

Python

Origem: Wikipédia, a enciclopédia livre.


Python é uma linguagem de programação de alto nível,^[5] interpretada de script, imperativa, orientada a objetos, funcional, de tipagem dinâmica e forte. Foi lançada por Guido van Rossum em 1991.^[1] Atualmente, possui um modelo de desenvolvimento comunitário, aberto e gerenciado pela organização sem fins lucrativos Python Software Foundation. Apesar de várias partes da linguagem possuírem padrões e especificações formais, a linguagem, como um todo, não é formalmente especificada. O padrão na prática é a implementação CPython.

A linguagem foi projetada com a filosofia de enfatizar a importância do esforço do programador sobre o esforço computacional. Prioriza a legibilidade do código sobre a velocidade ou expressividade. Combina uma sintaxe concisa e clara com os recursos poderosos de sua biblioteca padrão e por módulos e frameworks desenvolvidos por terceiros.

Python é uma linguagem de propósito geral de alto nível, multiparadigma, suporta o paradigma orientado a objetos, imperativo, funcional e procedural. Possui tipagem dinâmica e uma de suas principais características é permitir a fácil leitura do código e exigir poucas linhas de código se comparado ao mesmo programa em outras linguagens. Devido às suas características, ela é utilizada, principalmente, para processamento de textos, dados científicos e criação de CGIs para páginas dinâmicas para a web. Foi considerada pelo público a 3ª linguagem "mais amada", de acordo com uma pesquisa conduzida pelo site Stack Overflow em 2018^[6] e está entre as 5 linguagens mais populares, de acordo com uma pesquisa conduzida pela RedMonk.^[7]

O nome **Python** teve a sua origem no grupo humorístico britânico Monty Python,^[8] criador do programa Monty Python's Flying Circus, embora muitas pessoas façam associação com o réptil do mesmo nome (em português, pítão ou pitão).

História

<div>Python</div> <div> python™</div>	
Paradigma	Multiparadigma: <u>orientada a objetos</u> · <u>imperativa</u> · <u>funcional</u>
Surgido em	20 de fevereiro de 1991 (32 anos) ^[1]
Última versão	3.11.0 (24 de outubro de 2022 ^[2])
Criado por	<u>Guido van Rossum</u> ^[1]
Estilo de tipagem	dinâmica · forte · <i>duck</i> · gradual (desde a versão 3.5)
Principais implementações	<u>CPython</u> · <u>IronPython</u> · <u>Jython</u> · <u>PyPy</u> · <u>Stackless</u>
Influenciada por	<u>ABC</u> ^[3] · <u>ALGOL 68</u> · <u>C</u> ^[3] · <u>Haskell</u> · <u>Icon</u> · <u>Java</u> · <u>Lisp</u> · <u>Modula-3</u> ^[3] · <u>Perl</u> · <u>Smalltalk</u>
Influenciou	<u>Boo</u> · <u>CoffeeScript</u> · <u>D</u> · <u>Fantom</u> · <u>GDScript</u> · <u>Go</u> · <u>Groovy</u> · <u>JavaScript</u> · <u>Julia</u> · <u>Nim</u> · <u>Py</u> · <u>Ruby</u> · <u>Squirrel</u> · <u>Swift</u>
Licença:	Python Software Foundation License ^[4]
Extensão do arquivo:	.py · .pyc · .pyd · .pyo · .pyw · .pyz
Página oficial	<u>www.python.org</u> (<u>http</u> <u>s://www.python.org/</u>)

O Python foi concebido no final de 1989^{[5][8]} por Guido van Rossum no Instituto de Pesquisa Nacional para Matemática e Ciência da Computação (CWI), nos Países Baixos, como um sucessor da ABC capaz de tratar exceções e prover interface com o sistema operacional Amoeba^[9] através de scripts. Também da CWI, a linguagem ABC era mais produtiva que C, ainda que com o custo do desempenho em tempo de execução. Mas ela não possuía funcionalidades importantes para a interação com o sistema operacional, uma necessidade do grupo. Um dos focos primordiais de Python era aumentar a produtividade do programador.^[8]



Guido van Rossum, São Francisco, Califórnia

Python foi feita com base na linguagem ABC, possui parte da sintaxe derivada do C, compreensão de listas, funções anônimas e função map de Haskell. Os iteradores são baseados na Icon, tratamentos de exceção e módulos da Modula-3, expressões regulares de Perl.

Em 1991, Guido publicou o código (nomeado versão 0.9.0) no grupo de discussão alt.sources.^[1] Nessa versão já estavam presentes classes com herança, tratamento de exceções, funções e os tipos de dado nativos list, dict, str, e assim por diante. Também estava presente nessa versão um sistema de módulos emprestado do Modula-3. O modelo de exceções também lembrava muito o do Modula-3, com a adição da opção else clause.^[9] Em 1994 foi formado o principal fórum de discussão do Python, comp.lang.python, um marco para o crescimento da base de usuários da linguagem.

