC

Comando do compilador C.

Exemplo: `gcc -pthread`.

CXX

Comando do compilador C++.

Exemplo: `g++ -pthread`.

CFLAGS

Sinalizadores do compilador C.

CFLAGS_NODIST

CFLAGS_NODIST é usado para construir o interpretador e extensões C da stdlib. Use-o quando um sinalizador do compilador *não* deve fazer parte de *CFLAGS* depois que o Python estiver instalado (gh-65320).

Em particular, *CFLAGS* não deve conter:

- o sinalizador do compilador `-I` (para definir o caminho de pesquisa para arquivos incluídos). Os sinalizadores `-I` são processadas da esquerda para a direita, e quaisquer sinalizadores em *CFLAGS* terão precedência sobre os sinalizadores `-I` fornecidos pelo usuário e pelo pacote.
- sinalizadores de segurança como `-Werror` porque as distribuições não podem controlar se os pacotes instalados pelos usuários estão em conformidade com esses padrões elevados.

Adicionado na versão 3.5.

COMPILEALL_OPTS

Opções passadas para a linha de comando `compileall` ao construir arquivos PYC em `make install`. Padrão: `-j0`.

Adicionado na versão 3.12.

EXTRA_CFLAGS

Sinalizadores extra do compilador C.

CONFIGURE_CFLAGS

Valor da variável *CFLAGS* passado para o script `./configure`.

Adicionado na versão 3.2.

CONFIGURE_CFLAGS_NODIST

Valor da variável *CFLAGS_NODIST* passado para o script `./configure`.

Adicionado na versão 3.5.

BASECFLAGS

Sinalizadores base do compilador.

OPT

Sinalizadores de otimização.

CFLAGS_ALIASING

Sinalizadores de alias estritos ou não estritos usados para compilar `Python/dtoa.c`.

Adicionado na versão 3.7.

CCSHARED

Sinalizadores de compilador usados para construir uma biblioteca compartilhada.

Por exemplo, `-fPIC` é usado no Linux e no BSD.

CFLAGSFORSHARED

Sinalizadores extras de C adicionados para construir os arquivos de objeto do interpretador.

Padrão: `$(CCSHARED)` quando *`--enable-shared`* é usado, ou uma string vazia caso contrário.

PY_CFLAGS

Padrão: `$(BASECFLAGS) $(OPT) $(CONFIGURE_CFLAGS) $(CFLAGS) $(EXTRA_CFLAGS)`.

PY_CFLAGS_NODIST

Padrão: `$(CONFIGURE_CFLAGS_NODIST) $(CFLAGS_NODIST) -I$(srcdir)/Include/internal`.

Adicionado na versão 3.5.

PY_STDMODULE_CFLAGS

Sinalizadores do C usados para construir os arquivos de objeto do interpretador.

Padrão: `$(PY_CFLAGS) $(PY_CFLAGS_NODIST) $(PY_CPPFLAGS) $(CFLAGSFORSHARED)`.

Adicionado na versão 3.7.

PY_CORE_CFLAGS

Padrão: `$(PY_STDMODULE_CFLAGS) -DPy_BUILD_CORE`.

Adicionado na versão 3.2.

PY_BUILTIN_MODULE_CFLAGS

Sinalizadores do compilador para construir um módulo de extensão de biblioteca padrão como um módulo embutido, como o módulo `posix`.

Padrão: `$(PY_STDMODULE_CFLAGS) -DPy_BUILD_CORE_BUILTIN`.

Adicionado na versão 3.8.

PURIFY

Comando de Purify. Purify é um programa depurador de memória.

Padrão: string vazia (não usada).

3.5.3 Sinalizadores do vinculador

LINKCC

Comando do vinculador usado para construir programas como `python` e `_testembed`.

Padrão: `$(PURIFY) $(CC)`.

CONFIGURE_LDFLAGS

Valor da variável *`LDFLAGS`* passado para o script `./configure`.

Evite atribuir *`CFLAGS`*, *`LDFLAGS`*, etc. para que os usuários possam usá-los na linha de comando para anexar a esses valores sem pisotear os valores predefinidos.

Adicionado na versão 3.2.

LDFLAGS_NODIST

`LDFLAGS_NODIST` é usado da mesma maneira que *`CFLAGS_NODIST`*. Use-o quando um sinalizador de vinculador *não* fizer parte de *`LDFLAGS`* depois que o Python estiver instalado (gh-65320).

Em particular, *`LDFLAGS`* não deve conter:

- o sinalizador do compilador `-L` (para definir o caminho de pesquisa para arquivos incluídos). Os sinalizadores `-L` são processadas da esquerda para a direita, e quaisquer sinalizadores em *`LDFLAGS`* terão precedência sobre os sinalizadores `-L` fornecidos pelo usuário e pelo pacote.

CONFIGURE_LDFLAGS_NODIST

Valor da variável *LDFLAGS_NODIST* passado para o script `./configure`.

Adicionado na versão 3.8.

LDFLAGS

Sinalizadores do vinculador, p.ex. `-Llib_dir`, se você tiver bibliotecas em um diretório não padrão *lib_dir*.

Tanto *CPPFLAGS* quanto *LDFLAGS* precisam conter o valor do shell para poder construir módulos de extensão usando os diretórios especificados nas variáveis de ambiente.

LIBS

Sinalizadores do vinculador para passar bibliotecas para o vinculador ao vincular o executável Python.

Exemplo: `-lrt`.

LDSHARED

Comando para construir uma biblioteca compartilhada.

Padrão: `@LDSHARED@ $(PY_LDFLAGS)`.

BLDSHARED

Comando para construir a biblioteca compartilhada `libpython`.

Padrão: `@BLDSHARED@ $(PY_CORE_LDFLAGS)`.

PY_LDFLAGS

Padrão: `$(CONFIGURE_LDFLAGS) $(LDFLAGS)`.

PY_LDFLAGS_NODIST

Padrão: `$(CONFIGURE_LDFLAGS_NODIST) $(LDFLAGS_NODIST)`.

Adicionado na versão 3.8.

PY_CORE_LDFLAGS

Sinalizadores de vinculador usados para construir os arquivos de objeto do interpretador.

Adicionado na versão 3.8.

CAPÍTULO 4

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 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.

As specified in **PEP 11**, a Python release only supports a Windows platform while Microsoft considers the platform under extended support. This means that Python 3.14 supports Windows 10 and newer. If you require Windows 7 support, please install Python 3.8. If you require Windows 8.1 support, please install Python 3.12.

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 e superior, 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.

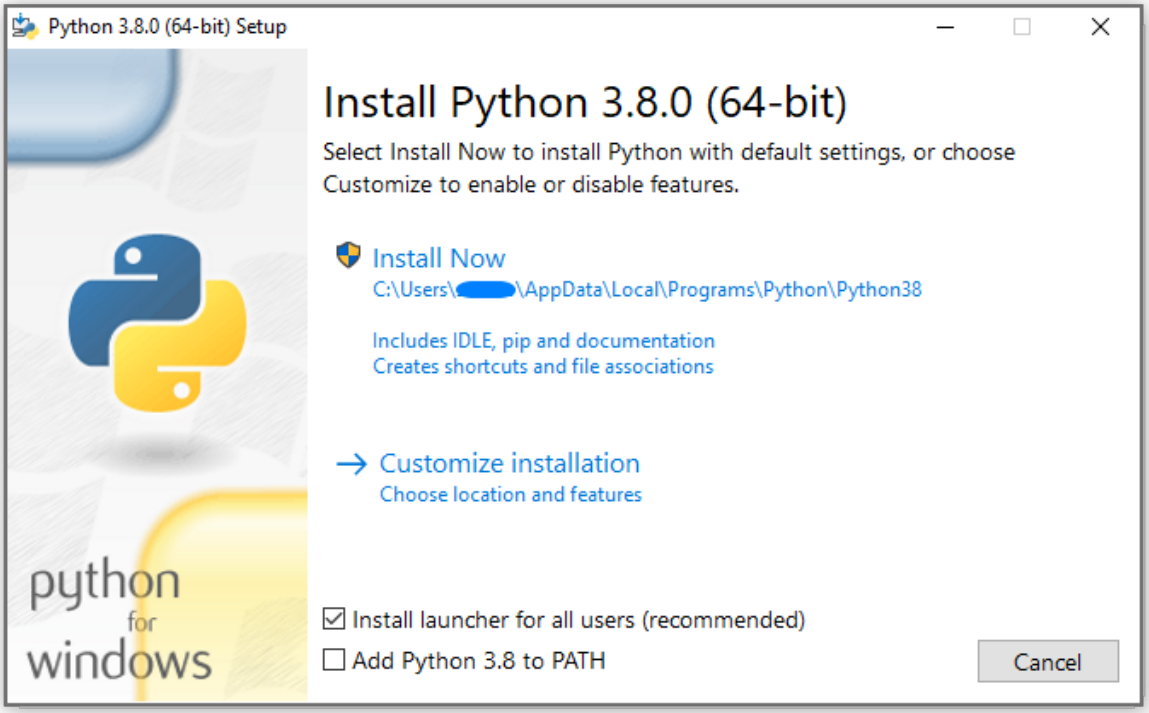
O pacote embutível é um pacote mínimo de Python adequado para ser incorporado em uma aplicação maior.

4.1 O instalador completo

4.1.1 Etapas de instalação

Quatro instaladores Python 3.14 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

4.1.2 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.

4.1.3 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.

As seguintes opções (encontradas executando o instalador com `/?`) podem ser passadas para o instalador:

Nome	Descrição
/passive	para exibir o progresso sem exigir interação do usuário
/quiet	para instalar/desinstalar sem exibir nenhuma UI
/simple	para evitar a personalização do usuário
/uninstall	para remover o Python (sem exigir confirmação)
/layout [diretório]	para pré-baixar todos os componentes
/log [nome-de-arquivo]	para especificar a localização dos arquivos de log

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\Python\PythonXY` ou `%LocalAppData%\Programs\Python\PythonXY-32` ou `%LocalAppData%\Programs\Python\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)	Anexa os diretórios de instalação e Scripts ao início de `PATH` e adiciona `.PY` a `PATHEXT`	0
AppendPath	Anexa os diretórios de instalação e Scripts ao final de `PATH` e adiciona `.PY` a `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	Instala cabeçalhos e bibliotecas do desenvolvedor. Omitir isso pode levar a uma instalação inutilizável.	1
Include_exe	Instala `python.exe` e arquivos relacionados. Omitir isso pode levar a uma instalação inutilizável.	1
Include_launcher	Instalar *Inicializador Python para Windows*.	1
InstallLauncherAllUsers	Instala o iniciador para todos os usuários. Também requer `Include_launcher` para ser definido como 1	1
Include_lib	Instala a biblioteca padrão e os módulos de extensão. Omitir isso pode levar a uma instalação inutilizável.	1
Include_pip	Instale o pacote pip e setuptools	1
Include_symbols	Instala 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	0

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-3.9.0.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-3.9.0.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>
```

4.1.4 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-3.9.0.exe` pelo nome real do seu instalador, e de criar layouts nos respectivos diretórios para evitar colisão entre arquivos com o mesmo nome.

```
python-3.9.0.exe /layout [diretório alvo opcional]
```

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

4.1.5 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.

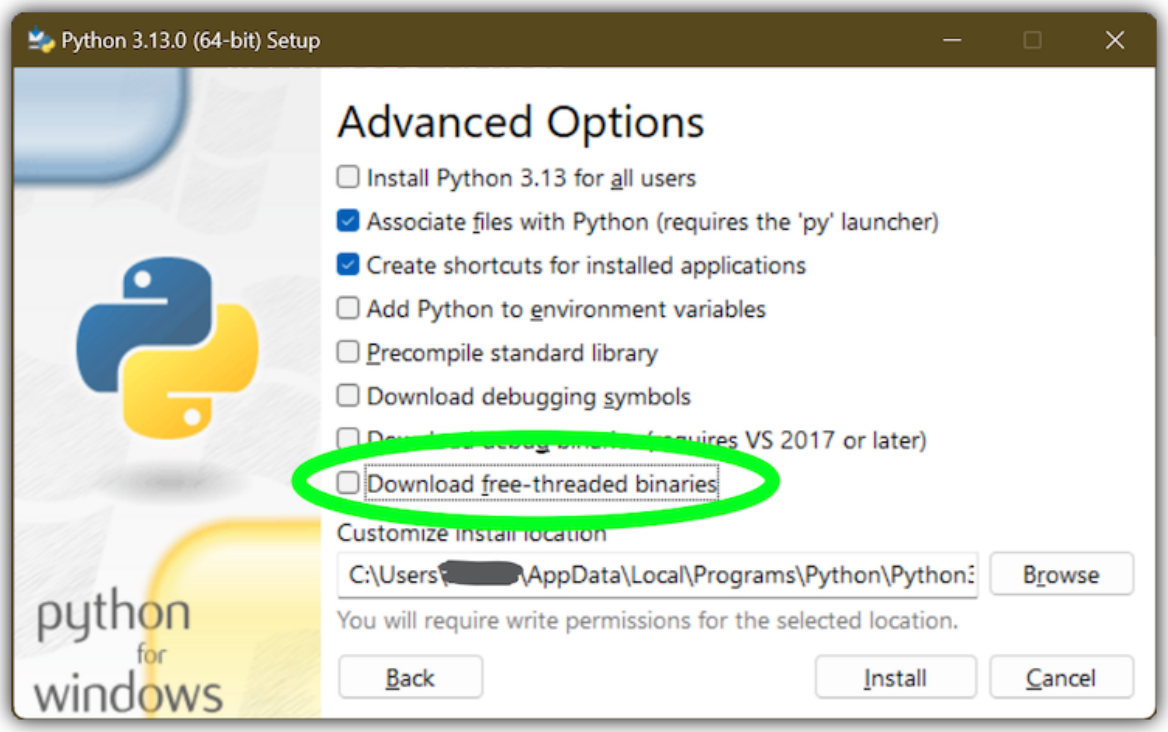
"Uninstall" irá remover o Python completamente, com a exceção do *Inicializador Python para Windows*, que tem seu próprio registro nos Programas e Recursos.

4.1.6 Instalando binários com threads livres

Adicionado na versão 3.13: (Experimental)

> **Nota**
>
> Tudo descrito nesta seção é considerado experimental e espera-se que mude em versões futuras.

Para instalar binários pré-construídos com recurso de threads livres habilitado (veja **PEP 703**), você deve selecionar "Customize installation" (Instalação personalizada). A segunda página de opções inclui a caixa de seleção "Download free-threaded binaries" (Baixar binários de threads livres).



Selecionar esta opção fará o download e instalará binários adicionais no mesmo local da instalação principal do Python. O executável principal é chamado `python3.13t.exe`, e outros binários recebem um sufixo `t` ou um sufixo ABI completo. Arquivos de origem do Python e dependências de terceiros empacotadas são compartilhadas com a instalação principal.

A versão com threads livres é registrada como uma instalação regular do Python com a tag `3.13t` (com um sufixo `-32` ou `-arm64` como normal para essas plataformas). Isso permite que as ferramentas o descubram e que o *Inicializador Python para Windows* ofereça suporte a `py.exe -3.13t`. Observe que o inicializador interpretará `py.exe -3` (ou um shebang `python3`) como "a instalação 3.x mais recente", que preferirá os binários com threads livres aos normais, enquanto `py.exe -3.13` não. Se você usar o estilo curto da opção, talvez prefira não instalar os binários com threads livres neste momento.

Para especificar a opção de instalação na linha de comando, use `Include_freethreaded=1`. Veja *Instalando Sem Download* para instruções sobre como baixar preventivamente os binários adicionais para instalação offline. As opções para incluir símbolos de depuração e binários também se aplicam às construções com threads livres.

Binários com threads livres também estão disponíveis *em nuget.org*.

4.2 O pacote Microsoft Store

Adicionado 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 3.14" 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 3.14). 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.

4.2.1 Problemas conhecidos

Redirecionamento de dados locais, registro e caminhos temporários

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.

No tempo de execução, o Python usará uma cópia privada de pastas conhecidas do Windows e do registro. Por exemplo, se a variável de ambiente `%APPDATA%` for `c:\Users\<user>\AppData\`, ao gravar em `C:\Users\<user>\AppData\Local` gravará em `C:\Users\<user>\AppData\Local\Packages\PythonSoftwareFoundation.Python.3.8_qbz5n2kfra8p0\LocalCache\Local\`

Ao ler arquivos, o Windows retornará o arquivo da pasta privada ou, se não existir, o diretório real do Windows. Por exemplo, ler `C:\Windows\System32` retorna o conteúdo de `C:\Windows\System32` mais o conteúdo de `C:\Program Files\WindowsApps \nome_pacote\VFS\SystemX86`.

Você pode encontrar o caminho real de qualquer arquivo existente usando `os.path.realpath()`:

```
>>> import os
>>> test_file = 'C:\\Users\\exemplo\\AppData\\Local\\test.txt'
>>> os.path.realpath(test_file)
```

(continua na próxima página)

(continuação da página anterior)

```
'C:\\Users\\exemplo\\AppData\\Local\\Packages\\PythonSoftwareFoundation.Python.3.8_
↪qbz5n2kfra8p0\\LocalCache\\Local\\test.txt'
```

Ao gravar no Registro do Windows, existem os seguintes comportamentos:

- A leitura de `HKLM\\Software` é permitida e os resultados são mesclados com o arquivo `registry.dat` no pacote.
- Gravar em `HKLM\\Software` não é permitido se a chave/valor correspondente existir, ou seja, modificar chaves existentes.
- Gravar em `HKLM\\Software` é permitido desde que uma chave/valor correspondente não exista no pacote e o usuário tenha as permissões de acesso corretas.

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

4.3 Os pacotes nuget.org

Adicionado 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:

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

# Com -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 bits, www.nuget.org/packages/pythonx86 para a versão 32 bits e www.nuget.org/packages/pythonarm64 para a versão ARM64

4.3.1 Pacotes com threads livres

Adicionado na versão 3.13: (Experimental)

> **Nota**
>
> Tudo descrito nesta seção é considerado experimental e espera-se que mude em versões futuras.

Pacotes contendo binários com threads livres são chamados python-freethreaded para a versão de 64 bits, pythonx86--freethreaded para a versão de 32 bits e pythonarm64-freethreaded para a versão ARM64. Esses dois pacotes contêm os pontos de entrada `python3.13t.exe` e `python.exe`, os quais são executados com threads livres.

4.4 O pacote embutível

Adicionado 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.

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.

4.4.1 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 ar-

mazenados em outras localidades já que há uma oportunidade de especificar o caminho de busca antes de executar a aplicação.

4.4.2 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.

4.5 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`.

Enthought Deployment Manager
: "A próxima geração de ambiente Python e gerenciador de pacotes".

  Anteriormente, a Enthought fornecia Canopy, mas este chegou ao fim de vida em 2016.

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.

4.6 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*.

4.6.1 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 3.9;%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, portanto, você não deve configurá-la permanentemente, a menos que os caminhos listados incluam apenas código compatível com todas as versões instaladas do Python.

Ver também

https://docs.microsoft.com/en-us/windows/win32/procthread/environment-variables
Visão geral das variáveis de ambiente no Windows

https://docs.microsoft.com/en-us/windows-server/administration/windows-commands/set_1
O comando `set`, para modificar temporariamente as variáveis de ambiente

https://docs.microsoft.com/en-us/windows-server/administration/windows-commands/setx
O comando `setx`, para modificar permanentemente as variáveis de ambiente

4.6.2 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 3.9
```

4.7 Modo UTF-8

Adicionado 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.getencoding()`).