A versão 1.0 foi lançada em janeiro de 1994. Novas funcionalidades incluíam ferramentas para programação funcional como lambda, map, filter e reduce. A última versão enquanto Guido estava na CWI foi o Python 1.2. Em 1995, ele continuou o trabalho no CNRI em Reston, Estados Unidos, de onde lançou diversas versões. Na versão 1.4 a linguagem ganhou parâmetros nomeados (a capacidade de passar parâmetro pelo nome e não pela posição na lista de parâmetros) e suporte nativo a números complexos, assim como uma forma de encapsulamento.^[10]

Ainda na CNRI, Guido lançou a iniciativa *Computer Programming for Everybody* (CP4E; literalmente, "Programação de Computadores para Todos"), que visava tornar a programação mais acessível, um projeto financiado pela DARPA.^[11] Atualmente o CP4E encontra-se inativo.

Em 2000, o time de desenvolvimento da linguagem se mudou para a BeOpen a fim de formar o time PythonLabs. A CNRI pediu que a versão 1.6 fosse lançada para marcar o fim de desenvolvimento da linguagem naquele local. O único lançamento na BeOpen foi o Python 2.0, e após o lançamento o grupo de desenvolvedores da PythonLabs agrupou-se na Digital Creations.

Python 2.0 implementou list comprehension, uma relevante funcionalidade de linguagens funcionais como SETL e Haskell. A sintaxe da linguagem para essa construção é bastante similar a de Haskell, exceto pela preferência do Haskell por caracteres de pontuação e da preferência do python por palavras reservadas alfabéticas. Essa versão 2.0 também introduziu um sistema coletor de lixo capaz de identificar e tratar ciclos de referências.^[12]

Já o 1.6 incluiu uma licença CNRI substancialmente mais longa que a licença CWI que estavam usando nas versões anteriores. Entre outras mudanças, essa licença incluía uma cláusula atestando que a licença era governada pelas leis da Virgínia. A Free Software Foundation alegou que isso era incompatível com a GNU GPL. Tanto BeOpen quanto CNRI e FSF negociaram uma mudança na licença livre do Python que o tornaria compatível com a GPL. Python 1.6.1 é idêntico ao 1.6.0, exceto por pequenas correções de falhas e uma licença nova, compatível com a GPL.^[13]

Python 2.1 era parecido com as versões 1.6.1 e 2.0. Sua licença foi renomeada para Python Software Foundation License.^[4] Todo código, documentação e especificação desde o lançamento da versão alfa da 2.1 é propriedade da Python Software Foundation (PSF), uma organização sem fins lucrativos fundada em 2001, um modelo tal qual da Apache Software Foundation.^[13] O lançamento incluiu a mudança na especificação para suportar escopo aninhado, assim como outras linguagens com escopo estático.^[14] Esta funcionalidade estava desativada por padrão, e somente foi requerida na versão 2.2.

Uma grande inovação da versão 2.2 foi a unificação dos tipos Python (escritos em C) e classes (escritas em Python) em somente uma hierarquia. Isto tornou o modelo de objetos do Python consistentemente orientado a objeto.^[15] Também foi adicionado generator, inspirado em Icon.^[16]

O incremento da biblioteca padrão e as escolhas sintáticas foram fortemente influenciadas por Java em alguns casos: o pacote logging^[17] introduzido na versão 2.3,^[18] o analisador sintático SAX, introduzido na versão 2.0 e a sintaxe de decoradores que usa @,^[19] adicionadas na versão 2.4.^[20]

Em 1 de outubro de 2008 foi lançada a versão 2.6, já visando a transição para a versão 3.0 da linguagem. Entre outras modificações, foram incluídas bibliotecas para multiprocessamento, JSON e E/S, além de uma nova forma de formatação de cadeias de caracteres.^[21]

Atualmente a linguagem é usada em diversas áreas, como servidores de aplicação e computação gráfica. Está disponível como linguagem de script em aplicações como OpenOffice (Python UNO Bridge), Blender e pode ser utilizada em procedimentos armazenados no sistema gerenciador de banco de dados PostgreSQL (PL/Python).

A terceira versão da linguagem foi lançada em dezembro de 2008,^[22] chamada Python 3.0 ou Python 3000. Com noticiado desde antes de seu lançamento,^[23] houve quebra de compatibilidade com a família 2.x para corrigir falhas que foram descobertas neste padrão, e para limpar os excessos das versões anteriores.^[8] A primeira versão alfa foi lançada em 31 de agosto de 2007, a segunda em 7 de dezembro do mesmo ano.

Mudanças da versão incluem a alteração da palavra reservada print, que passa a ser uma função, tornando mais fácil a utilização de uma versão alternativa da rotina. Em Python 2.6, isso já está disponível ao adicionar o código `from __future__ import print_function`.^[24] Também, a mudança para Unicode de todas as cadeias de caracteres.^[25]

Em 2012, foi criado o Raspberry Pi, cujo nome foi baseado na linguagem Python. Uma das principais linguagens escolhidas é Python. Python influenciou várias linguagens, algumas delas foram Boo e Cobra, que usa a indentação como definição de bloco e Go, que se baseia nos princípios de desenvolvimento rápido de Python.