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 do Python para alterar a codificação de texto padrão para UTF-8. Você pode habilitar Modo UTF-8 do Python 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 do Python está habilitado, você ainda pode usar a codificação do sistema (a página de código ANSI) 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* (veja a **PEP 529** para detalhes).

4.8 Inicializador Python para Windows

Adicionado 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**.

4.8.1 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, 3.7 e 3.14) você deve ter notado que o Python 3.14 foi iniciado – para iniciar o Python 3.7, use o comando:

```
py -3.7
```

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

```
py -2
```

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.
```

O comando:

```
py --list
```

exibe a(s) versão(ões) atualmente instalada(s) do Python.

O argumento `-x.y` é a forma abreviada do argumento `-V:Empresa/Tag`, que permite selecionar um tempo de execução Python específico, incluindo aqueles que podem ter vindo de algum lugar diferente de python.org. Qualquer tempo de execução registrado seguindo a **PEP 514** será detectável. O comando `--list` lista todos os tempos de execução disponíveis usando o formato `-V:`.

Ao usar o argumento `-V:`, especificar a Empresa limitará a seleção aos tempos de execução desse provedor, enquanto especificar apenas a Tag selecionará de todos os provedores. Observe que omitir a barra implica uma tag:

```
# Seleciona qualquer runtime sinalizado como '3.*'
py -V:3

# Seleciona qualquer runtime lançado como 'PythonCore'
py -V:PythonCore/

# Seleciona o runtime Python 3 mais recente de PythonCore
py -V:PythonCore/3
```

A forma abreviada do argumento (`-3`) só seleciona a partir de versões principais do Python, e não de outras distribuições. No entanto, a forma mais longa (`-V:3`) selecionará qualquer um.

A Empresa é correspondida na string completa, sem distinção entre maiúsculas e minúsculas. A Tag corresponde a uma string completa ou a um prefixo, desde que o próximo caractere seja um ponto ou um hífen. Isso permite que `-V:3.1` corresponda a `3.1-32`, mas não a `3.10`. As tags são classificadas usando ordenação numérica (`3.10` é mais recente que `3.1`), mas são comparadas usando texto (`-V:3.01` não corresponde a `3.1`).

Ambientes virtuais

Adicionado 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("olá do 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. Presumindo que você tem o Python 3.7 instalado, tente mudar a primeira linha para `#! python3.7` e você deve ver as informações da versão 3.7 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.

4.8.2 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`
- `/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 ou um ambiente virtual ativo 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 bits pode ser solicitada adicionando "-32" após a menor versão. Isto é, `/usr/bin/python3.7-32` irá solicitar o uso do python 3.7 32-bit. Se um ambiente virtual estiver ativo, a versão será ignorada e o ambiente será usado.

Adicionado 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`).

Alterado na versão 3.11: O sufixo "-64" foi descontinuado e agora implica "qualquer arquitetura que não seja comprovadamente i386/32 bits". Para solicitar um ambiente específico, use o novo argumento `-V:`*`TAG`* com a tag completa.

Alterado na versão 3.13: Comandos virtuais que referenciam `python` agora preferem um ambiente virtual ativo em vez de pesquisar `PATH`. Isso lida com casos em que o shebang especifica `/usr/bin/env python3`, mas `python3.exe` não está presente no ambiente ativo.

A forma `/usr/bin/env` da linha shebang tem mais uma propriedade especial. Antes de procurar por interpretadores Python instalados, este formulário pesquisará o executável `PATH` por um executável Python correspondente ao nome fornecido como o primeiro argumento. Isso corresponde ao comportamento do programa Unix `env`, que executa uma pesquisa `PATH`. Se um executável correspondente ao primeiro argumento após o comando `env` não puder ser encontrado, mas o argumento começar com `python`, ele será tratado conforme descrito para os outros comandos virtuais. A variável de ambiente `PYLAUNCHER_NO_SEARCH_PATH` pode ser definida (para qualquer valor) para ignorar esta pesquisa de `PATH`.

As linhas Shebang que não correspondem a nenhum desses padrões são procuradas na seção `[commands]` do arquivo *.INI* do lançador. Isso pode ser usado para lidar com certos comandos de uma maneira que faça sentido para o seu sistema. O nome do comando deve ser um único argumento (sem espaços no executável shebang) e o valor substituído é o caminho completo para o executável (argumentos adicionais especificados no .INI serão citados como parte do nome do arquivo).

```
[commands]
/bin/xpython=C:\Program Files\XPython\python.exe
```

Quaisquer comandos não encontrados no arquivo .INI são tratados como caminhos executáveis do **Windows** que são absolutos ou relativos ao diretório que contém o arquivo de script. Isso é uma conveniência para scripts somente do Windows, como aqueles gerados por um instalador, pois o comportamento não é compatível com shells de estilo Unix. Esses caminhos podem ser citados e podem incluir vários argumentos, após os quais o caminho para o script e quaisquer argumentos adicionais serão anexados.

4.8.3 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`

4.8.4 Personalização

Personalização via arquivos INI

Dois arquivos .ini serão pesquisados pelo inicializador - `py.ini` no diretório de dados de aplicações do usuário atual (`%LOCALAPPDATA%` ou `$env:LocalAppData`) 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.
- O comando `python3.7` não vai 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=3.7-32`, o comando `python` irá usar a implementação 32 bits do 3.7 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=3.7`, os comandos `python` e `python3` irão ambos usar especificamente 3.7

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=3.7` é o equivalente ao arquivo INI contendo:

```
[defaults]
python=3.7
```

- Configurar `PY_PYTHON=3` e `PY_PYTHON3=3.7` é o equivalente ao arquivo INI contendo:

```
[defaults]
python=3
python3=3.7
```

4.8.5 Diagnósticos

Se uma variável de ambiente `PYLAUNCHER_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. Destina-se principalmente para testes e depuração.

4.8.6 Teste de simulação

Se uma variável de ambiente `PYLAUNCHER_DRYRUN` for definida (para qualquer valor), o inicializador emitirá o comando que ele teria executado, mas não iniciará o Python de fato. Isso pode ser útil para ferramentas que desejam usar o inicializador para detectar e então iniciar o Python diretamente. Observe que o comando escrito na saída padrão é sempre codificado usando UTF-8 e pode não ser renderizado corretamente no console.

4.8.7 Instalar sob demanda

Se uma variável de ambiente `PYLAUNCHER_ALLOW_INSTALL` for definida (para qualquer valor), e a versão do Python solicitada não estiver instalada, mas estiver disponível na Microsoft Store, o inicializador tentará instalá-la. Isso pode exigir interação do usuário para ser concluído, e você pode precisar executar o comando novamente.

Uma variável adicional `PYLAUNCHER_ALWAYS_INSTALL` faz com que o inicializador sempre tente instalar o Python, mesmo se for detectado. Isso é destinado principalmente para teste (e deve ser usado com `PYLAUNCHER_DRYRUN`).

4.8.8 Códigos de retorno

Os seguintes códigos de saída podem ser retornados pelo inicializador do Python. Infelizmente, não há como distingui-los do código de saída do próprio Python.

Os nomes dos códigos são como usados nas fontes e são apenas para referência. Não há como acessá-los ou resolvê-los além de ler esta página. As entradas são listadas em ordem alfabética de nomes.

Nome	Valor	Descrição
RC_BAD_VENV_CFG	107	Um `pyvenv.cfg` foi encontrado, mas está corrompido.
RC_CREATE_PROCES	101	Falha ao iniciar o Python.
RC_INSTALLING	111	Uma instalação foi iniciada, mas o comando precisará ser executado novamente após sua conclusão.
RC_INTERNAL_ERRO	109	Erro inesperado. Por favor, relate um bug.
RC_NO_COMMANDL	108	Não é possível obter a linha de comando do sistema operacional.
RC_NO_PYTHON	103	Não foi possível localizar a versão solicitada.
RC_NO_VENV_CFG	106	Um `pyvenv.cfg` era necessário, mas não foi encontrado.

4.9 Encontrando módulos

Estas notas complementam a descrição em sys-path-init com notas detalhadas do Windows.

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 estiver carregando `python3.dll` ou `python37.dll` em seu próprio executável, defina explicitamente `PyConfig.module_search_paths` antes de `Py_InitializeFromConfig()`.
- 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`.

Alterado na versão 3.6: 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.

4.10 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.

4.10.1 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) user interfaces

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 e COM
by David and Paul Boddie

4.10.2 cx_Freeze

cx_Freeze 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.

4.11 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, 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.

4.12 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 não é mais suportado desde o Python 3 (se é que foi em algum momento).
- O instalador do Cygwin oferece instalar o interpretador do Python também

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

CAPÍTULO 5

Usando Python em macOS

Este documento tem o objetivo de fornecer uma visão geral dos comportamentos específicos do macOS que você deve conhecer para começar a usar o Python em computadores Mac. O Python em um Mac com macOS é muito semelhante ao Python em outras plataformas derivadas do Unix, mas há algumas diferenças na instalação e alguns recursos.

Há várias maneiras de obter e instalar o Python para macOS. Versões pré-construídas das versões mais recentes do Python estão disponíveis em vários distribuidores. Grande parte deste documento descreve o uso dos Pythons fornecidos pela equipe de lançamento do CPython, disponíveis para download no site python.org. Consulte *Distribuições alternativas* para conhecer outras opções.

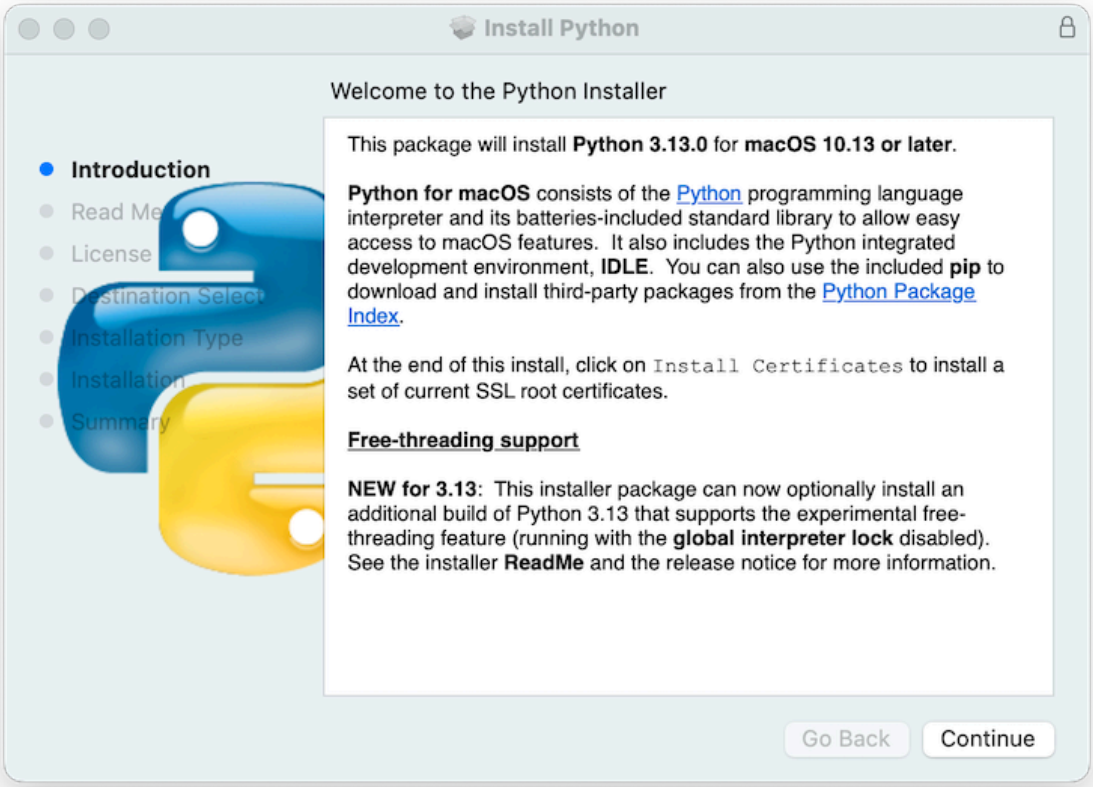
5.1 Usando Python para macOS do `python.org`

5.1.1 Etapas de instalação

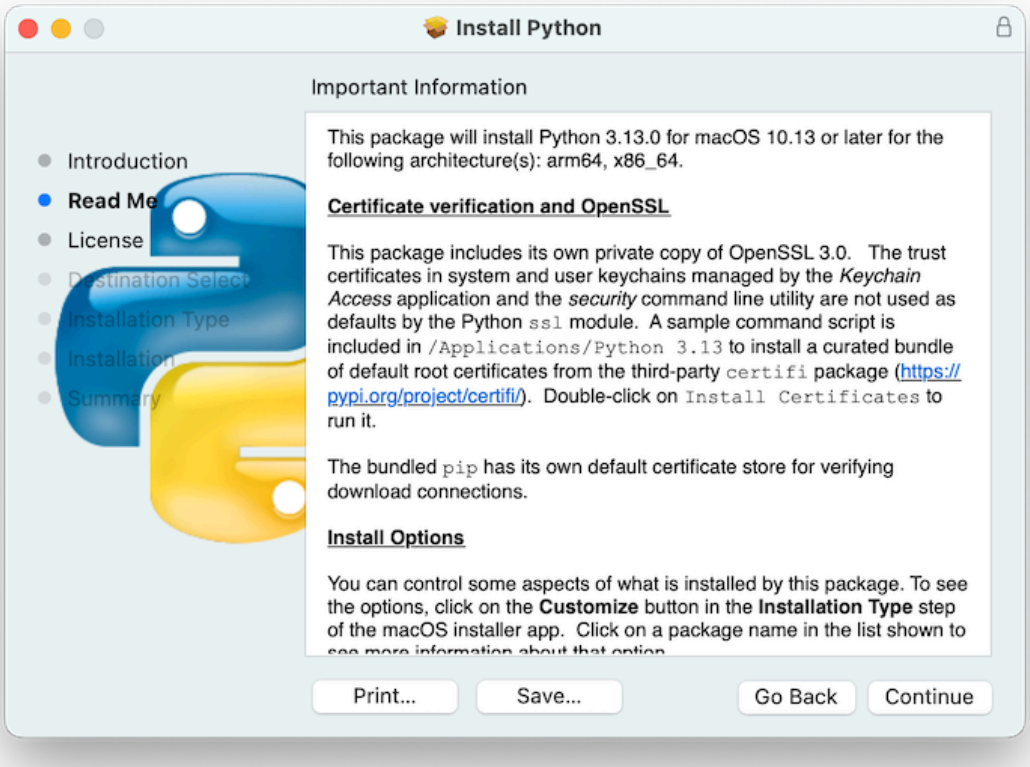
Para as versões atuais de Python (exceto as com status `security`), a equipe de lançamento produz um pacote de instalação **Python para macOS** em cada novo lançamento. Uma lista dos instaladores disponíveis está disponível aqui. Recomendamos usar a versão suportada mais recente de Python sempre que possível. Os instaladores atuais fornecem uma versão de Python como binário universal 2 que é executado nativamente em todos os Macs (Apple Silicon e Intel) compatíveis com uma ampla gama de versões do macOS, atualmente do **macOS 10.13 High Sierra** em diante.

O arquivo baixado é um arquivo padrão do instalador de pacotes do macOS (`.pkg`). As informações de integridade do arquivo (soma de verificação, tamanho, assinatura de Sigstore, etc.) de cada arquivo estão inclusas na página de download dos lançamentos. Os pacotes de instalação e seus conteúdos são assinados e autenticados com certificados Apple Developer ID `Python Software Foundation` para atender aos requisitos do macOS Gatekeeper.

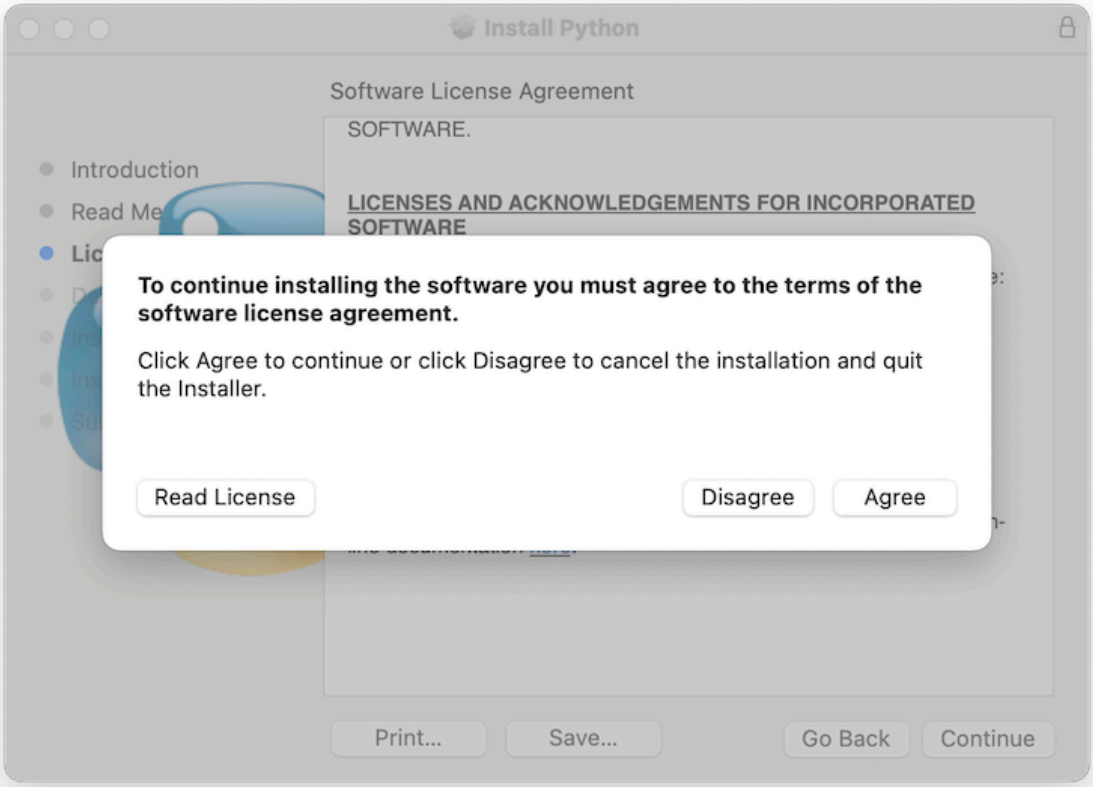
Para uma instalação padrão, clique duas vezes no arquivo instalador de pacotes baixado. Isso deverá abrir o aplicativo padrão de instalação do macOS e exibir o primeiro de vários passos de instalação.



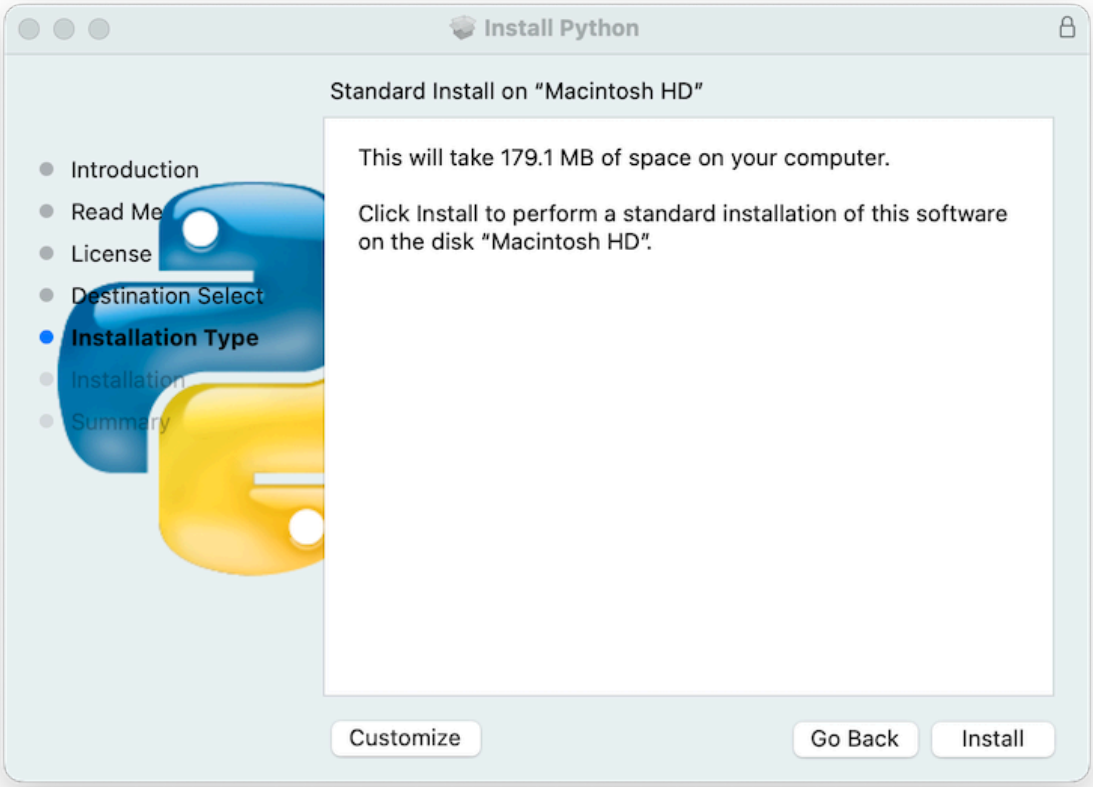
Ao clicar no botão **Continuar**, é exibido o **Read Me** desse instalador. Além de outras informações importantes, o **Read Me** documenta qual versão de Python será instalada e em quais versões do macOS ele é compatível. Talvez você precise rolar a tela para ler o arquivo inteiro. Por padrão, esse **Read Me** também será instalado em `/Applications/Python 3.13/` e estará disponível para leitura a qualquer momento.



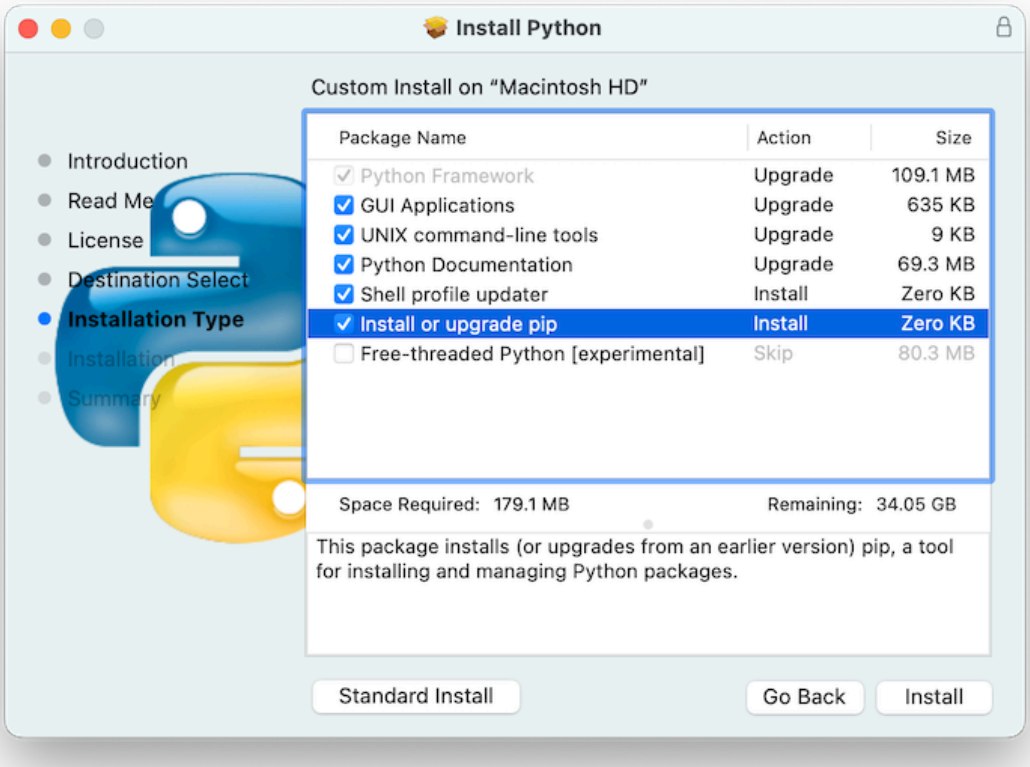
Ao clicar em **Continuar**, o instalador exibe a licença do Python e de outros softwares incluídos. Você precisará **Concordar** com os termos da licença antes de prosseguir para a próxima etapa. Esse arquivo de licença também será instalado e estará disponível para leitura a qualquer momento.



Após aceitar os termos da licença, a próxima etapa será a tela **Tipo de instalação**. Para a maioria dos usos, o conjunto padrão de operações de instalação é adequado.



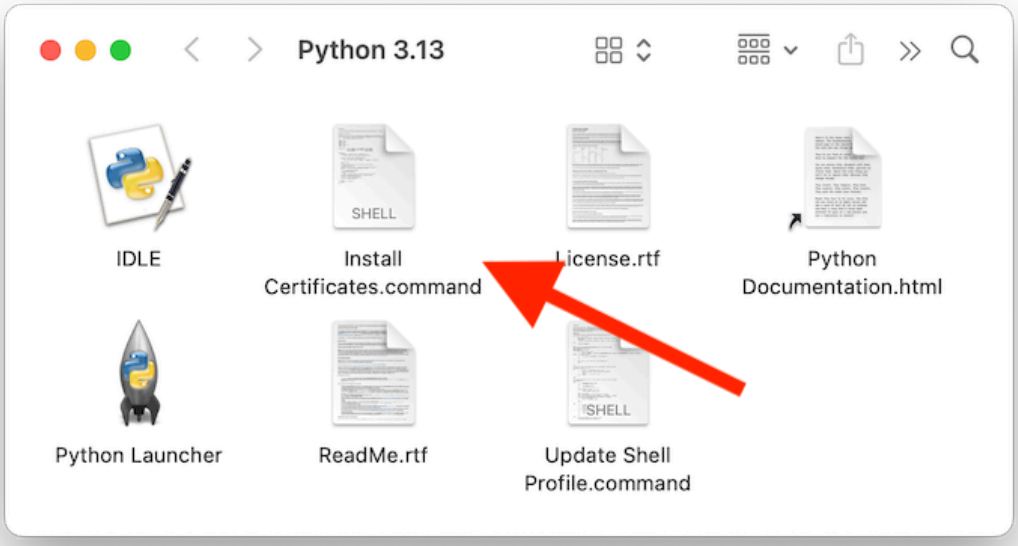
Ao pressionar o botão **Personalizar**, você pode optar por omitir ou selecionar determinados componentes de pacote do instalador. Clique em cada nome de pacote para ver uma descrição do que ele instala. Para também instalar o suporte para o recurso experimental opcional de threads livres, consulte *Instalando binários com threads livres*.



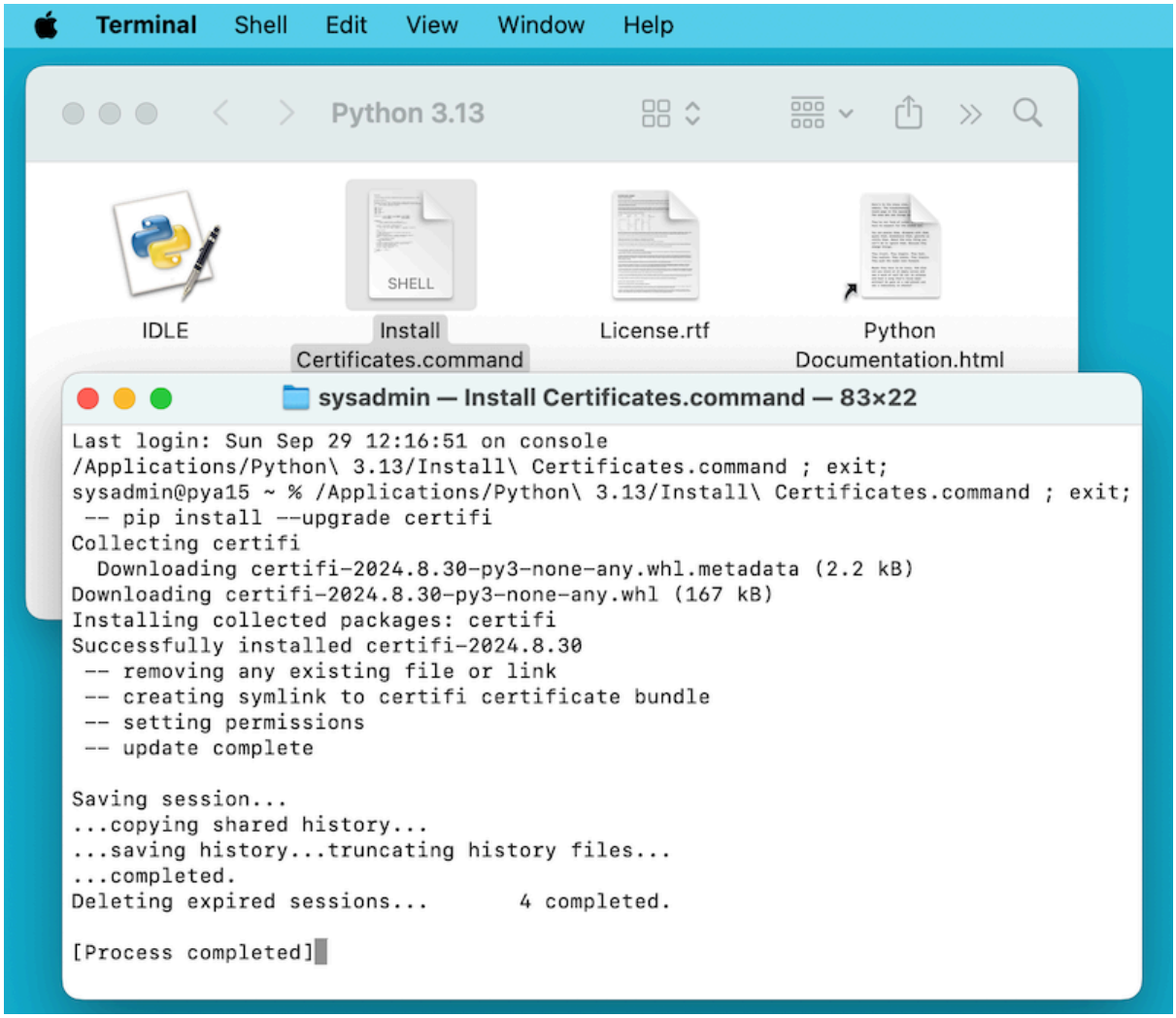
Em ambos os casos, clicar em **Instalar** iniciará o processo de instalação solicitando a permissão para instalar um novo software. É necessário um nome de usuário do macOS com privilégio de `Administrador`, pois o Python instalado estará disponível para todos os usuários do Mac.

Quando a instalação for concluída, a janela **Resumo** será exibida.

Clique duas vezes no ícone ou arquivo **`Install Certificates.command`** na janela `/Applications/Python 3.13/` para concluir a instalação.



Isso abrirá uma janela temporária do console **`Terminal`** que usará o novo Python para baixar e instalar certificados raiz SSL para seu uso.



Se `Successfully installed certifi` e `update complete` aparecerem na janela do terminal, a instalação estará concluída. Feche essa janela do terminal e a janela do instalador.

Uma instalação do padrão incluirá:

- Uma pasta `Python 3.13` em sua pasta `Applications`. Aqui você encontra o **`IDLE`**, o ambiente de desenvolvimento que é parte das distribuições oficiais do Python; e o **`Python Launcher`**, que executa scripts Python quando são clicados duas vezes dentro do Finder .
- Uma framework `/Library/Frameworks/Python.framework`, que inclui o executável Python e suas bibliotecas. O instalador adiciona esse local ao seu *path* do console. Para desinstalar o Python, você pode remover esses três itens. É um link simbólico para o executável Python armazenado na pasta `/usr/local/bin/`.

Nota

As versões recentes do macOS incluem um comando **`python3`** em `/usr/bin/python3` que aponta para uma versão geralmente mais antiga e incompleta de Python fornecida por e para uso pelas ferramentas de desenvolvimento da Apple, **`Xcode`** ou o **`Command Line Tools for Xcode`**. Você nunca deve modificar ou tentar excluir essa instalação por ser controlada pela Apple e é usada por softwares da Apple ou de terceiros. Se você optar por instalar uma versão mais recente do Python a partir de `python.org`, você terá duas instalações diferentes, mas funcionais, de Python no computador que podem coexistir. As opções do instalador padrão devem garantir que o comando **`python3`** seja usado em vez do **`python3`** do sistema.

5.1.2 Como executar um script Python

Há duas maneiras de invocar o interpretador Python. Se você estiver familiarizado com o uso de um console Unix em terminal, você pode invocar `python3.13` ou `python3`, opcionalmente seguido por uma ou mais opções da linha de comando (descritas em *Linha de comando e ambiente*). O tutorial de Python também tem uma seção útil sobre o uso interativo do Python a partir de um console.

Você também pode invocar o interpretador por um ambiente de desenvolvimento integrado (IDE). O idle é um editor e ambiente de desenvolvimento básico incluído com a distribuição padrão de Python. **IDLE** inclui um menu Ajuda que te permite acessar a documentação de Python. Se você for completamente iniciante em Python, você pode ler o tutorial de introdução na documentação.

Há muitos outros editores e IDEs disponíveis. Consulte *Editores e IDEs* para mais informações.

Para executar um script Python na janela do terminal, você pode invocar o interpretador com o nome do arquivo de script:

```
python3.13 myscript.py
```

Para executar seu script a partir do Finder, você pode:

- Arrastá-lo para o **Python Launcher**.
- Selecione **Python Launcher** como a aplicação padrão para abrir seu script (ou qualquer script `.py`) na janela de informações Finder Info, e clique duas vezes nele. O **Python Launcher** tem várias preferências para controlar como o script é iniciado. Você pode arrastar opções para alterá-las por uma invocação, ou usar o menu `Preferências` para alterar configurações globalmente.

Saiba que executar o script diretamente do Finder pode produzir resultados diferentes daqueles obtidos em uma janela de terminal, pois o script não será executado no ambiente normal do console, incluindo qualquer configuração de variáveis de ambiente nos perfis do console. E, como em qualquer outro script ou programa, certifique-se do que está prestes a executar.

5.2 Distribuições alternativas

Além do instalador padrão do `python.org` para macOS, existem distribuições de terceiros para macOS que podem incluir funcionalidades adicionais. Algumas distribuições populares e seus recursos-chave são:

ActivePython
: Instalador com compatibilidade multiplataforma, documentação

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

Homebrew
: Gerenciador de pacotes para macOS que inclui várias versões do Python e muitos pacotes Python de terceiros (incluindo numpy, scipy e pandas).

MacPorts
: Outro gerenciador de pacotes para macOS, incluindo várias versões de Python e muitos pacotes Python de terceiros. Pode incluir versões pré-construídas do Python e muitos pacotes para versões mais antigas do macOS.

Note que distribuições podem não incluir versões mais recentes de Python ou de outras bibliotecas, e não são mantidas ou providas pela equipe do Python.

5.3 Instalando pacotes adicionais ao python

Consulte o Guia de Usuário para Empacotamento de Python para mais informações.

5.4 Programação de GUI

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

O kit de ferramentas GUI padrão de Python é o módulo `tkinter`, baseado no kit de ferramentas multiplataforma Tk (https://www.tcl.tk). Uma versão nativa do Tk para macOS está incluída no instalador.

O PyObjC é uma ligação em Python para o Objective-C/Cocoa da Apple framework. Informações sobre o PyObjC estão disponíveis em pyobjc.

Há vários kits de ferramentas GUI alternativas disponíveis para o macOS, incluindo:

- PySide: Ligações oficiais em Python ao kit de ferramentas Qt GUI .
- PyQt: Ligações alternativas em Python ao Qt.
- Kivy: Um kit de ferramentas GUI multiplataforma que dá suporte a plataformas de desktop e dispositivos móveis.
- Toga: Parte do BeeWare Project; dá suporte a aplicativos para desktop, dispositivos móveis, Web e console.
- wxPython: Um kit de ferramentas multiplataforma que dá suporte a sistemas operacionais desktop.

5.5 Tópicos Avançados

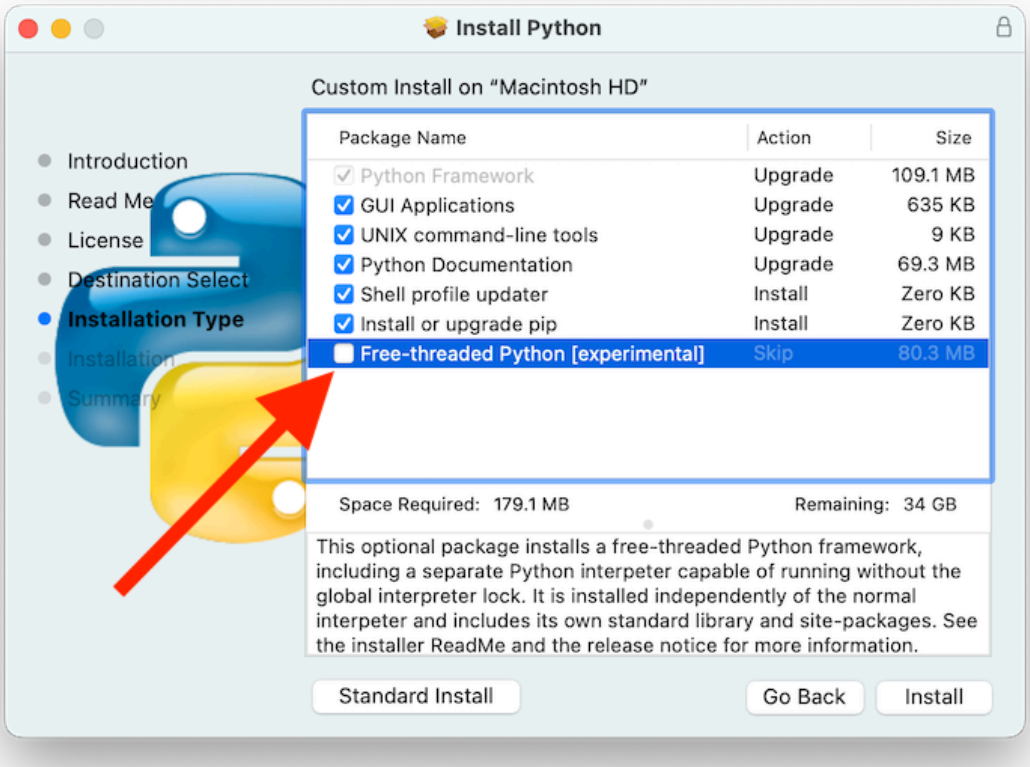
5.5.1 Instalando binários com threads livres

Adicionado na versão 3.13: (Experimental)

> **Nota**
>
> Tudo descrito nesta seção é considerado experimental e espera-se que mude em lançamentos futuros.

O pacote de instalação do *Python para macOS* em `python.org` pode, opcionalmente, instalar uma compilação adicional do Python 3.13 que dá suporte à **PEP 703**, o recurso experimental de threads livres (executado com o *trava global do interpretador* desativado). Verifique a página de lançamento em `python.org` para obter informações atualizadas.

Como esse recurso ainda é considerado experimental, seu suporte não é instalado por padrão. Ele é empacotado como uma opção de instalação separada, disponível clicando no botão **Personalizar** na etapa **Tipo de instalação** do instalador, como descrito acima.



Se a caixa ao lado do nome **Free-Threaded Python** for marcada, um arquivo `PythonT.framework` também será instalado junto ao arquivo `Python.framework` em `/Library/Frameworks`. Essa configuração permite que a compilação de Python |usemac_x_doy_y| com threads livres coexista no seu sistema com a compilação de Python 3.13 tradicional (com GIL) com risco mínimo durante instalação ou testagem. Esse layout de instalação é experimental, e sujeito a mudanças em lançamentos futuros.

Precauções e limitações conhecidas:

- O pacote **Unix command-line tools**, que é selecionado por padrão, instalará links em `/usr/local/bin` para `python3.13t`, o interpretador com threads livres e `python3.13t-config`, um utilitário de configuração que pode ser útil para criadores de pacote. Como `/usr/local/bin` é normalmente incluído em seu `PATH` do console, geralmente não é necessário alterar as variáveis de ambiente do `PATH` para usar o `python3.13t`.
- Neste lançamento, o pacote **Shell profile updater** e o comando `Update Shell Profile.command` em `/Applications/Python 3.13/` não dão suporte ao pacote de threads livres.
- A compilação com threads livres e a compilação tradicional têm caminhos de pesquisa separados e diretórios `site-packages` separados, portanto, por padrão, se você precisar de um pacote disponível em ambas compilações, talvez seja necessário instalá-lo em ambos. O pacote com threads livres instalará uma instância separada do **`pip`** para uso com o `python3.13t`.
  - Para instalar um pacote usando **`pip`** sem um **`venv`**:

    ```
    python3.13t -m pip install <package_name>
    ```

- Ao trabalhar com vários ambientes Python, geralmente é mais seguro e mais fácil criar e usar ambientes virtuais. Isso pode tanto evitar possíveis conflitos de comandos, quanto evitar a confusão sobre qual Python está sendo usado:

  ```
  python3.13t -m venv <venv_name>
  ```

  e, em seguida, **`activate`**.
- Para executar uma versão do IDLE com threads livres:

```
python3.13t -m idlelib
```

- Os intérpretes em ambos as compilações respondem às mesmas variáveis de ambiente *PYTHON*, o que pode gerar resultados inesperados, por exemplo, se você tiver `PYTHONPATH` definido em um perfil de console. Se necessário, há *opções de linha de comando* como `-E` para ignorar essas variáveis de ambiente.
- A compilação com threads livres é ligada às bibliotecas compartilhadas de terceiros, como `OpenSSL` e `Tk`, instaladas no framework tradicional. Isso significa que ambas as compilações também compartilham um conjunto de certificados, como instalados pelo script **`Install Certificates.command`**, portanto, ele só precisa ser executado uma vez.
- Se você não puder depender do link em `/usr/local/bin` apontando para `python3.13t` com threads livres do `python.org` (por exemplo, se quiser instalar sua própria versão lá ou se alguma outra distribuição instalá-lo lá), você poderá definir explicitamente sua variável de ambiente de console `PATH` para incluir o diretório da framework `PythonT bin`:

  ```
  export PATH="/Library/Frameworks/PythonT.framework/Versions/3.13/bin":"$PATH"
  ```

  A instalação tradicional do framework por padrão faz algo semelhante, exceto pelo arquivo `Python.framework`. Observe que ter dois diretórios `bin` do framework em `PATH` pode gerar confusão se houver nomes duplicados, como `python3.13`, em ambos; qual deles é realmente usado depende da ordem em que aparecem em `PATH`. Os comandos `which python3.x` ou `which python3.xt` pode mostrar qual caminho está sendo usado. Usar ambientes virtuais pode ajudar a evitar essas ambiguidades. Outra opção pode ser criar um **`alias`** no console para o interpretador desejado, como:

  ```
  alias py3.13="/Library/Frameworks/Python.framework/Versions/3.13/bin/python3.13
  ↪"
  alias py3.13t="/Library/Frameworks/PythonT.framework/Versions/3.13/bin/python3.
  ↪13t"
  ```

5.5.2 Instalação usando a linha de comando

Se você quiser usar automação para instalar o pacote de instalação `python.org` (em vez de usar o aplicativo GUI **`Installer`** do macOS), o utilitário para linha de comando **`installer`** permite que você selecione opções que não sejam do padrão. Se você não estiver familiarizado com o **`installer`**, ele pode ser um tanto enigmático (consulte: command:*man installer* para obter mais informações). Como exemplo, o seguinte trecho de console mostra uma maneira de fazer isso, usando a versão `3.13.0b2` e selecionando a opção de interpretador com threads livres:

```
RELEASE="python-3.13.0b2-macos11.pkg"

# baixa o pkg do instalador
curl -O https://www.python.org/ftp/python/3.13.0/${RELEASE}

# cria o choicechanges do instalador para personalizar a instalação:
#    habilita o pacote PythonTFramework-3.13
#    enquanto aceita outros padrões (instala todos os outros apcotes)
cat > ./choicechanges.plist <<EOF
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plist PUBLIC "-//Apple//DTD PLIST 1.0//EN" "http://www.apple.com/DTDs/
↪PropertyList-1.0.dtd">
<plist version="1.0">
<array>
        <dict>
                <key>attributeSetting</key>
                <integer>1</integer>
                <key>choiceAttribute</key>
                <string>selected</string>
                <key>choiceIdentifier</key>
                <string>org.python.Python.PythonTFramework-3.13</string>
```

(continua na próxima página)

(continuação da página anterior)

```
        </dict>
</array>
</plist>
EOF

sudo installer -pkg ./${RELEASE} -applyChoiceChangesXML ./choicechanges.plist -
↪target /
```

Você pode então testar que ambas as compilações de instaladores estão disponíveis com algo como:

```
$ # teste que o interpretador com threads livres foi instalado se o pacote Unix␣
↪Command Tools foi habilitado
$ /usr/local/bin/python3.13t -VV
Python 3.13.0b2 experimental free-threading build (v3.13.0b2:3a83b172af, Jun  5␣
↪2024, 12:57:31) [Clang 15.0.0 (clang-1500.3.9.4)]
$ #    e o interpretador tradicional
$ /usr/local/bin/python3.13 -VV
Python 3.13.0b2 (v3.13.0b2:3a83b172af, Jun  5 2024, 12:50:24) [Clang 15.0.0 (clang-
↪1500.3.9.4)]
$ # teste que também eles estão disponíveis sem o prefixo se /usr/local/bin␣
↪estiver no $PATH
$ python3.13t -VV
Python 3.13.0b2 experimental free-threading build (v3.13.0b2:3a83b172af, Jun  5␣
↪2024, 12:57:31) [Clang 15.0.0 (clang-1500.3.9.4)]
$ python3.13 -VV
Python 3.13.0b2 (v3.13.0b2:3a83b172af, Jun  5 2024, 12:50:24) [Clang 15.0.0 (clang-
↪1500.3.9.4)]
```

Nota

Os instaladores atuais do `python.org` só instalam em locais fixos, como `/Library/Frameworks/`, `/Applications` e `/usr/local/bin`. Não é possível usar a opção `-domain` do **`installer`** para instalar em outros locais.

5.5.3 Distribuindo aplicações Python

Existe uma variedade de ferramentas para converter seu código Python em uma aplicação distribuível independente:

- py2app: Oferece suporte à criação de pacotes `.app` do macOS a partir de um projeto Python.
- Briefcase: Parte do BeeWare Project; uma ferramenta de empacotamento multiplataforma que permite criar *bundles* `.app` no macOS, além de gerenciar assinaturas digitais e a notarização.
- PyInstaller: Uma ferramenta de empacotamento multiplataforma que cria um único arquivo ou pasta como um artefato distribuível.

5.5.4 Conformidade com a App Store

As aplicações enviadas para distribuição pela macOS App Store devem passar pelo processo de revisão de aplicativos da Apple. Esse processo inclui um conjunto de regras de validação automatizadas que inspecionam o pacote de aplicativos enviado em busca de código problemático.

A biblioteca padrão do Python contém alguns códigos que violam essas regras automatizadas. Embora essas violações pareçam ser falsos positivos, as regras de revisão da Apple não podem ser contestadas. Portanto, é necessário modificar a biblioteca padrão do Python para que uma aplicação passe na revisão da App Store.

A árvore de fontes do Python contém um arquivo de patch que removerá todo o código que é conhecido por causar problemas no processo de revisão da App Store. Este patch é aplicado automaticamente quando o CPython é

configurado com a opção `--with-app-store-compliance`.

Este patch normalmente não é necessário para usar o CPython em um Mac; nem é necessário se você estiver distribuindo uma aplicação *fora* da App Store do macOS. É necessário *apenas* se você estiver usando a App Store do macOS como canal de distribuição.

5.6 Outros recursos

A página de ajuda do python.org contém links para vários recursos úteis. A lista de discussão Pythonmac-SIG é outro recurso específico de suporte para usuários e desenvolvedores de Python no Mac.

CAPÍTULO 6

Utilizando Python no Android

Python no Android é diferente do Python em plataformas de desktop. Em uma plataforma desktop, o Python geralmente é instalado como um recurso do sistema que pode ser usado por qualquer usuário daquele computador. Os usuários então interagem com o Python executando um executável **`python`** e inserindo comandos em um prompt interativo ou executando um script Python.

No Android, não existe o conceito de instalação como recurso do sistema. A única unidade de distribuição de software é uma aplicação, ou "app". Também não há console onde você possa executar um executável **`python`** ou interagir com um REPL do Python.

Como resultado, a única maneira de usar Python no Andoid é no modo incorporado – ou seja, escrevendo uma aplicação Android nativo, incorporando um interpretador Python usando `libpython` e invocando o código Python usando API de incorporação do Python. O interpretador Python completo, a biblioteca padrão e todo o seu código Python são então empacotados para sua aplicação para seu próprio uso privado.

A biblioteca padrão do Python tem algumas omissões e restrições notáveis no Andoid. Consulte o guia de disponibilidade de API para Android para obter detalhes.

6.1 Adicionando Python a uma aplicação Android

Essas instruções são necessárias somente se você estiver planejando compilar Python para Android você mesmo. A maioria dos usuários *não* precisa fazer isso. Em vez disso, use uma das seguintes ferramentas, que fornecerão uma experiência muito mais fácil:

- Briefcase, do projeto BeeWare
- Buildozer, do projeto Kivy
- Chaquopy
- pyqtdeploy
- Termux

Se você tem certeza de que quer fazer tudo isso manualmente, continue lendo. Você pode usar a aplicação testbed como um guia; cada passo abaixo contém um link para o arquivo relevante.

- Construa o Python seguindo as instruções em Android/README.md.
- Adicione código ao seu arquivo build.gradle para copiar os seguintes itens para o seu projeto. Todos, exceto seu próprio código Python, podem ser copiados de `cross-build/HOST/prefix/lib`:
  - Em suas bibliotecas JNI:

    * `libpython*.*.so`
    * `lib*_python.so` (bibliotecas externas, tal como OpenSSL)
  - Em seus ativos (*assets*):
    * `python*.*` (a biblioteca padrão do Python)
    * `python*.*/site-packages` (seu próprio código Python)
- Adicione código ao seu aplicativo para extrair os ativos para o sistema de arquivos.
- Adicione código a sua aplicação para iniciar o Python em modo incorporado. Isso precisará ser código C chamado via JNI.

CAPÍTULO 7

Usando Python no iOS

Autores
Russell Keith-Magee (2024-03)

Python no iOS é diferente do Python em plataformas de desktop. Em uma plataforma desktop, o Python geralmente é instalado como um recurso do sistema que pode ser usado por qualquer usuário daquele computador. Os usuários então interagem com o Python executando um executável **`python`** e inserindo comandos em um prompt interativo ou executando um script Python.

No iOS, não existe o conceito de instalação como recurso do sistema. A única unidade de distribuição de software é uma aplicação, ou "app". Também não há console onde você possa executar um executável **`python`** ou interagir com um REPL do Python.

Como resultado, a única maneira de usar Python no iOS é no modo incorporado – ou seja, escrevendo uma aplicação iOS nativo e incorporando um interpretador Python usando `libPython` e invocando o código Python usando API de incorporação do Python. O interpretador Python completo, a biblioteca padrão e todo o seu código Python são então empacotados como um pacote independente que pode ser distribuído pela iOS App Store.

Se você deseja experimentar pela primeira vez escrever uma aplicação iOS em Python, projetos como BeeWare e Kivy irão fornecer uma experiência de usuário muito mais acessível. Esses projetos gerenciam as complexidades associadas à execução de um projeto iOS, portanto, você só precisa lidar com o próprio código Python.

7.1 Python em tempo de execução no iOS

7.1.1 Compatibilidade com a versão do iOS

A versão mínima suportada do iOS é especificada em tempo de compilação, usando a opção `--host` do `configure`. Por padrão, quando compilado para iOS, Python será compilado com uma versão iOS mínima suportada de 13.0. Para usar uma versão mínima diferente do iOS, forneça o número da versão como parte do argumento `--host` - por exemplo, `--host=arm64-apple-ios15.4-simulator` compilaria um simulador ARM64 construído com uma meta de implantação de 15.4.

7.1.2 Identificação da plataforma

Ao executar no iOS, `sys.platform` reportará como `ios`. Este valor será retornado em um iPhone ou iPad, independentemente da aplicação estar execitando no simulador ou em um dispositivo físico.

Informações sobre o ambiente de execução específico, incluindo a versão do iOS, modelo do dispositivo e se o dispositivo é um simulador, podem ser obtidas usando `platform.ios_ver()`. `platform.system()` reportará

`iOS` ou `iPadOS`, dependendo do dispositivo.

`os.uname()` reporta detalhes em nível de kernel; ele reportará o nome `Darwin`.

7.1.3 Disponibilidade da biblioteca padrão

A biblioteca padrão do Python tem algumas omissões e restrições notáveis no iOS. Consulte o guia de disponibilidade de API para iOS para obter detalhes.

7.1.4 Módulos de extensão binária

Uma diferença notável sobre o iOS como plataforma é que a distribuição da App Store impõe requisitos rígidos ao empacotamento de uma aplicação. Um desses requisitos rege como os módulos de extensão binária são distribuídos.

A iOS App Store exige que *todos* os módulos binários em uma aplicação iOS sejam bibliotecas dinâmicas, contidas em um framework com metadados apropriados, armazenados na pasta `Frameworks` da aplicação empacotada. Pode haver apenas um único binário por framework, e não pode haver nenhum material binário executável fora da pasta `Frameworks`.

Isto entra em conflito com a abordagem usual do Python para distribuição de binários, que permite que um módulo de extensão binária seja carregado de qualquer local em `sys.path`. Para garantir a conformidade com as políticas da App Store, um projeto iOS deve pós-processar quaisquer pacotes Python, convertendo módulos binários `.so` em estruturas independentes individuais com metadados e assinatura apropriados. Para obter detalhes sobre como realizar esse pós-processamento, consulte o guia para *adicionar Python ao seu projeto*.

Para ajudar o Python a descobrir binários em seu novo local, o arquivo `.so` original em `sys.path` é substituído por um arquivo `.fwork`. Este arquivo é um arquivo texto que contém a localização do binário do framework, relativo ao pacote de aplicações. Para permitir que o framework retorne ao local original, o framework deve conter um arquivo `.origin` que contém a localização do arquivo `.fwork`, relativo ao pacote da aplicação.

Por exemplo, considere o caso de uma importação `from foo.bar import _whiz`, onde `_whiz` é implementado com o módulo binário `sources/foo/bar/_whiz.abi3.so`, com `sources` sendo o local registrado em `sys.path`, relativo ao pacote da aplicação. Este módulo *deve* ser distribuído como `Frameworks/foo.bar._whiz.framework/foo.bar._whiz` (criando o nome do framework a partir do caminho de importação completo do módulo), com um arquivo `Info.plist` no diretório `.framework` identificando o binário como um framework. O módulo `foo.bar._whiz` seria representado no local original com um arquivo marcador `sources/foo/bar/_whiz.abi3.fwork`, contendo o caminho `Frameworks/foo.bar._whiz/foo.bar._whiz`. O framework também conteria `Frameworks/foo.bar._whiz.framework/foo.bar._whiz.origin`, contendo o caminho para o arquivo `.fwork`.

Ao executar no iOS, o interpretador Python instalará um `AppleFrameworkLoader` que é capaz de ler e importar arquivos `.fwork`. Uma vez importado, o atributo `__file__` do módulo binário reportará como a localização do arquivo `.fwork`. Entretanto, o `ModuleSpec` para o módulo carregado reportará a `origin` como a localização do binário na pasta do framework.

7.1.5 Binários stub do compilador

O Xcode não expõe compiladores explícitos para iOS; em vez disso, ele usa um script `xcrun` que resolve um caminho do compilador (por exemplo, `xcrun --sdk iphoneos clang` para obter o `clang` para um dispositivo iPhone). No entanto, usar este script apresenta dois problemas:

- A saída de `xcrun` inclui caminhos que são específicos da máquina, resultando em um módulo sysconfig que não pode ser compartilhado entre usuários; e
- Isso resulta em definições `CC`/`CPP`/`LD`/`AR` que incluem espaços. Existem muitas ferramentas do ecossistema C que pressupõem que você pode dividir uma linha de comando no primeiro espaço para obter o caminho para o executável do compilador; este não é o caso ao usar `xcrun`.

Para evitar esses problemas, o Python forneceu stubs para essas ferramentas. Esses stubs são wrappers de script de shell em torno das ferramentas subjacentes `xcrun`, distribuídos em uma pasta `bin` distribuída junto com a estrutura iOS compilada. Esses scripts são relocáveis e sempre serão resolvidos para os caminhos apropriados do sistema local. Ao incluir esses scripts na pasta bin que acompanha um framework, o conteúdo do módulo `sysconfig` se torna útil

para usuários finais compilarem seus próprios módulos. Ao compilar módulos Python de terceiros para iOS, você deve garantir que esses binários stub estejam no seu caminho.

7.2 Instalando Python no iOS

7.2.1 Ferramentas para construir aplicações de iOS

A construção para iOS requer o uso das ferramentas Xcode da Apple. É altamente recomendável que você use a versão estável mais recente do Xcode. Isso exigirá o uso da versão mais recente (ou segunda) do macOS lançada recentemente, já que a Apple não mantém o Xcode para versões mais antigas do macOS. As ferramentas de linha de comando do Xcode não são suficientes para o desenvolvimento iOS; você precisa de uma instalação *completa* do Xcode.

Se quiser executar seu código no simulador iOS, você também precisará instalar um iOS Simulator Platform. Você deverá ser solicitado a selecionar um iOS Simulator Platform ao executar o Xcode pela primeira vez. Alternativamente, você pode adicionar um iOS Simulator Platform selecionando na guia Platforms do painel Settings do Xcode.

7.2.2 Adicionando Python a um projeto iOS

Python pode ser adicionado a qualquer projeto iOS, usando Swift ou Objective C. Os exemplos a seguir usarão Objective C; se você estiver usando Swift, poderá achar uma biblioteca como PythonKit útil.

Para adicionar Python a um projeto Xcode de iOS:

1. Construa ou obtenha um `XCFramework` em Python. Veja as instruções em iOS/README.rst (na distribuição fonte do CPython) para detalhes sobre como construir um `XCFramework` em Python. No mínimo, você precisará de uma construção que tenha suporte `arm64-apple-ios`, além de `arm64-apple-ios-simulator` ou `x86_64-apple-ios-simulator`.
2. Arraste o `XCframework` para o seu projeto iOS. Nas instruções a seguir, presumiremos que você colocou o `XCframework` na raiz do seu projeto; no entanto, você pode usar qualquer outro local desejado ajustando os caminhos conforme necessário.
3. Arraste o arquivo `iOS/Resources/dylib-Info-template.plist` para o seu projeto e certifique-se de que ele esteja associado ao destino da aplicação.
4. Adicione o código da sua aplicação como uma pasta no seu projeto Xcode. Nas instruções a seguir, presumiremos que seu código de usuário está em uma pasta chamada `app` na raiz do seu projeto; você pode usar qualquer outro local ajustando os caminhos conforme necessário. Certifique-se de que esta pasta esteja associada ao destino da sua aplicação.
5. Selecione o destino da aplicação selecionando o nó raiz do seu projeto Xcode e, em seguida, o nome do destino na barra lateral que aparece.
6. Nas configurações de "General", em "Frameworks, Libraries and Embedded Content", adicione `Python.xcframework`, com "Embed & Sign" selecionado.
7. Na guia "Build Settings", modifique o seguinte:
   - Build Options
     - User Script Sandboxing: No
     - Enable Testability: Yes
   - Search Paths
     - Framework Search Paths: `$(PROJECT_DIR)`
     - Header Search Paths: `"$(BUILT_PRODUCTS_DIR)/Python.framework/Headers"`
   - Apple Clang - Warnings - All languages
     - Quoted Include In Framework Header: No

8. Adicione uma etapa de construção que copie a biblioteca padrão do Python em sua aplicação. Na aba "Build Phases", adicione uma nova etapa de construção "Run Script" *antes* da etapa "Embed Frameworks", mas *depois* da etapa "Copy Bundle Resources". Nomeie a etapa como "Install Target Specific Python Standard Library", desative a caixa de seleção "Based on dependency analysis" e defina o conteúdo do script como:

```
set -e

mkdir -p "$CODESIGNING_FOLDER_PATH/python/lib"
if [ "$EFFECTIVE_PLATFORM_NAME" = "-iphonesimulator" ]; then
    echo "Instalando módulos Python para iOS Simulator"
    rsync -au --delete "$PROJECT_DIR/Python.xcframework/ios-arm64_x86_64-
↪simulator/lib/" "$CODESIGNING_FOLDER_PATH/python/lib/"
else
    echo "Instalando módulos Python para iOS Device"
    rsync -au --delete "$PROJECT_DIR/Python.xcframework/ios-arm64/lib/" "
↪$CODESIGNING_FOLDER_PATH/python/lib/"
fi
```

Note que o nome da "fatia" do simulador no XCframework pode ser diferente, dependendo das arquiteturas de CPU que seu `XCFramework` provê suporte.

9. Adicione uma segunda etapa de construção que processe os módulos de extensão binária na biblioteca padrão no formato "Framework". Adicione uma etapa de construção "Run Script" *diretamente após* aquela que você adicionou na etapa 8, chamada "Prepare Python Binary Modules". Também deve ter "Based on dependency analysis" desmarcado, com o seguinte conteúdo de script:

```
set -e

install_dylib () {
    INSTALL_BASE=$1
    FULL_EXT=$2

    # O nome do arquivo da extensão
    EXT=$(basename "$FULL_EXT")
    # A localização do arquivo da extensão, relativo ao pacote
    RELATIVE_EXT=${FULL_EXT#$CODESIGNING_FOLDER_PATH/}
    # O caminho para o arquivo da extensão, relativo à base de instalação
    PYTHON_EXT=${RELATIVE_EXT/$INSTALL_BASE/}
    # O nome completo e pontilhado do módulo de extensão, construído a partir␣
↪do caminho do arquivo.
    FULL_MODULE_NAME=$(echo $PYTHON_EXT | cut -d "." -f 1 | tr "/" ".");
    # Um identificador de pacote; não é realmente usado, mas é necessário para␣
↪empacotamento de frameworks no Xcode
    FRAMEWORK_BUNDLE_ID=$(echo $PRODUCT_BUNDLE_IDENTIFIER.$FULL_MODULE_NAME |␣
↪tr "_" "-")
    # O nome da pasta do framework.
    FRAMEWORK_FOLDER="Frameworks/$FULL_MODULE_NAME.framework"

    # Se a pasta do framework não existir, cria-a.
    if [ ! -d "$CODESIGNING_FOLDER_PATH/$FRAMEWORK_FOLDER" ]; then
        echo "Criando framework para $RELATIVE_EXT"
        mkdir -p "$CODESIGNING_FOLDER_PATH/$FRAMEWORK_FOLDER"
        cp "$CODESIGNING_FOLDER_PATH/dylib-Info-template.plist" "$CODESIGNING_
↪FOLDER_PATH/$FRAMEWORK_FOLDER/Info.plist"
        plutil -replace CFBundleExecutable -string "$FULL_MODULE_NAME" "
↪$CODESIGNING_FOLDER_PATH/$FRAMEWORK_FOLDER/Info.plist"
        plutil -replace CFBundleIdentifier -string "$FRAMEWORK_BUNDLE_ID" "
↪$CODESIGNING_FOLDER_PATH/$FRAMEWORK_FOLDER/Info.plist"
```

(continua na próxima página)

(continuação da página anterior)

```
    fi

    echo "Instalando binário para $FRAMEWORK_FOLDER/$FULL_MODULE_NAME"
    mv "$FULL_EXT" "$CODESIGNING_FOLDER_PATH/$FRAMEWORK_FOLDER/$FULL_MODULE_
↪NAME"
    # Cria um arquivo substituto .fwork onde o .so estava
    echo "$FRAMEWORK_FOLDER/$FULL_MODULE_NAME" > ${FULL_EXT%.so}.fwork
    # Cria uma referência para o local do arquivo .so no framework
    echo "${RELATIVE_EXT%.so}.fwork" > "$CODESIGNING_FOLDER_PATH/$FRAMEWORK_
↪FOLDER/$FULL_MODULE_NAME.origin"
}

PYTHON_VER=$(ls -1 "$CODESIGNING_FOLDER_PATH/python/lib")
echo "Instalando módulos de extensão da biblioteca padrão para Python $PYTHON_
↪VER..."
find "$CODESIGNING_FOLDER_PATH/python/lib/$PYTHON_VER/lib-dynload" -name "*.so
↪" | while read FULL_EXT; do
    install_dylib python/lib/$PYTHON_VER/lib-dynload/ "$FULL_EXT"
done

# Remove o modelo dylib
rm -f "$CODESIGNING_FOLDER_PATH/dylib-Info-template.plist"

echo "Assinando frameworks como $EXPANDED_CODE_SIGN_IDENTITY_NAME ($EXPANDED_
↪CODE_SIGN_IDENTITY)..."
find "$CODESIGNING_FOLDER_PATH/Frameworks" -name "*.framework" -exec /usr/bin/
↪codesign --force --sign "$EXPANDED_CODE_SIGN_IDENTITY" ${OTHER_CODE_SIGN_
↪FLAGS:-} -o runtime --timestamp=none --preserve-metadata=identifier,
↪entitlements,flags --generate-entitlement-der "{}" \;
```

10. Adicione código Objective C para inicializar e usar um interpretador Python em modo embarcado. Você deve garantir que:

    - `Modo UTF-8` esteja *enabled*;
    - `Buffered stdio` esteja *disabled*;
    - `Escrita de bytecode` esteja *disabled*;
    - `Manipuladores de sinais` estejam *enabled*;
    - `PYTHONHOME` para o interpretador esteja configurado para apontar para a subpasta `python` do pacote da sua aplicação; e
    - O `PYTHONPATH` para o interpretador inclui:
        - a subpasta `python/lib/python3.X` do pacote da sua aplicação,
        - a subpasta `python/lib/python3.X/lib-dynload` do pacote da sua aplicação, e
        - a subpasta `app` do pacote da sua aplicação

    O local do pacote da sua aplicação pode ser determinada usando `[[NSBundle mainBundle] resourcePath]`.

Os passos 8, 9 e 10 destas instruções presumem que você tem uma única pasta de código de aplicação Python puro, chamada `app`. Se você tiver módulos binários de terceiros em sua aplicação, serão necessárias algumas etapas adicionais:

- Você precisa garantir que todas as pastas que contenham binários de terceiros sejam associadas ao destino da aplicação ou copiadas como parte da etapa 8. A etapa 8 também deve limpar quaisquer binários que não sejam apropriados para a plataforma que uma construção específica está direcionando (ou seja, exclua todos os binários do dispositivo se estiver criando uma aplicação direcionado ao simulador).

- Quaisquer pastas que contenham binários de terceiros devem ser processadas no formato framework no passo 9. A invocação de `install_dylib` que processa a pasta `lib-dynload` pode ser copiada e adaptada para este propósito.
- Se você estiver usando uma pasta separada para pacotes de terceiros, certifique-se de que essa pasta esteja incluída como parte da configuração `PYTHONPATH` no passo 10.

7.3 Conformidade com a App Store

O único mecanismo para distribuir aplicações para dispositivos iOS de terceiros é enviar a aplicação para a App Store do iOS; as aplicações enviadas para distribuição devem passar pelo processo de revisão de aplicações da Apple. Esse processo inclui um conjunto de regras de validação automatizadas que inspecionam o pacote de aplicação enviado em busca de código problemático.

A biblioteca padrão do Python contém alguns códigos que violam essas regras automatizadas. Embora essas violações pareçam ser falsos positivos, as regras de revisão da Apple não podem ser contestadas; portanto, é necessário modificar a biblioteca padrão do Python para que uma aplicação passe na revisão da App Store.

A árvore de fontes do Python contém um arquivo de patch que removerá todo o código que é conhecido por causar problemas no processo de revisão da App Store. Este patch é aplicado automaticamente quando o ao construir para o iOS.

CAPÍTULO 8

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 console *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`.

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`.

annotate function

A function that can be called to retrieve the *annotations* of an object. This function is accessible as the `__annotate__` attribute of functions, classes, and modules. Annotate functions are a subset of *evaluate functions*.

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*.

Annotations of local variables cannot be accessed at runtime, but annotations of global variables, class attributes, and functions can be retrieved by calling `annotationlib.get_annotations()` on modules, classes, and functions, respectively.

See *variable annotation*, *function annotation*, **PEP 484**, **PEP 526**, and **PEP 649**, which describe this functionality. Also see annotations-howto for best practices on working with annotations.

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 geralmente referenciado pelo nome separado por um ponto. Por exemplo, se um objeto *o* tem um atributo *a* esse seria referenciado como *o.a*.

É possível dar a um objeto um atributo cujo nome não seja um identificador conforme definido por identifiers, por exemplo usando `setattr()`, se o objeto permitir. Tal atributo não será acessível usando uma expressão pontilhada e, em vez disso, precisaria ser recuperado com `getattr()`.

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 bytes 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`.

referência emprestada
Na API C do Python, uma referência emprestada é uma referência a um objeto que não é dona da referência. Ela se torna um ponteiro solto se o objeto for destruído. Por exemplo, uma coleta de lixo pode remover a última *referência forte* para o objeto e assim destruí-lo.

Chamar `Py_INCREF()` na *referência emprestada* é recomendado para convertê-lo, internamente, em uma *referência forte*, exceto quando o objeto não pode ser destruído antes do último uso da referência emprestada. A função `Py_NewRef()` pode ser usada para criar uma nova *referência forte*.

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.

chamável
Um chamável é um objeto que pode ser chamado, possivelmente com um conjunto de argumentos (veja *argumento*), com a seguinte sintaxe:

```
chamavel(argumento1, argumento2, argumentoN)
```

Uma *função*, e por extensão um *método*, é um chamável. Uma instância de uma classe que implementa o método `__call__()` também é um chamável.

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).

variável de clausura
Uma *variável livre* referenciada de um *escopo aninhado* que é definida em um escopo externo em vez de ser resolvida em tempo de execução a partir dos espaços de nomes embutido ou globais. Pode ser explicitamente definida com a palavra reservada `nonlocal` para permitir acesso de gravação, ou implicitamente definida se a variável estiver sendo somente lida.

Por exemplo, na função `interna` no código a seguir, tanto `x` quanto `print` são *variáveis livres*, mas somente `x` é uma *variável de clausura*:

```
def externa():
    x = 0
    def interna():
        nonlocal x
        x += 1
        print(x)
    return interna
```

Devido ao atributo `codeobject.co_freevars` (que, apesar do nome, inclui apenas os nomes das variáveis de clausura em vez de listar todas as variáveis livres referenciadas), o termo mais geral *variável livre* às vezes é usado mesmo quando o significado pretendido é se referir especificamente às variáveis de clausura.

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.

contexto
Este termo tem diferentes significados dependendo de onde e como ele é usado. Alguns significados comuns:

- O estado ou ambiente temporário estabelecido por um *gerenciador de contexto* por meio de uma instrução `with`.
- A coleção de ligações de chave-valor associadas a um objeto `contextvars.Context` específico e acessadas por meio de objetos `ContextVar`. Veja também *variável de contexto*.
- Um objeto `contextvars.Context`. Veja também *contexto atual*.

protocolo de gerenciamento de contexto
Os métodos `__enter__()` e `__exit__()` chamados pela instrução `with`. Veja **PEP 343**.

gerenciador de contexto
Um objeto que implementa o *protocolo de gerenciamento de contexto* e controla o ambiente visto em uma instrução `with`. Veja **PEP 343**.

variável de contexto
Uma variável cujo valor depende de qual contexto é o *contexto atual*. Os valores são acessados por meio de objetos `contextvars.ContextVar`. Variáveis de contexto são usadas principalmente para isolar o estado entre tarefas assíncronas simultâneas.

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.

contexto atual
O *contexto* (objeto `contextvars.Context`) que é usado atualmente pelos objetos `ContextVar` para acessar (obter ou definir) os valores de *variáveis de contexto*. Cada thread tem seu próprio contexto atual. Frameworks para executar tarefas assíncronas (veja `asyncio`) associam cada tarefa a um contexto que se torna o contexto atual sempre que a tarefa inicia ou retoma a execução.

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(arg):
    ...
f = staticmethod(f)

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

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 ou o Guia de Descritores.

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__()`. Isso é chamado de hash em 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.

evaluate function
A function that can be called to evaluate a lazily evaluated attribute of an object, such as the value of type aliases created with the `type` statement.

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*.

tratador de erros e codificação do sistema de arquivos
Tratador de erros e codificação usado pelo Python para decodificar bytes do sistema operacional e codificar Unicode para o sistema operacional.

A codificação do sistema de arquivos deve garantir a decodificação bem-sucedida de todos os bytes abaixo de 128. Se a codificação do sistema de arquivos falhar em fornecer essa garantia, as funções da API podem

levantar `UnicodeError`.

As funções `sys.getfilesystemencoding()` e `sys.getfilesystemencodeerrors()` podem ser usadas para obter o tratador de erros e codificação do sistema de arquivos.

O *tratador de erros e codificação do sistema de arquivos* são configurados na inicialização do Python pela função `PyConfig_Read()`: veja os membros `filesystem_encoding` e `filesystem_errors` do `PyConfig`.

Veja também *codificação da localidade*.

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

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 finders-and-loaders e `importlib` para muito mais detalhes.

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**.

threads livres
Um modelo de threads onde múltiplas threads podem simultaneamente executar bytecode Python no mesmo interpretador. Isso está em contraste com a *trava global do interpretador* que permite apenas uma thread por vez executar bytecode Python. Veja **PEP 703**.

variável livre
Formalmente, conforme definido no modelo de execução de linguagem, uma variável livre é qualquer variável usada em um espaço de nomes que não seja uma variável local naquele espaço de nomes. Veja *variável de clausura* para um exemplo. Pragmaticamente, devido ao nome do atributo `codeobject.co_freevars`, o termo também é usado algumas vezes como sinônimo de *variável de clausura*.

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 soma_dois_numeros(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 esta funcionalidade. Veja também annotations-howto para as melhores práticas sobre como trabalhar com anotações.

__future__
A instrução future, `from __future__ import <feature>`, direciona o compilador a compilar o módulo atual usando sintaxe ou semântica que será padrão em uma versão futura de Python. O módulo `__future__` documenta os possíveis valores de *feature*. Importando esse módulo e avaliando suas variáveis, você pode ver quando um novo recurso foi inicialmente adicionado à linguagem e quando será (ou se já é) o 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 laço, um intervalo, e uma cláusula `if` opcional. A expressão combinada gera valores para uma função encapsuladora:

```
>>> sum(i*i for i in range(10))         # soma dos quadrados 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**.

tipo genérico

Um *tipo* que pode ser parametrizado; tipicamente uma classe contêiner tal como `list` ou `dict`. Usado para *dicas de tipo* e *anotações*.

Para mais detalhes, veja tipo apelido genérico, **PEP 483**, **PEP 484**, **PEP 585**, e o módulo `typing`.

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 a GIL ao realizar tarefas computacionalmente muito intensas, como compactação ou cálculos de hash. Além disso, a GIL é sempre liberado nas operações de E/S.

A partir de Python 3.13, o GIL pode ser desabilitado usando a configuração de construção `--disable-gil`. Depois de construir Python com essa opção, o código deve ser executado com a opção `-X gil=0` ou a variável de ambiente `PYTHON_GIL=0` deve estar definida. Esse recurso provê um desempenho melhor para aplicações com múltiplas threads e torna mais fácil o uso eficiente de CPUs com múltiplos núcleos. Para mais detalhes, veja **PEP 703**.

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 e aprendizado integrado para Python. idle é um editor básico e um ambiente interpretador que vem junto com a distribuição padrão do Python.

imortal

Objetos imortais são um detalhe da implementação do CPython introduzida na **PEP 683**.

Se um objeto é imortal, sua *contagem de referências* nunca é modificada e, portanto, nunca é desalocado enquanto o interpretador está em execução. Por exemplo, `True` e `None` são imortais no CPython.

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)`). Para saber mais sobre modo interativo, veja tut-interac.

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 de não-sequência, como o `dict`, *objetos arquivos*, além dos objetos de quaisquer classes que você definir com um método `__iter__()` ou `__getitem__()` que implementam a semântica de *sequência* .

Iteráveis podem ser usados em um laço `for` e em vários outros lugares em que uma sequência é necessária (`zip()`, `map()`, ...). Quando um objeto iterável é passado como argumento para a função embutida `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.

Detalhes da implementação do CPython: O CPython não aplica consistentemente o requisito de que um iterador defina `__iter__()`. E também observe que o CPython com threads livres não garante a segurança do thread das operações do iterador.

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, `operator.attrgetter()`, `operator.itemgetter()` e `operator.methodcaller()` são três construtores de função chave. Consulte o guia 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 também:

- finders-and-loaders
- `importlib.abc.Loader`
- **PEP 302**

codificação da localidade
No Unix, é a codificação da localidade do LC_CTYPE, que pode ser definida com `locale.setlocale(locale.LC_CTYPE, new_locale)`.

No Windows, é a página de código ANSI (ex: `"cp1252"`).

No Android e no VxWorks, o Python usa `"utf-8"` como a codificação da localidade.

`locale.getencoding()` pode ser usado para obter a codificação da localidade.

Veja também *tratador de erros e codificação do sistema de arquivos*.

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

mapeamento
Um objeto contêiner que tem suporte a pesquisas de chave arbitrária e implementa os métodos especificados nas `collections.abc.Mapping` ou `collections.abc.MutableMapping` classes base abstratas. 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 python_2.3_mro 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*.

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

Veja também module-specs.

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]                    # acesso indexado
1024
>>> sys.float_info.max_exp               # acesso a campo nomeado
1024
>>> isinstance(sys.float_info, tuple)    # tipo de tupla
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 herdando `typing.NamedTuple` ou com uma função fábrica `collections.namedtuple()`. As duas últimas técnicas também adicionam 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*.

escopo otimizado
Um escopo no qual os nomes das variáveis locais de destino são conhecidos de forma confiável pelo compilador quando o código é compilado, permitindo a otimização do acesso de leitura e gravação a esses nomes. Os espaços de nomes locais para funções, geradores, corrotinas, compreensões e expressões geradoras são otimizados desta forma. Nota: a maioria das otimizações de interpretador são aplicadas a todos os escopos, apenas aquelas que dependem de um conjunto conhecido de nomes de variáveis locais e não locais são restritas a escopos otimizados.

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 *somentepos1* e *somentepos2* a seguir:

```
def func(somentepos1, somentepos2, /, posicional_ou_nomeado): ...
```

- *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 *somente_nom1* and *somente_nom2* a seguir:

```
def func(arg, *, somente_nom1, somente_nom2): ...
```

- *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_hooks` 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 as formas de escritas mais comuns da linguagem Python, ao invés de implementar códigos usando conceitos comuns a outras linguagens. Por exemplo, um formato comum em Python é fazer um laço 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(comida)):
    print(comida[i])
```

Ao contrário do método mais limpo, Pythônico:

```
for parte in comida:
    print(parte)
```

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 metodo(self):
...             pass
...
>>> C.__qualname__
'C'
>>> C.D.__qualname__
'C.D'
>>> C.D.metodo.__qualname__
'C.D.metodo'
```

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 a um objeto. Quando a contagem de referências de um objeto cai para zero, ele é desalocado. Alguns objetos são *imortais* e têm contagens de referências que nunca são modificadas e, portanto, os objetos nunca são desalocados. A contagem de referências geralmente não é visível para o código Python, mas é um elemento-chave da implementação do *CPython*. Os programadores podem chamar a função `sys.getrefcount()` para retornar a contagem de referências para um objeto específico.

pacote regular

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

Veja também *pacote de espaço de nomes*.

REPL

Um acrônimo para "read–eval–print loop", outro nome para o console *interativo* do interpretador.

__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 mapeamento e não uma sequência porque a busca usa uma chave *hasheável* 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()`. Para mais documentação sobre métodos de sequências em geral, veja Operações comuns de sequências.

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.

suavemente descontinuado
Uma API suavemente descontinuada não deve ser usada em código novo, mas é seguro para código já existente usá-la. A API continua documentada e testada, mas não será aprimorada mais.

A descontinuação suave, diferentemente da descontinuação normal, não planeja remover a API e não emitirá avisos.

Veja PEP 387: Descontinuação suave.

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`.

verificador de tipo estático
Uma ferramenta externa que lê o código Python e o analisa, procurando por problemas como tipos incorretos. Consulte também *dicas de tipo* e o módulo `typing`.

referência forte
Na API C do Python, uma referência forte é uma referência a um objeto que pertence ao código que contém a referência. A referência forte é obtida chamando `Py_INCREF()` quando a referência é criada e liberada com `Py_DECREF()` quando a referência é excluída.

A função `Py_NewRef()` pode ser usada para criar uma referência forte para um objeto. Normalmente, a função `Py_DECREF()` deve ser chamada na referência forte antes de sair do escopo da referência forte, para evitar o vazamento de uma referência.

Veja também *referência emprestada*.

codificador de texto
Uma string em Python é uma sequência de pontos de código Unicode (no intervalo `U+0000–U+10FFFF`). Para armazenar ou transferir uma string, ela precisa ser serializada como uma sequência de bytes.

A serialização de uma string em uma sequência de bytes é conhecida como "codificação" e a recriação da string a partir de uma sequência de bytes é conhecida como "decodificação".

Há uma variedade de diferentes serializações de texto codecs, que são coletivamente chamadas de "codificações de texto".

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*.

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 classe 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)`.

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

Apelidos de tipo são úteis para simplificar *dicas de tipo*. Por exemplo:

```
def remove_tons_de_cinza(
        cores: list[tuple[int, int, int]]) -> list[tuple[int, int, int]]:
    pass
```

pode tornar-se mais legível desta forma:

```
Cor = tuple[int, int, int]

def remove_tons_de_cinza(cores: list[Cor]) -> list[Cor]:
    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 *verificadores de tipo estático*. Eles também 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:
    campo: 'anotação'
```

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

```
contagem: 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. Veja também annotations--howto para as melhores práticas sobre como trabalhar com anotações.

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, pelo seu projeto de referência alternativa em Python, do qual Sphinx pegou muitas boas ideias.

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 3.14.0a1

```
1. This LICENSE AGREEMENT is between the Python Software Foundation ("PSF"), and
   the Individual or Organization ("Licensee") accessing and otherwise using Python
   3.14.0a1 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 3.14.0a1 alone or in any derivative
   version, provided, however, that PSF's License Agreement and PSF's notice of
   copyright, i.e., "Copyright © 2001-2024 Python Software Foundation; All Rights
   Reserved" are retained in Python 3.14.0a1 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 3.14.0a1 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
   3.14.0a1.

4. PSF is making Python 3.14.0a1 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 3.14.0a1 WILL NOT INFRINGE ANY THIRD PARTY RIGHTS.

5. PSF SHALL NOT BE LIABLE TO LICENSEE OR ANY OTHER USERS OF PYTHON 3.14.0a1
   FOR ANY INCIDENTAL, SPECIAL, OR CONSEQUENTIAL DAMAGES OR LOSS AS A RESULT OF
   MODIFYING, DISTRIBUTING, OR OTHERWISE USING PYTHON 3.14.0a1, 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 3.14.0a1, 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
   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.
```

(continua na próxima página)

(continuação da página anterior)

```
2. Subject to the terms and conditions of this License Agreement, CNRI hereby
   grants Licensee a nonexclusive, royalty-free, world-wide license to reproduce,
   analyze, test, perform and/or display publicly, prepare derivative works,
   distribute, and otherwise use Python 1.6.1 alone or in any derivative version,
   provided, however, that CNRI's License Agreement and CNRI's notice of copyright,
   i.e., "Copyright © 1995-2001 Corporation for National Research Initiatives; All
   Rights Reserved" are retained in Python 1.6.1 alone or in any derivative version
   prepared by Licensee.  Alternately, in lieu of CNRI's License Agreement,
   Licensee may substitute the following text (omitting the quotes): "Python 1.6.1
   is made available subject to the terms and conditions in CNRI's License
   Agreement.  This Agreement together with Python 1.6.1 may be located on the
   internet using the following unique, persistent identifier (known as a handle):
   1895.22/1013.  This Agreement may also be obtained from a proxy server on the
   internet using the following URL: http://hdl.handle.net/1895.22/1013."

3. In the event Licensee prepares a derivative work that is based on or
   incorporates Python 1.6.1 or any part thereof, and wants to make the derivative
   work available to others as provided herein, then Licensee hereby agrees to
   include in any such work a brief summary of the changes made to Python 1.6.1.

4. CNRI is making Python 1.6.1 available to Licensee on an "AS IS" basis.  CNRI
   MAKES NO REPRESENTATIONS OR WARRANTIES, EXPRESS OR IMPLIED.  BY WAY OF EXAMPLE,
   BUT NOT LIMITATION, CNRI MAKES NO AND DISCLAIMS ANY REPRESENTATION OR WARRANTY
   OF MERCHANTABILITY OR FITNESS FOR ANY PARTICULAR PURPOSE OR THAT THE USE OF
   PYTHON 1.6.1 WILL NOT INFRINGE ANY THIRD PARTY RIGHTS.

5. CNRI SHALL NOT BE LIABLE TO LICENSEE OR ANY OTHER USERS OF PYTHON 1.6.1 FOR
   ANY INCIDENTAL, SPECIAL, OR CONSEQUENTIAL DAMAGES OR LOSS AS A RESULT OF
   MODIFYING, DISTRIBUTING, OR OTHERWISE USING PYTHON 1.6.1, OR ANY DERIVATIVE
   THEREOF, EVEN IF ADVISED OF THE POSSIBILITY THEREOF.