Atualmente, Python é um dos componentes padrão de vários sistemas operacionais, entre eles estão a maioria das distribuições do Linux, AmigaOS 4, FreeBSD, NetBSD, OpenBSD e OS X. A linguagem se tornou a padrão no curso de ciências da computação do MIT em 2009

Filosofia

Parte da cultura da linguagem gira ao redor de *The Zen of Python*, um poema que faz parte do documento "PEP 20 (The Zen of Python)",^[26] escrito pelo programador em Python de longa data Tim Peters, descrevendo sumariamente a filosofia do Python. Entre os vinte princípios do poema, estão presentes:

- Bonito é melhor que feio;

- Explícito é melhor que implícito;
- Simples é melhor que complexo;
- Complexo é melhor que complicado;
- Legibilidade faz diferença.

Pode-se vê-lo através de um easter egg do Python pelo comando:

```
>>> import this
```

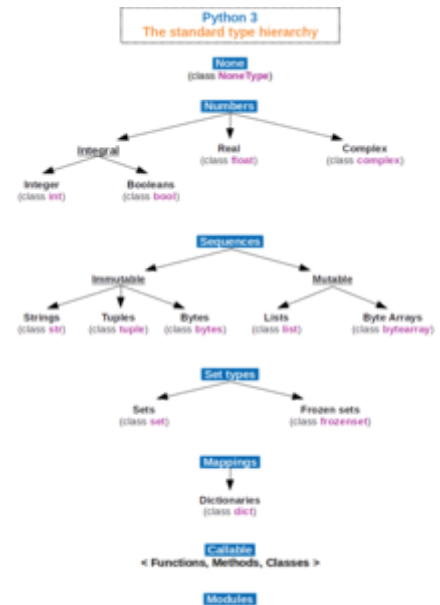
Sintaxe e semântica

Construções

Construções de Python incluem: estrutura de seleção (if, else, elif); estrutura de repetição (for, while), que itera por um container, capturando cada elemento em uma variável local dada; construção de classes (class); construção de sub-rotinas (def); construção de escopo (with), como por exemplo para adquirir um recurso.

Tipos de dado

A tipagem de Python é forte, pois os valores e objetos têm tipos bem definidos e não sofrem coerções como em C ou Perl. São disponibilizados diversos tipos de dados nativos:



Tipo de dado	Descrição	Exemplo da sintaxe
bool	Booleano	True ou False
int	Número de precisão fixa, é transparentemente convertido para long caso não caiba em um int.	42 2147483648L
float	Ponto flutuante	3.1415927
complex	Número complexo	3+2j
list	Lista heterogênea mutável	[4.0, 'string', True]
tuple	Tupla imutável	(4.0, 'string', True)
range	Sequência de números imutável que pode ser transformada em lista	range(10) range(0, 10) range(0, 10, 1)
set, frozenset	Conjunto não ordenado, não contém elementos duplicados	{4.0, 'string', True} frozenset([4.0, 'string', True])
str, unicode	Uma cadeia de caracteres imutável	'Wikipedia' u'Wikipedia'
bytes, bytearray, memoryview	Sequência binária	b'Wikipedia' bytearray(b'Wikipedia') memoryview(b'Wikipedia')
dict	Conjunto associativo	{'key1': 1.0, 'key2': False}

Python também permite a definição dos tipos de dados próprios, através de classes. Instâncias são construídas invocando a classe (`FOOClass()`), e as classes são instância da classe `type`, o que permite metaprogramação e reflexão. Métodos são definidos como funções anexadas à classe, e a sintaxe `instância.método(argumento)` é um atalho para `Classe.método(instância, argumento)`. Os métodos devem referenciar explicitamente a referência para o objeto incluindo o parâmetro `self` como o primeiro argumento do método.^[27]

Antes da versão 3.0, Python possuía dois tipos de classes: "old-style" e "new-style". Classes old-style foram eliminadas no Python 3.0, e todas são new-style. Em versões entre 2.2 e 3.0, ambos tipos de classes podiam ser usadas. A sintaxe de ambos estilos é a mesma, a diferença acaba sendo de onde objeto da classe é herdado, direta ou indiretamente (todas classes new-style herdam de `object` e são instancias de `type`). As classes new-styles nada mais são que tipos definidos pelo usuário.

Palavras reservadas

O Python 3 define as seguintes palavras reservadas:^[28]

```
False      None      True      and      as
assert     break     class     continue
def         del         elif      else     except
finally    for         from      global   if
import     in          is        lambda   not
nonlocal   or          pass      raise    try
return     while       with      yield
```

A versão 3.10.0 implementou a *Structural Pattern Matching* (Correspondência de Padrão Estrutural), semelhante ao Switch-Case de outras linguagens, assim como definido na PEP 634 (<https://www.python.org/dev/peps/pep-0634/>). Por isso as palavras `match` e `case` passarão a ser reservadas.

Operadores

Os operadores básicos de comparação como `==`, `<`, `>=`, entre outros são usados em todos os tipos de dados, como números, cadeias de texto, listas e mapeamentos. Comparações em cadeia como `a < b < c` possuem o mesmo significado básico que na matemática: os termos são comparadas na ordem. É garantido que o processamento da expressão lógica irá terminar tão cedo o veredito seja claro, o princípio da avaliação mínima. Usando a expressão anterior, se `a < b` é falso, `c` não é avaliado.

Quanto aos operadores lógicos, até Python 2.2 não havia o tipo de dado booleano. Em todas as versões da linguagem os operadores lógicos tratam `"", 0, None, 0.0, []` e `{}` como falso, enquanto o restante é tratado como verdadeiro de modo geral. Na versão 2.2.1 as constantes `True` e `False` foram adicionadas (subclasses de 1 e 0 respectivamente). A comparação binária retorna uma das duas constantes acima.

Os operadores booleanos `and` e `or` também seguem a avaliação mínima. Por exemplo, `y == 0 or x/y > 100` nunca lançará a exceção de divisão por zero.

Interpretador interativo

O interpretador interativo é uma característica diferencial da linguagem, porque há a possibilidade de testar o código de um programa e receber o resultado em tempo real, antes de iniciar a compilação ou incluí-las nos programas. Por exemplo:

```
>>> 1+1
2
>>>
>>> a = 1+1
>>> print a
2
>>> print(a)
2
>>>
```

Análise léxica

No segundo capítulo do *Manual de Referência da Linguagem Python* é citado que a análise léxica é uma análise do interpretador em si, os programas são lidos por um analisador sintático que divide o código em tokens.

```
#A is height B is radius
```

```
def cone (a, b):
    formula = (3.14 * .33 *a)*(b * b)
    return formula
```

Exemplo de script

Todo programa é dividido em linhas lógicas que são separadas pelo *token NEWLINE* ou *NOVA LINHA*, as linhas físicas são trechos de código divididos pelo caractere *ENTER*. Linhas lógicas não podem ultrapassar linhas físicas com exceção de junção de linhas, por exemplo:

```
if resultado > 2 and \
    1 <= 5 and \
```

```
2 < 5:
print ('Resultado: %f' % d)
```

ou

```
MESES_DO_ANO = ['janeiro', 'fevereiro', 'março',
                 'abril', 'maio', 'junho',
                 'julho', 'agosto', 'setembro',
                 'outubro', 'novembro', 'dezembro']
```

Para a delimitação de blocos de códigos, os delimitadores são colocados em uma pilha e diferenciados por sua indentação, iniciando a pilha com valor 0 (zero) e colocando valores maiores que os anteriores na pilha. Para cada começo de linha, o nível de indentação é comparado com o valor do topo da pilha. Se o número da linha for igual ao topo da pilha, a pilha não é alterada. Se o valor for maior, a pilha recebe o nível de indentação da linha e o nome *INDENT* (empilhamento). Se o nível de indentação for menor, então é desempilhado até chegar a um nível de indentação recebendo o nome *DEDENT* (desempilhamento). Se não encontrar nenhum valor, é gerado um erro de indentação.

Abaixo um exemplo de permutação, retirado do capítulo 2.1 sobre Estrutura de linhas na Análise léxica do Manual de Referência da linguagem (*Language Reference Manual*):

```
def perm(l):
    NOVA LINHA
    if len(l) <= 1:
        NOVA LINHA
        return l
    NOVA LINHA
    NOVA LINHA
    r = [ ]
    NOVA LINHA
    for i in range(len(l)):
        NOVA LINHA
        NOVA LINHA
        s = l[:i] + l[i+1:]
        NOVA LINHA
        p = perm(s)
        NOVA LINHA
        NOVA LINHA
        for x in p:
            NOVA LINHA
            NOVA LINHA
            r.append(l[:i+1]+x)
            NOVA LINHA
        NOVA LINHA
    return r
```

Indentação

Python foi desenvolvido para ser uma linguagem de fácil leitura, com um visual agradável, frequentemente usando palavras e não pontuações como em outras linguagens. Para a separação de blocos de código, a linguagem usa espaços em branco e indentação ao invés de delimitadores visuais como chaves (C, Java) ou palavras (BASIC, Fortran, Pascal). Diferente de linguagens com delimitadores visuais de blocos, em Python a indentação é obrigatória. O aumento da indentação indica o início de um novo bloco, que termina da diminuição da indentação.

Usando um editor de texto comum é muito fácil existir erros de indentação, o recomendado é configurar o editor conforme a análise léxica do Python ou utilizar uma IDE. Todas as IDE que suportam a linguagem fazem indentação automaticamente.

Exemplo:

Indentação correta

```
def valor1():
    while True:
        try:
            c = int(input('Primeiro Valor: '))
        except:
            pass
```

Indentação incorreta

```
def valor1():
    while True:
        try:
            c = int(input('Primeiro Valor: '))
        except:
            pass
    return c
```

```
return c
except ValueError:
    print 'Inválido!'
```

```
except ValueError:
    print 'Inválido!'
```

O código está correto para os dois exemplos, mas o analisador léxico verificará se a indentação está coerente. O analisador reconhecerá as palavras reservadas `while`, `def`, `try`, `except`, `return`, `print` e as cadeias de caracteres entre aspas simples e a indentação, e se não houver problemas o programa executará normalmente, senão apresentará a exceção: "Seu programa está com erro no bloco de indentação".

Na internet, há uma comparação de velocidade e de codificação entre as linguagens Python e BASIC, esta última, o dialeto BBC BASIC for Windows.

Compilador de bytecode

A linguagem é de altíssimo nível, como já dito, mas ela também pode compilar seus programas para que a próxima vez que o executar não precise compilar novamente o programa, reduzindo o tempo de carga na execução.

Utilizando o interpretador interativo não é necessário a criação do arquivo de Python compilado, os comandos são executados interativamente. Porém quando um programa ou um módulo é evocado, o interpretador realiza a análise léxica e sintática, compila o código de alto nível se necessário e o executa na máquina virtual da linguagem.