6. This License Agreement will automatically terminate upon a material breach of
   its terms and conditions.

7. This License Agreement shall be governed by the federal intellectual property
   law of the United States, including without limitation the federal copyright
   law, and, to the extent such U.S. federal law does not apply, by the law of the
   Commonwealth of Virginia, excluding Virginia's conflict of law provisions.
   Notwithstanding the foregoing, with regard to derivative works based on Python
   1.6.1 that incorporate non-separable material that was previously distributed
   under the GNU General Public License (GPL), the law of the Commonwealth of
   Virginia shall govern this License Agreement only as to issues arising under or
   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 3.14.0a1

```
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

A extensão C `_random` subjacente ao 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
```

(continua na próxima página)

(continuação da página anterior)

```
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, https://www.wide.ad.jp/.

```
Copyright (C) 1995, 1996, 1997, and 1998 WIDE Project.
All rights reserved.

Redistribution and use in source and binary forms, with or without
modification, are permitted provided that the following conditions
are met:
1. Redistributions of source code must retain the above copyright
   notice, this list of conditions and the following disclaimer.
2. Redistributions in binary form must reproduce the above copyright
   notice, this list of conditions and the following disclaimer in the
   documentation and/or other materials provided with the distribution.
3. Neither the name of the project nor the names of its contributors
   may be used to endorse or promote products derived from this software
   without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE PROJECT AND CONTRIBUTORS ``AS IS'' AND
ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
ARE DISCLAIMED.  IN NO EVENT SHALL THE PROJECT OR CONTRIBUTORS BE LIABLE
FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL
```

(continua na próxima página)

(continuação da página anterior)

```
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 `test.support.asynchat` e `test.support.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
```

(continua na próxima página)

(continuação da página anterior)

```
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 codec `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.
```

(continua na próxima página)

(continuação da página anterior)

```
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.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
```

(continua na próxima página)

(continuação da página anterior)

```
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/
```

(continua na próxima página)

(continuação da página anterior)

```
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 https://web.archive.org/web/20220517033456/http://www.netlib.org/fp/dtoa.c. 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` e `ssl` 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: Para o lançamento do OpenSSL 3.0, e lançamentos posteriores derivados deste, se aplica a Apache License v2:

```
                              Apache License
                        Version 2.0, January 2004
                     https://www.apache.org/licenses/

TERMS AND CONDITIONS FOR USE, REPRODUCTION, AND DISTRIBUTION

1. Definitions.

   "License" shall mean the terms and conditions for use, reproduction,
   and distribution as defined by Sections 1 through 9 of this document.

   "Licensor" shall mean the copyright owner or entity authorized by
   the copyright owner that is granting the License.

   "Legal Entity" shall mean the union of the acting entity and all
   other entities that control, are controlled by, or are under common
   control with that entity. For the purposes of this definition,
   "control" means (i) the power, direct or indirect, to cause the
```

(continua na próxima página)

(continuação da página anterior)

```
   direction or management of such entity, whether by contract or
   otherwise, or (ii) ownership of fifty percent (50%) or more of the
   outstanding shares, or (iii) beneficial ownership of such entity.

   "You" (or "Your") shall mean an individual or Legal Entity
   exercising permissions granted by this License.

   "Source" form shall mean the preferred form for making modifications,
   including but not limited to software source code, documentation
   source, and configuration files.

   "Object" form shall mean any form resulting from mechanical
   transformation or translation of a Source form, including but
   not limited to compiled object code, generated documentation,
   and conversions to other media types.

   "Work" shall mean the work of authorship, whether in Source or
   Object form, made available under the License, as indicated by a
   copyright notice that is included in or attached to the work
   (an example is provided in the Appendix below).

   "Derivative Works" shall mean any work, whether in Source or Object
   form, that is based on (or derived from) the Work and for which the
   editorial revisions, annotations, elaborations, or other modifications
   represent, as a whole, an original work of authorship. For the purposes
   of this License, Derivative Works shall not include works that remain
   separable from, or merely link (or bind by name) to the interfaces of,
   the Work and Derivative Works thereof.

   "Contribution" shall mean any work of authorship, including
   the original version of the Work and any modifications or additions
   to that Work or Derivative Works thereof, that is intentionally
   submitted to Licensor for inclusion in the Work by the copyright owner
   or by an individual or Legal Entity authorized to submit on behalf of
   the copyright owner. For the purposes of this definition, "submitted"
   means any form of electronic, verbal, or written communication sent
   to the Licensor or its representatives, including but not limited to
   communication on electronic mailing lists, source code control systems,
   and issue tracking systems that are managed by, or on behalf of, the
   Licensor for the purpose of discussing and improving the Work, but
   excluding communication that is conspicuously marked or otherwise
   designated in writing by the copyright owner as "Not a Contribution."

   "Contributor" shall mean Licensor and any individual or Legal Entity
   on behalf of whom a Contribution has been received by Licensor and
   subsequently incorporated within the Work.

2. Grant of Copyright License. Subject to the terms and conditions of
   this License, each Contributor hereby grants to You a perpetual,
   worldwide, non-exclusive, no-charge, royalty-free, irrevocable
   copyright license to reproduce, prepare Derivative Works of,
   publicly display, publicly perform, sublicense, and distribute the
   Work and such Derivative Works in Source or Object form.

3. Grant of Patent License. Subject to the terms and conditions of
   this License, each Contributor hereby grants to You a perpetual,
```

(continua na próxima página)

(continuação da página anterior)

```
   worldwide, non-exclusive, no-charge, royalty-free, irrevocable
   (except as stated in this section) patent license to make, have made,
   use, offer to sell, sell, import, and otherwise transfer the Work,
   where such license applies only to those patent claims licensable
   by such Contributor that are necessarily infringed by their
   Contribution(s) alone or by combination of their Contribution(s)
   with the Work to which such Contribution(s) was submitted. If You
   institute patent litigation against any entity (including a
   cross-claim or counterclaim in a lawsuit) alleging that the Work
   or a Contribution incorporated within the Work constitutes direct
   or contributory patent infringement, then any patent licenses
   granted to You under this License for that Work shall terminate
   as of the date such litigation is filed.

4. Redistribution. You may reproduce and distribute copies of the
   Work or Derivative Works thereof in any medium, with or without
   modifications, and in Source or Object form, provided that You
   meet the following conditions:

   (a) You must give any other recipients of the Work or
       Derivative Works a copy of this License; and

   (b) You must cause any modified files to carry prominent notices
       stating that You changed the files; and

   (c) You must retain, in the Source form of any Derivative Works
       that You distribute, all copyright, patent, trademark, and
       attribution notices from the Source form of the Work,
       excluding those notices that do not pertain to any part of
       the Derivative Works; and

   (d) If the Work includes a "NOTICE" text file as part of its
       distribution, then any Derivative Works that You distribute must
       include a readable copy of the attribution notices contained
       within such NOTICE file, excluding those notices that do not
       pertain to any part of the Derivative Works, in at least one
       of the following places: within a NOTICE text file distributed
       as part of the Derivative Works; within the Source form or
       documentation, if provided along with the Derivative Works; or,
       within a display generated by the Derivative Works, if and
       wherever such third-party notices normally appear. The contents
       of the NOTICE file are for informational purposes only and
       do not modify the License. You may add Your own attribution
       notices within Derivative Works that You distribute, alongside
       or as an addendum to the NOTICE text from the Work, provided
       that such additional attribution notices cannot be construed
       as modifying the License.

   You may add Your own copyright statement to Your modifications and
   may provide additional or different license terms and conditions
   for use, reproduction, or distribution of Your modifications, or
   for any such Derivative Works as a whole, provided Your use,
   reproduction, and distribution of the Work otherwise complies with
   the conditions stated in this License.

5. Submission of Contributions. Unless You explicitly state otherwise,
```

(continua na próxima página)

(continuação da página anterior)

```
   any Contribution intentionally submitted for inclusion in the Work
   by You to the Licensor shall be under the terms and conditions of
   this License, without any additional terms or conditions.
   Notwithstanding the above, nothing herein shall supersede or modify
   the terms of any separate license agreement you may have executed
   with Licensor regarding such Contributions.

6. Trademarks. This License does not grant permission to use the trade
   names, trademarks, service marks, or product names of the Licensor,
   except as required for reasonable and customary use in describing the
   origin of the Work and reproducing the content of the NOTICE file.

7. Disclaimer of Warranty. Unless required by applicable law or
   agreed to in writing, Licensor provides the Work (and each
   Contributor provides its Contributions) on an "AS IS" BASIS,
   WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or
   implied, including, without limitation, any warranties or conditions
   of TITLE, NON-INFRINGEMENT, MERCHANTABILITY, or FITNESS FOR A
   PARTICULAR PURPOSE. You are solely responsible for determining the
   appropriateness of using or redistributing the Work and assume any
   risks associated with Your exercise of permissions under this License.

8. Limitation of Liability. In no event and under no legal theory,
   whether in tort (including negligence), contract, or otherwise,
   unless required by applicable law (such as deliberate and grossly
   negligent acts) or agreed to in writing, shall any Contributor be
   liable to You for damages, including any direct, indirect, special,
   incidental, or consequential damages of any character arising as a
   result of this License or out of the use or inability to use the
   Work (including but not limited to damages for loss of goodwill,
   work stoppage, computer failure or malfunction, or any and all
   other commercial damages or losses), even if such Contributor
   has been advised of the possibility of such damages.

9. Accepting Warranty or Additional Liability. While redistributing
   the Work or Derivative Works thereof, You may choose to offer,
   and charge a fee for, acceptance of support, warranty, indemnity,
   or other liability obligations and/or rights consistent with this
   License. However, in accepting such obligations, You may act only
   on Your own behalf and on Your sole responsibility, not on behalf
   of any other Contributor, and only if You agree to indemnify,
   defend, and hold each Contributor harmless for any liability
   incurred by, or claims asserted against, such Contributor by reason
   of your accepting any such warranty or additional liability.

END OF TERMS AND CONDITIONS
```

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
```

(continua na próxima página)

(continuação da página anterior)

```
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 C `_ctypes` subjacente ao módulo `ctypes` é construída usando uma cópia incluída das fontes do libffi, a menos que a construção esteja configurada com `--with-system-libffi`:

```
Copyright (c) 1996-2008  Red Hat, Inc and others.

Permission is hereby granted, free of charge, to any person obtaining
a copy of this software and associated documentation files (the
``Software''), to deal in the Software without restriction, including
without limitation the rights to use, copy, modify, merge, publish,
distribute, sublicense, and/or sell copies of the Software, and to
permit persons to whom the Software is furnished to do so, subject to
the following conditions:

The above copyright notice and this permission notice shall be included
in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED ``AS IS'', WITHOUT WARRANTY OF ANY KIND,
EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF
MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND
NONINFRINGEMENT.  IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT
HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY,
WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER
DEALINGS IN THE SOFTWARE.
```

C.3.15 zlib

A extensão `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.
```

(continua na próxima página)

(continuação da página anterior)

```
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
    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

A extensão C `_decimal` subjacente ao módulo `decimal` é construída usando uma cópia incluída da biblioteca libmpdec, a menos que a construção esteja configurada com `--with-system-libmpdec`:

```
Copyright (c) 2008-2020 Stefan Krah. All rights reserved.

Redistribution and use in source and binary forms, with or without
modification, are permitted provided that the following conditions
are met:

1. Redistributions of source code must retain the above copyright
   notice, this list of conditions and the following disclaimer.

2. Redistributions in binary form must reproduce the above copyright
   notice, this list of conditions and the following disclaimer in the
   documentation and/or other materials provided with the distribution.

THIS SOFTWARE IS PROVIDED BY THE AUTHOR AND CONTRIBUTORS "AS IS" AND
ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
ARE DISCLAIMED.  IN NO EVENT SHALL THE AUTHOR OR CONTRIBUTORS BE LIABLE
FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL
DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS
OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION)
HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT
LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY
OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF
SUCH DAMAGE.
```

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.

Redistribution and use in source and binary forms, with or without
modification, are permitted provided that the following conditions
are met:

* Redistributions of works must retain the original copyright notice,
  this list of conditions and the following disclaimer.
* Redistributions in binary form must reproduce the original copyright
  notice, this list of conditions and the following disclaimer in the
  documentation and/or other materials provided with the distribution.
* Neither the name of the W3C nor the names of its contributors may be
  used to endorse or promote products derived from this work without
  specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
"AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR
A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT
OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT
LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE,
```

(continua na próxima página)

(continuação da página anterior)

```
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.3.19 mimalloc

Licença MIT:

```
Copyright (c) 2018-2021 Microsoft Corporation, Daan Leijen

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.3.20 asyncio

Partes do módulo `asyncio` são incorporadas do uvloop 0.16, que é distribuído sob a licença MIT:

```
Copyright (c) 2015-2021 MagicStack Inc.  http://magic.io

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.3.21 Global Unbounded Sequences (GUS)

O arquivo `Python/qsbr.c` é adaptado do esquema de recuperação de memória segura "Global Unbounded Sequences" do FreeBSD em subr_smr.c. O arquivo é distribuído sob a licença BSD de 2 cláusulas:

```
Copyright (c) 2019,2020 Jeffrey Roberson <jeff@FreeBSD.org>

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 unmodified, 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 ``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 BE LIABLE FOR ANY DIRECT, INDIRECT,
INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT
NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE,
DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY
THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
(INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF
THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
```

APÊNDICE D

Direitos autorais

Python e essa documentação é:

Copyright © 2001-2024 Python Software Foundation. Todos os direitos reservados.

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

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

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

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

Índice

Não alfabético

...., **87**
-?
    opção de linha de comando, 5
%APPDATA%, 49
>>>, **87**
__future__, **93**
__slots__, **101**

A

aguardável, **89**
ambiente virtual, **104**
annotate function, **87**
anotação, **87**
anotação de função, **93**
anotação de variável, **103**
apelido de tipo, **103**
API provisória, **100**
argumento, **87**
argumento nomeado, **96**
argumento posicional, **100**
arquivo binário, **89**
arquivo texto, **102**
aspas triplas, **103**
atributo, **88**

B

-B
    opção de linha de comando, 6
-b
    opção de linha de comando, 6
BDFL, **89**
BOLT_APPLY_FLAGS
    opção de linha de comando, 31
BOLT_INSTRUMENT_FLAGS
    opção de linha de comando, 31
--build
    opção de linha de comando, 36
bytecode, **89**
BZIP2_CFLAGS
    opção de linha de comando, 27
BZIP2_LIBS
    opção de linha de comando, 27

C

-c
    opção de linha de comando, 4
caminho de importação, **95**
carregador, **97**
CC
    opção de linha de comando, 27
CFLAGS, 30, 40, 41
    opção de linha de comando, 27
CFLAGS_NODIST, 40, 41
chamável, **89**
--check-hash-based-pycs
    opção de linha de comando, 6
classe, **89**
classe base abstrata, **87**
classe estilo novo, **99**
codificação da localidade, **97**
codificador de texto, **102**
coleta de lixo, **93**
compreensão de conjunto, **102**
compreensão de dicionário, **91**
compreensão de lista, **97**
CONFIG_SITE
    opção de linha de comando, 36
contagem de referências, **101**
contexto, **90**
contexto atual, **91**
contíguo, **90**
contíguo C, 90
contíguo Fortran, 90
corrotina, **91**
CPP
    opção de linha de comando, 27
CPPFLAGS, 39, 42
    opção de linha de comando, 27
CPython, **91**
CURSES_CFLAGS
    opção de linha de comando, 27
CURSES_LIBS
    opção de linha de comando, 27

D

-d

opção de linha de comando, 6
decorador, 91
descritor, 91
desligamento do interpretador, 95
despacho único, 102
dica de tipo, 103
dicionário, 91
--disable-gil
opção de linha de comando, 27
--disable-ipv6
opção de linha de comando, 24
--disable-safety
opção de linha de comando, 35
--disable-test-modules
opção de linha de comando, 29
divisão pelo piso, 93
docstring, 92

E

-E
opção de linha de comando, 6
EAFP, 92
--enable-big-digits
opção de linha de comando, 24
--enable-bolt
opção de linha de comando, 30
--enable-framework
opção de linha de comando, 35, 36
--enable-loadable-sqlite-extensions
opção de linha de comando, 24
--enable-optimizations
opção de linha de comando, 30
--enable-profiling
opção de linha de comando, 31
--enable-pystats
opção de linha de comando, 25
--enable-shared
opção de linha de comando, 33
--enable-slower-safety
opção de linha de comando, 35
--enable-universalsdk
opção de linha de comando, 35
--enable-wasm-dynamic-linking
opção de linha de comando, 29
--enable-wasm-pthreads
opção de linha de comando, 29
entrada de caminho, 99
escopo aninhado, 98
escopo otimizado, 99
espaço de nomes, 98
especial
método, 102
especificação do módulo, 98
evaluate function, 92
--exec-prefix
opção de linha de comando, 29
expressão, 92
expressão geradora, 94

F

f-string, 92
fatia, 102
função, 93
função chave, 96
função de corrotina, 91
função de retorno, 89
função genérica, 94

G

gancho de entrada de caminho, 100
GDBM_CFLAGS
opção de linha de comando, 28
GDBM_LIBS
opção de linha de comando, 28
gerador, 94
gerador assíncrono, 88
gerenciador de contexto, 90
gerenciador de contexto assíncrono, 88
GIL, 94

H

-h
opção de linha de comando, 5
hasheável, 95
--help
opção de linha de comando, 5
--help-all
opção de linha de comando, 5
--help-env
opção de linha de comando, 5
--help-xoptions
opção de linha de comando, 5
--host
opção de linha de comando, 36
HOSTRUNNER
opção de linha de comando, 37