O *bytecode* é armazenado em arquivos com extensão `.pyc` ou `.pyo`, este último no caso de *bytecode* otimizado. Interessante notar que o *bytecode* da linguagem também é de alto nível, ou seja, é mais legível aos seres humanos que o código de byte do C, por exemplo. Para descompilar um código de byte é utilizado o módulo `dif` da biblioteca padrão da linguagem e existem módulos de terceiros que tornam o *bytecode* mais confuso, tornando a descompilação ineficaz.

Normalmente, o Python trabalha com dois grupos de arquivos:

1. Os módulos do núcleo da linguagem, sua biblioteca padrão e os módulos independentes, criados pelo usuário.
2. No núcleo do interpretador existe o analisador léxico, o analisador sintático que utiliza *Estruturas de Objetos* (tempo de execução), o Compilador que *aloca memória* (tempo de execução) e depois do Avaliador de código que modifica o *estado atual* do programa (tempo de execução), mostrando resultado para o usuário.

Orientação a objetos

Python suporta a maioria das técnicas da programação orientada a objeto. Qualquer objeto pode ser usado para qualquer tipo, e o código funcionará enquanto haja métodos e atributos adequados. O conceito de objeto na linguagem é bastante abrangente: classes, funções, números e módulos são todos considerados objetos. Também há suporte para metaclasses, polimorfismo, e herança (inclusive herança múltipla). Há um suporte limitado para variáveis privadas.

Na versão 2.2 de Python foi introduzido um novo estilo de classes em que objetos e tipos foram unificados, permitindo a especialização de tipos. Já a partir da versão 2.3 foi introduzido um novo método de resolução de ambiguidades para heranças múltiplas.^[30]

Uma classe é definida com `class nome:`, e o código seguinte é a composição dos atributos. Todos os métodos da classe recebem uma referência a uma instância da própria classe como seu primeiro argumento, e a convenção é que se chame este argumento `self`. Assim os métodos são chamados `objeto.método(argumento1, argumento2, ...)` e são definidos iguais a uma função, como `método(self, argumento1, argumento2, ...)`. Veja que o parâmetro `self` conterá uma referência para a instância da classe definida em `objeto` quando for efetuada esta chamada. Os atributos da classe podem ser acessados em qualquer lugar da classe, e os atributos de instância (ou variável de instância) devem ser declarados dentro dos métodos utilizando a referência à instância atual (`self`) (ver código contextualizado em anexo).

Em Python não existe proteção dos membros duma classe ou instância pelo interpretador, o chamado encapsulamento. Convenciona-se que atributos com o nome começando com um `_` são de uso privado da classe, mas não há um policiamento do interpretador contra acesso a estes atributos. Uma exceção são nomes começando com `__`, no caso em que o interpretador modifica o nome do atributo (ver código contextualizado em anexo).

Python permite polimorfismo, que condiz com a reutilização de código. É fato que funções semelhantes em várias partes do software sejam utilizadas várias vezes, então definimos esta função como uma biblioteca e todas as outras funções que precisarem desta a chamam sem a necessidade de reescrevê-la (ver código contextualizado em anexo).

Python não possui overloading; não é possível criar duas funções com o mesmo nome, pois elas são consideradas atributos da classe. Caso o nome da função se repita em outra assinatura, o interpretador considera esta última como override e sobrescreve a função anterior. Algumas operações entre diferentes tipos são realizadas através de coerção (ex.: $3.2 + 3$).

É possível encapsular abstrações em módulos e pacotes. Quando um arquivo é criado com a extensão `.py`, ele automaticamente define um módulo. Um diretório com vários módulos é chamado de pacote e deve conter um módulo chamado `__init__`, para defini-lo como principal. Estas diferenciações ocorrem apenas no sistema de arquivos. Os objetos criados são sempre módulos. Caso o código não defina qual dos módulos será importado, o padrão é o `__init__`.

Programação funcional

Uma das construções funcionais de Python é compreensão de listas, uma forma de construir listas. Por exemplo, pode-se usar a técnica para calcular as cinco primeiras potências de dois. O algoritmo quicksort também pode ser expressado usando a mesma técnica (ver códigos contextualizados para ambos os casos em anexo).

Em Python, funções são objetos de primeira classe que podem ser criados e armazenados dinamicamente. O suporte a funções anônimas está na construção `lambda` (cálculo Lambda). Não há disponibilidade de funções anônimas de fato, pois os lambdas contêm somente expressões e não blocos de código.

Python também suporta clausuras léxicas desde a versão 2.2 (ver códigos contextualizados para ambos os casos em anexo). Já geradores foram introduzidos na versão 2.2 e finalizados na versão 2.3, e representam o mecanismo de Python para a avaliação preguiçosa de funções (ver códigos contextualizados para ambos os casos em anexo).

Tratamento de exceções

Python suporta e faz uso constante de tratamento de exceções como uma forma de testar condições de erro e outros eventos inesperados no programa. É inclusive possível capturar uma exceção causada por um erro de sintaxe. O estilo da linguagem apóia o uso de exceções sempre que uma condição de erro pode aparecer. Por exemplo, ao invés de testar a disponibilidade de acesso a um recurso, a convenção é simplesmente tentar usar o recurso e capturar a exceção caso o acesso seja rejeitado (recurso inexistente, permissão de acesso insuficiente, recurso já em uso, ...).

Exceções são usadas frequentemente como uma estrutura de seleção, substituindo blocos `if-else`, especialmente em situações que envolvem *threads*. Uma convenção de codificação é o EAFP, do inglês, "é mais fácil pedir perdão que permissão". Isso significa que em termos de desempenho é preferível capturar exceções do que testar atributos antes de os usar. Segue abaixo exemplos de código que testam atributos ("pedem permissão") e que capturam exceções ("pedem perdão"):

Teste de atributo

```
if hasattr(spam, 'eggs'):
    ham = spam.eggs
else:
    handle_error()
```

Captura de exceção

```
try:
    ham = spam.eggs
except AttributeError:
    handle_error()
```

Ambos os códigos produzem o mesmo efeito, mas há diferenças de desempenho. Quando `spam` possui o atributo `eggs`, o código que captura exceções é mais rápido. Caso contrário, a captura da exceção representa uma perda considerável de desempenho, e o código que testa o atributo é mais rápido. Na maioria dos casos o paradigma da captura de exceções é mais rápido, e também pode evitar problemas de concorrência.^[31] Por exemplo, num ambiente multitarefa, o espaço de tempo entre o teste do atributo e seu uso de fato pode invalidar o atributo, problema que não acontece no caso da captura de exceções.

Biblioteca padrão

Python possui uma grande biblioteca padrão, geralmente citada como um dos maiores trunfos da linguagem,^[32] fornecendo ferramentas para diversas tarefas. Por conta da grande variedade de ferramentas fornecida pela biblioteca padrão, combinada com a habilidade de usar linguagens de nível mais baixo como C e C++, Python pode ser poderosa para conectar componentes diversos de software.

A biblioteca padrão conta com facilidades para escrever aplicações para a Internet, contando com diversos formatos e protocolos como MIME e HTTP. Também há módulos para criar interfaces gráficas, conectar em bancos de dados relacionais e manipular expressões regulares.

Algumas partes da biblioteca são cobertas por especificações (por exemplo, a implementação WSGI da `wsgiref` segue o PEP 333^[33]), mas a maioria dos módulos não segue.

Interoperabilidade

Um outro ponto forte da linguagem é sua capacidade de interoperar com várias outras linguagens, principalmente código nativo. A documentação da linguagem inclui exemplos de como usar a Python C-API para escrever funções em C que podem ser chamadas diretamente de código Python - mas atualmente esse sequer é o modo mais indicado de interoperação, havendo alternativas tais como Cython, Swig ou ffi (<https://cffi.readthedocs.org/en/latest/>). A biblioteca Boost do C++ inclui uma biblioteca para permitir a interoperabilidade entre as duas linguagens, e pacotes científicos fazem uso de bibliotecas de alta performance numérica escritos em Fortran e mantidos há décadas.

Comentários

Python fornece duas alternativas para documentar o código. A primeira é o uso de comentários para indicar o que certo código faz. Comentários começam com `#` e são terminados pela quebra da linha. Não há suporte para comentários que se estendem por mais de uma linha; cada linha consecutiva de comentário deve indicar `#`. A segunda alternativa é o uso de cadeias de caractere, literais de texto inseridos no código sem atribuição. Cadeias de caracteres em Python são delimitadas por `"` ou `'` para única linha e por `"""` ou `'''` para múltiplas linhas. Entretanto, é convenção usar o métodos de múltiplas linhas em ambos os casos.

Diferente de comentários, a cadeias de caracteres usadas como documentação são objetos Python e fazem parte do código interpretado. Isso significa que um programa pode acessar sua própria documentação e manipular a informação. Há ferramentas que extraem automaticamente essa documentação para a geração da documentação de API a partir do código. Documentação através de cadeias de caracteres também pode ser acessada a partir do interpretador através da função `help()`.

Plataformas disponíveis

A linguagem e seu interpretador estão disponíveis para as mais diversas plataformas, desde Unix (Linux, FreeBSD, Solaris, MacOS X, etc.), Windows, .NET, versões antigas de MacOS até consoles de jogos eletrônicos ou mesmo alguns celulares, como a série 60, N8xx(PyMaemo) da Nokia e palmtops.

Para algum sistema operacional não suportado, basta que exista um compilador C disponível e gerar o Python a partir do fonte. O código fonte é traduzido pelo interpretador para o formato bytecode, que é multiplataforma e pode ser executado e distribuído sem fonte original.

Implementações

A implementação original e mais conhecida do Python é o CPython, escrita em C e compatível com o padrão C89,^[34] sendo distribuída com uma grande biblioteca padrão escrita em um misto de Python e C. Esta implementação é suportada em diversas plataformas, incluindo Microsoft Windows e sistemas Unix-like modernos.

Stackless Python é uma variação do CPython que implementa microthreads (permitindo multitarefa sem o uso de threads), sendo suportada em quase todas as plataformas que a implementação original.

Existem também implementações para plataformas já existentes: Jython para a Plataforma Java e IronPython para .NET.