I

-I
opção de linha de comando, 7
-i
opção de linha de comando, 6
IDLE, 95
imortal, 95
importação, 95
importador, 95
imutável, 95
instrução, 102
interativo, 95
interpretado, 95
iterador, 96
iterador assíncrono, 88
iterador gerador, 94
iterador gerador assíncrono, 88
iterável, 96
iterável assíncrono, 88

J

-J
    opção de linha de comando, 11

L

lambda, **96**
LBYL, **96**
LDFLAGS, 39, 41, 42
    opção de linha de comando, 27
LDFLAGS_NODIST, 41, 42
LIBB2_CFLAGS
    opção de linha de comando, 28
LIBB2_LIBS
    opção de linha de comando, 28
LIBEDIT_CFLAGS
    opção de linha de comando, 28
LIBEDIT_LIBS
    opção de linha de comando, 28
LIBFFI_CFLAGS
    opção de linha de comando, 28
LIBFFI_LIBS
    opção de linha de comando, 28
LIBLZMA_CFLAGS
    opção de linha de comando, 28
LIBLZMA_LIBS
    opção de linha de comando, 28
LIBMPDEC_CFLAGS
    opção de linha de comando, 28
LIBMPDEC_LIBS
    opção de linha de comando, 28
LIBREADLINE_CFLAGS
    opção de linha de comando, 28
LIBREADLINE_LIBS
    opção de linha de comando, 28
LIBS
    opção de linha de comando, 27
LIBSQLITE3_CFLAGS
    opção de linha de comando, 28
LIBSQLITE3_LIBS
    opção de linha de comando, 28
LIBUUID_CFLAGS
    opção de linha de comando, 28
LIBUUID_LIBS
    opção de linha de comando, 28
lista, **97**
localizador, **93**
localizador baseado no caminho, **100**
localizador de entrada de caminho, **100**
localizador de metacaminho, **97**

M

-m
    opção de linha de comando, 4
MACHDEP
    opção de linha de comando, 27
mágico
    método, 97
mapeamento, **97**
máquina virtual, **104**
metaclasse, **97**
método, **97**
    especial, 102
    mágico, 97
método especial, **102**
método mágico, **97**
módulo, **98**
módulo de extensão, **92**
MRO, **98**
mutável, **98**

N

nome qualificado, **101**
novas linhas universais, **103**
número complexo, **90**

O

-O
    opção de linha de comando, 7
objeto, **99**
objeto arquivo, **92**
objeto arquivo ou similar, **92**
objeto byte ou similar, **89**
objeto caminho ou similar, **100**
-OO
    opção de linha de comando, 7
opção de linha de comando
    -?, 5
    -B, 6
    -b, 6
    BOLT_APPLY_FLAGS, 31
    BOLT_INSTRUMENT_FLAGS, 31
    --build, 36
    BZIP2_CFLAGS, 27
    BZIP2_LIBS, 27
    -c, 4
    CC, 27
    CFLAGS, 27
    --check-hash-based-pycs, 6
    CONFIG_SITE, 36
    CPP, 27
    CPPFLAGS, 27
    CURSES_CFLAGS, 27
    CURSES_LIBS, 27
    -d, 6
    --disable-gil, 27
    --disable-ipv6, 24
    --disable-safety, 35
    --disable-test-modules, 29
    -E, 6
    --enable-big-digits, 24
    --enable-bolt, 30
    --enable-framework, 35, 36
    --enable-loadable-sqlite-extensions, 24
    --enable-optimizations, 30
    --enable-profiling, 31

--enable-pystats, 25
--enable-shared, 33
--enable-slower-safety, 35
--enable-universalsdk, 35
--enable-wasm-dynamic-linking, 29
--enable-wasm-pthreads, 29
--exec-prefix, 29
GDBM_CFLAGS, 28
GDBM_LIBS, 28
-h, 5
--help, 5
--help-all, 5
--help-env, 5
--help-xoptions, 5
--host, 36
HOSTRUNNER, 37
-I, 7
-i, 6
-J, 11
LDFLAGS, 27
LIBB2_CFLAGS, 28
LIBB2_LIBS, 28
LIBEDIT_CFLAGS, 28
LIBEDIT_LIBS, 28
LIBFFI_CFLAGS, 28
LIBFFI_LIBS, 28
LIBLZMA_CFLAGS, 28
LIBLZMA_LIBS, 28
LIBMPDEC_CFLAGS, 28
LIBMPDEC_LIBS, 28
LIBREADLINE_CFLAGS, 28
LIBREADLINE_LIBS, 28
LIBS, 27
LIBSQLITE3_CFLAGS, 28
LIBSQLITE3_LIBS, 28
LIBUUID_CFLAGS, 28
LIBUUID_LIBS, 28
-m, 4
MACHDEP, 27
-O, 7
-OO, 7
-P, 7
PANEL_CFLAGS, 28
PANEL_LIBS, 29
PKG_CONFIG, 27
PKG_CONFIG_LIBDIR, 27
PKG_CONFIG_PATH, 27
--prefix, 29
-q, 7
-R, 7
-S, 8
-s, 7
TCLTK_CFLAGS, 29
TCLTK_LIBS, 29
-u, 8
-V, 6
-v, 8
--version, 6
-W, 8
--with-address-sanitizer, 33
--with-app-store-compliance, 36
--with-assertions, 32
--with-build-python, 36
--with-builtin-hashlib-hashes, 34
--with-computed-gotos, 31
--with-dbmliborder, 25
--with-dtrace, 33
--with-emscripten-target, 29
--with-ensurepip, 29
--with-framework-name, 36
--with-hash-algorithm, 34
--with-libc, 34
--with-libm, 34
--with-libs, 33
--with-lto, 30
--with-memory-sanitizer, 33
--with-openssl, 34
--with-openssl-rpath, 34
--without-c-locale-coercion, 25
--without-decimal-contextvar, 25
--without-doc-strings, 31
--without-mimalloc, 31
--without-pymalloc, 31
--without-readline, 34
--without-static-libpython, 33
--with-pkg-config, 25
--with-platlibdir, 25
--with-pydebug, 32
--with-readline, 34
--with-ssl-default-suites, 35
--with-strict-overflow, 31
--with-suffix, 24
--with-system-expat, 33
--with-system-libmpdec, 33
--with-thread-sanitizer, 33
--with-trace-refs, 32
--with-tzpath, 24
--with-undefined-behavior-sanitizer, 33
--with-universal-archs, 35
--with-valgrind, 33
--with-wheel-pkg-dir, 25
-X, 9
-x, 9
ZLIB_CFLAGS, 29
ZLIB_LIBS, 29
OPT, 33
ordem de resolução de métodos, **97**

P

-P
opção de linha de comando, 7
pacote, **99**
pacote de espaço de nomes, **98**
pacote provisório, **100**
pacote regular, **101**

PANEL_CFLAGS
    opção de linha de comando, 28
PANEL_LIBS
    opção de linha de comando, 29
parâmetro, **99**
PATH, 11, 20, 44, 46, 53, 54, 56
PATHEXT, 46
PEP, **100**
PKG_CONFIG
    opção de linha de comando, 27
PKG_CONFIG_LIBDIR
    opção de linha de comando, 27
PKG_CONFIG_PATH
    opção de linha de comando, 27
porção, **100**
--prefix
    opção de linha de comando, 29
PROFILE_TASK, 30
Propostas de Melhorias do Python
    PEP 1, 100
    PEP 7, 23
    PEP 8, 85
    PEP 11, 23, 43, 61
    PEP 238, 93
    PEP 278, 103
    PEP 302, 97
    PEP 338, 4
    PEP 343, 90
    PEP 362, 88, 99
    PEP 370, 8, 13
    PEP 397, 54
    PEP 411, 100
    PEP 420, 98, 100
    PEP 443, 94
    PEP 483, 94
    PEP 484, 87, 93, 94, 103, 104
    PEP 488, 7
    PEP 492, 88, 89, 91
    PEP 498, 92
    PEP 514, 55
    PEP 519, 100
    PEP 525, 88
    PEP 526, 87, 104
    PEP 528, 54
    PEP 529, 15, 54
    PEP 538, 16, 25
    PEP 585, 94
    PEP 649, 87
    PEP 683, 95
    PEP 703, 48, 72, 93, 94
    PEP 3116, 103
    PEP 3155, 101
protocolo de gerenciamento de contexto, **90**
PY_PYTHON, 57
pyc baseado em hash, **94**
PYLAUNCHER_ALLOW_INSTALL, 58
PYLAUNCHER_ALWAYS_INSTALL, 58
PYLAUNCHER_DEBUG, 58
PYLAUNCHER_DRYRUN, 58
PYLAUNCHER_NO_SEARCH_PATH, 56
Python 3000, **100**
PYTHON_COLORS, 11
PYTHON_CPU_COUNT, 10
PYTHON_FROZEN_MODULES, 10
PYTHON_GIL, 10, 94
PYTHON_PERF_JIT_SUPPORT, 10
PYTHON_PRESITE, 10
PYTHONCOERCECLOCALE, 25
PYTHONDEBUG, 6, 32
PYTHONDEVMODE, 9
PYTHONDONTWRITEBYTECODE, 6
PYTHONDUMPREFS, 32
PYTHONFAULTHANDLER, 9
PYTHONHASHSEED, 7, 13
PYTHONHOME, 6, 11, 59, 60
Pythônico, **100**
PYTHONINSPECT, 7
PYTHONINTMAXSTRDIGITS, 9
PYTHONIOENCODING, 15
PYTHONLEGACYWINDOWSSTDIO, 13
PYTHONMALLOC, 15, 31
PYTHONNODEBUGRANGES, 10
PYTHONNOUSERSITE, 8
PYTHONOPTIMIZE, 7
PYTHONPATH, 6, 11, 53, 59, 60
PYTHONPERFSUPPORT, 10
PYTHONPROFILEIMPORTTIME, 9
PYTHONPYCACHEPREFIX, 10
PYTHONSAFEPATH, 7
PYTHONSTARTUP, 7, 12
PYTHONTRACEMALLOC, 9
PYTHONUNBUFFERED, 8
PYTHONUTF8, 9, 16, 54
PYTHONVERBOSE, 8
PYTHONWARNDEFAULTENCODING, 10
PYTHONWARNINGS, 9

Q

-q
    opção de linha de comando, 7

R

-R
    opção de linha de comando, 7
referência emprestada, **89**
referência forte, **102**
REPL, **101**

S

-S
    opção de linha de comando, 8
-s
    opção de linha de comando, 7
sequência, **101**
suavemente descontinuado, **102**

T

TCLTK_CFLAGS
    opção de linha de comando, 29
TCLTK_LIBS
    opção de linha de comando, 29
TEMP, 49
threads livres, **93**
tipagem pato, **92**
tipo, **103**
tipo genérico, **94**
tratador de erros e codificação do sistema de arquivos, **92**
trava global do interpretador, **94**
tupla nomeada, **98**

U

-u
    opção de linha de comando, 8

V

-V
    opção de linha de comando, 6
-v
    opção de linha de comando, 8
variável de ambiente
    %APPDATA%, 49
    BASECFLAGS, 40
    BASECPPFLAGS, 39
    BLDSHARED, 42
    CC, 40
    CCSHARED, 40
    CFLAGS, 30, 40, 41
    CFLAGS_ALIASING, 40
    CFLAGS_NODIST, 40, 41
    CFLAGSFORSHARED, 41
    COMPILEALL_OPTS, 40
    CONFIGURE_CFLAGS, 40
    CONFIGURE_CFLAGS_NODIST, 40
    CONFIGURE_CPPFLAGS, 39
    CONFIGURE_LDFLAGS, 41
    CONFIGURE_LDFLAGS_NODIST, 41
    CPPFLAGS, 39, 42
    CXX, 40
    EXTRA_CFLAGS, 40
    LDFLAGS, 39, 41, 42
    LDFLAGS_NODIST, 41, 42
    LDSHARED, 42
    LIBS, 42
    LINKCC, 41
    OPT, 33, 40
    PATH, 11, 20, 44, 46, 53, 54, 56
    PATHEXT, 46
    PROFILE_TASK, 30
    PURIFY, 41
    PY_BUILTIN_MODULE_CFLAGS, 41
    PY_CFLAGS, 41
    PY_CFLAGS_NODIST, 41
    PY_CORE_CFLAGS, 41
    PY_CORE_LDFLAGS, 42
    PY_CPPFLAGS, 39
    PY_LDFLAGS, 42
    PY_LDFLAGS_NODIST, 42
    PY_PYTHON, 57
    PY_STDMODULE_CFLAGS, 41
    PYLAUNCHER_ALLOW_INSTALL, 58
    PYLAUNCHER_ALWAYS_INSTALL, 58
    PYLAUNCHER_DEBUG, 58
    PYLAUNCHER_DRYRUN, 58
    PYLAUNCHER_NO_SEARCH_PATH, 56
    PYTHON_BASIC_REPL, 17
    PYTHON_COLORS, 11, 17
    PYTHON_CPU_COUNT, 10, 16
    PYTHON_FROZEN_MODULES, 10, 16
    PYTHON_GIL, 10, 17, 94
    PYTHON_HISTORY, 17
    PYTHON_PERF_JIT_SUPPORT, 10, 16
    PYTHON_PRESITE, 10, 17
    PYTHONASYNCIODEBUG, 14
    PYTHONBREAKPOINT, 12
    PYTHONCASEOK, 12
    PYTHONCOERCECLOCALE, 15, 25
    PYTHONDEBUG, 6, 12, 32
    PYTHONDEVMODE, 9, 16
    PYTHONDONTWRITEBYTECODE, 6, 12
    PYTHONDUMPREFS, 17, 32
    PYTHONDUMPREFSFILE, 17
    PYTHONEXECUTABLE, 13
    PYTHONFAULTHANDLER, 9, 14
    PYTHONHASHSEED, 7, 12, 13
    PYTHONHOME, 6, 11, 59, 60
    PYTHONINSPECT, 7, 12
    PYTHONINTMAXSTRDIGITS, 9, 13
    PYTHONIOENCODING, 13, 15
    PYTHONLEGACYWINDOWSFSENCODING, 15
    PYTHONLEGACYWINDOWSSTDIO, 13, 15
    PYTHONMALLOC, 14, 15, 31
    PYTHONMALLOCSTATS, 14
    PYTHONNODEBUGRANGES, 10, 16
    PYTHONNOUSERSITE, 8, 13
    PYTHONOPTIMIZE, 7, 12
    PYTHONPATH, 6, 11, 53, 59, 60
    PYTHONPERFSUPPORT, 10, 16
    PYTHONPLATLIBDIR, 11
    PYTHONPROFILEIMPORTTIME, 9, 14
    PYTHONPYCACHEPREFIX, 10, 12
    PYTHONSAFEPATH, 7, 11
    PYTHONSTARTUP, 7, 11, 12
    PYTHONTRACEMALLOC, 9, 14
    PYTHONUNBUFFERED, 8, 12
    PYTHONUSERBASE, 13
    PYTHONUTF8, 9, 16, 54
    PYTHONVERBOSE, 8, 12
    PYTHONWARNDEFAULTENCODING, 10, 16
    PYTHONWARNINGS, 9, 13
    TEMP, 49
variável de classe, **90**

variável de clausura, **90**
variável de contexto, **90**
variável livre, **93**
verificador de tipo estático, **102**
--version
    opção de linha de comando, 6
visão de dicionário, **91**

W

-W
    opção de linha de comando, 8
--with-address-sanitizer
    opção de linha de comando, 33
--with-app-store-compliance
    opção de linha de comando, 36
--with-assertions
    opção de linha de comando, 32
--with-build-python
    opção de linha de comando, 36
--with-builtin-hashlib-hashes
    opção de linha de comando, 34
--with-computed-gotos
    opção de linha de comando, 31
--with-dbmliborder
    opção de linha de comando, 25
--with-dtrace
    opção de linha de comando, 33
--with-emscripten-target
    opção de linha de comando, 29
--with-ensurepip
    opção de linha de comando, 29
--with-framework-name
    opção de linha de comando, 36
--with-hash-algorithm
    opção de linha de comando, 34
--with-libc
    opção de linha de comando, 34
--with-libm
    opção de linha de comando, 34
--with-libs
    opção de linha de comando, 33
--with-lto
    opção de linha de comando, 30
--with-memory-sanitizer
    opção de linha de comando, 33
--with-openssl
    opção de linha de comando, 34
--with-openssl-rpath
    opção de linha de comando, 34
--without-c-locale-coercion
    opção de linha de comando, 25
--without-decimal-contextvar
    opção de linha de comando, 25
--without-doc-strings
    opção de linha de comando, 31
--without-mimalloc
    opção de linha de comando, 31
--without-pymalloc
    opção de linha de comando, 31
--without-readline
    opção de linha de comando, 34
--without-static-libpython
    opção de linha de comando, 33
--with-pkg-config
    opção de linha de comando, 25
--with-platlibdir
    opção de linha de comando, 25
--with-pydebug
    opção de linha de comando, 32
--with-readline
    opção de linha de comando, 34
--with-ssl-default-suites
    opção de linha de comando, 35
--with-strict-overflow
    opção de linha de comando, 31
--with-suffix
    opção de linha de comando, 24
--with-system-expat
    opção de linha de comando, 33
--with-system-libmpdec
    opção de linha de comando, 33
--with-thread-sanitizer
    opção de linha de comando, 33
--with-trace-refs
    opção de linha de comando, 32
--with-tzpath
    opção de linha de comando, 24
--with-undefined-behavior-sanitizer
    opção de linha de comando, 33
--with-universal-archs
    opção de linha de comando, 35
--with-valgrind
    opção de linha de comando, 33
--with-wheel-pkg-dir
    opção de linha de comando, 25

X

-X
    opção de linha de comando, 9
-x
    opção de linha de comando, 9

Z

Zen do Python, **104**
ZLIB_CFLAGS
    opção de linha de comando, 29
ZLIB_LIBS
    opção de linha de comando, 29