Em 2005 a Nokia lançou um interpretador Python para os telefones celulares S60, chamado PyS60. Essa versão inclui vários módulos das implementações tradicionais, mas também alguns módulos adicionais para a integração com o sistema operacional Symbian. Uma implementação para Palm pode ser encontrada no Pippy. Já o PyPy, é a linguagem Python totalmente escrita em Python.

Diversas implementações, como CPython, pode funcionar como um interpretador de comandos em que o usuário executa as instruções sequencialmente, recebendo o resultado automaticamente. A execução compilada do código oferece um ganho substancial em velocidade, com o custo da perda da interatividade.

Desenvolvimento

O desenvolvimento de Python é conduzido amplamente através do processo Python Enhancement Proposal ("PEP"), em português Proposta de Melhoria do Python. Os PEPs são documentos de projeto padronizados que fornecem informações gerais relacionadas ao Python, incluindo propostas, descrições, justificativas de projeto (design rationales) e explicações para características da linguagem. PEPs pendentes são revisados e comentados por Van Rossum, o Benevolent Dictator for Life (líder arquiteto da linguagem) do projeto Python. Desenvolvedores do CPython também se comunicam através de uma lista de discussão, python-dev, que é o fórum principal para discussão sobre o desenvolvimento da linguagem. Questões específicas são discutidas no gerenciador de erros Roundup mantido em python.org. O desenvolvimento acontece no auto-hospedado svn.python.org

Licença

Python possui uma licença livre aprovada pela OSI e compatível com a GPL, porém menos restritiva. Ela prevê (entre outras coisas) que binários da linguagem sejam distribuídos sem a necessidade de fornecer o código fonte junto.^[4]

Módulos e *frameworks*

Ao longo do tempo têm sido desenvolvidos pela comunidade de programadores muitas bibliotecas de funções especializadas (módulos) que permitem expandir as capacidades base da linguagem. Entre estes módulos especializados destacam-se:

	Descrição	Campos de atuação
<u>Django</u>	<i>Framework</i> para desenvolvimento ágil de aplicações <u>web</u> ;	<u>desenvolvimento web</u>
<u>Pylons</u>	<i>Framework</i> para desenvolvimento de aplicações <u>web</u> ;	desenvolvimento web
<u>TurboGears</u>	<i>Framework</i> baseado em várias outras tecnologias existentes no mundo que gira em torno da linguagem Python;	desenvolvimento web
<u>Matplotlib - Matplotlib / Pylab</u>	biblioteca para manipulação de gráficos 2D;	<u>processamento de imagem</u>
<u>Python Imaging Library</u>	biblioteca para manipulação de <u>imagens digitais</u> ;	processamento de imagem
<u>PyOpenGL - Python OpenGL Binding</u>	suporte <u>multiplataforma</u> ao <u>OpenGL</u> ;	<u>computação gráfica</u>
<u>Pygame</u>	Conjunto de módulos para o desenvolvimento de <u>jogos eletrônicos</u> , incluindo gráficos <u>SDL</u> ;	<u>desenvolvimento de jogos eletrônicos</u> ; <u>computação gráfica</u>
<u>Twisted</u>	<i>Framework</i> para o desenvolvimento de aplicações de rede. Inclui módulos para servidor <u>web</u> , de <u>aplicação</u> , <u>SSH</u> e diversos outros <u>protocolos</u> ;	<u>desenvolvimento de software</u> ; <u>desenvolvimento web</u>
<u>PYRO - Python Remote Objects</u>	<i>Framework</i> para o desenvolvimento de <u>sistemas distribuídos</u> ;	<u>computação distribuída</u>
<u>ZODB</u>	Sistema de <u>persistência</u> e <u>banco de dados orientado a objetos</u> ;	<u>banco de dados</u>
<u>Plone</u>	SGC - <u>Sistema de gerenciamento de conteúdo</u> ;	desenvolvimento web
<u>CherryPy</u>	<i>Framework</i> para aplicações <u>web</u> ;	desenvolvimento web
<u>Web2py</u>	<i>Framework</i> para aplicações <u>web</u> ;	desenvolvimento web
<u>Visual Python</u>	<i>Framework</i> 3D de alto nível;	computação gráfica
<u>SQLObject</u>	<u>Mapeador objeto-relacional</u> : traduz estruturas relacionais para objetos Python e manipula o <u>banco de dados</u> de forma transparente;	banco de dados
<u>Numarray</u>	Módulo para manipulação de vetores e <u>computação científica</u> .	<u>computação científica</u>

Interfaces gráficas

Exemplos de bibliotecas de GUI disponíveis para Python incluem:

	Descrição
<u>Tkinter</u>	Módulo padrão para GUI no Python
<u>PyGTK</u>	interface para a biblioteca <u>GTK+</u>
<u>PyQt</u>	interface para a biblioteca <u>Qt</u>
<u>wxPython</u>	interface para a biblioteca <u>wxWidgets</u>
<u>Etk</u>	interface para a biblioteca <u>EFL</u>
<u>Wax</u>	Construído para simplificar o uso do wxPython
<u>Kivy</u> (https://kivy.org/)	<u>Toolkit</u> multiplataforma

Ambientes de desenvolvimento integrado

Existem vários ambientes de desenvolvimento integrado (IDE) disponíveis para Python:

IDE	Desenvolvedor	Última versão	Plataforma	Toolkit	Licença
IDLE	Guido van Rossum et al.	Distribuído com CPython	Multiplataforma	Tkinter	PSFL
PyCharm	JetBrains	2017.3 (29/11/2017)	Java	Swing	Apache 2.0
Komodo Edit	ActiveState	10 (17/05/2016)	Windows, Linux, macOS	XUL	MPL 1.1
Atom	GitHub	1.25.0 (15/03/2018)	Windows, Linux, macOS	Electron	MIT
GNOME Builder	GNOME Project	3.36.0 (06/03/2020)	Linux	GTK	GNU GPLv3+
Pyzo	Pyzo team	4.4.3 (09/10/2017)	Multiplataforma	PyQt	BSD
Boa Constructor	Team	0.6.1	Independente	wxPython	GNU GPL
Eric Python IDE	Detlev Offenbach	4.1.2	Independente	Qt	GNU GPL
Geany	Team	1.23	Independente	GTK2	GNU GPL
IronPython Studio	Clarius Labs	1.0 (10/12/2007)	Windows	VS2008 Shell Runtime	Microsoft Public License
PyDev (Eclipse)	Appcelerator	5.7.0 (11/04/2017)	Java	SWT	EPL
PythonCard	Alex Tweedly	0.8.2	Multiplataforma	wxPython	BSD
PyScripter	mmm-experts	1.7.2 (10/2006)	Windows		MIT
Stani's Python Editor	Stani	0.8.4c (14/02/2008)	Independente	wxPython	GNU GPL
Spyder (https://github.com/spyder-ide/spyder)	Spyder developer community (https://groups.google.com/group/spyderlib)	2.3.2 (03/12/2014)	Windows, Linux, macOS	PyQt	MIT
Wing IDE	Wingware	3.0.2-1 (27/11/2007)	Windows, Linux, macOS	PyGTK	Proprietário

Aplicações

Alguns dos maiores projetos que utilizam Python são o servidor de aplicação [Zope](#), o compartilhador de arquivos [Mnet](#), o sítio [YouTube](#) e o cliente original do [BitTorrent](#). Grandes organizações que usam a linguagem incluem [Google](#)^[35] (parte dos *crawlers*), [Yahoo!](#) (para o sítio de grupos de usuários) e [NASA](#).^[36] O sistema de gerenciamento de reservas da [Air Canada](#) também usa Python em alguns de seus componentes.^[37] A linguagem também tem bastante uso na indústria da [segurança da informação](#).

A linguagem tem sido embarcada como linguagem de script em diversos softwares, como em programas de edição tridimensional como [Maya](#),^[38] [Autodesk Softimage](#), [TrueSpace](#) e [Blender](#).^[39] Programas de edição de imagem também a usam para scripts, como o [GIMP](#).^[40] Para diversos sistemas operacionais a linguagem

já é um componente padrão, estando disponível em diversas distribuições Linux. O Red Hat Linux usa Python para instalação, configuração e gerenciamento de pacotes.

Outros exemplos incluem o Plone, sistema de gerenciamento de conteúdo desenvolvido em Python e Zope e a Industrial Light & Magic,^[41] que produz filmes da série Star Wars usando extensivamente Python para a computação gráfica nos processos de produção dos filmes.

Ver também

- Django (framework web)
- Nim (linguagem de programação)
- Perl
- Ruby (linguagem de programação)

Referências

1. «HISTORY» (https://web.archive.org/web/20160217132249/http://svn.python.org/view/*checkout*/python/trunk/Misc/HISTORY). *Fonte do Python* (em inglês). Python Software Foundation. Consultado em 5 de junho de 2008. Arquivado do original (http://svn.python.org/view/*checkout*/python/trunk/Misc/HISTORY) em 17 de fevereiro de 2016
2. «Python 3.11.0» (<https://www.python.org/downloads/release/python-3110/>). *www.python.org* (em inglês). 24 de outubro de 2022. Consultado em 26 de outubro de 2022
3. Guido van Rossum (Maio de 1996). «Foreword for "Programming Python" (1st ed.)» (<https://www.python.org/doc/essays/foreword/>) (em inglês). Python Software Foundation. Consultado em 12 de junho de 2008
4. «History and License» (<https://docs.python.org/3/license.html>) (em inglês). Python Software Foundation. Consultado em 7 de abril de 2020
5. «The Making of Python» (<http://www.artima.com/intv/pythonP.html>) (em inglês). Artima Developer. Consultado em 22 de março de 2007
6. «Stack Overflow Developer Survey 2018» (<https://insights.stackoverflow.com/survey/2018#technology-most-loved-dreaded-and-wanted-languages>). *Stack Overflow*. Consultado em 16 de abril de 2018
7. O'Grady, Stephen (7 de março de 2018). «The RedMonk Programming Language Rankings: January 2018» (<https://redmonk.com/sograde/2018/03/07/language-rankings-1-18/>) (em inglês). RedMonk. Consultado em 13 de março de 2018
8. Naomi Hamilton (5 de agosto de 2008). «The A-Z of Programming Languages: Python» (http://www.computerworld.com.au/article/255835/a-z_programming_languages_python/?fp=4194304&fpid=1&pf=1) (em inglês). Computerworld. Consultado em 17 de agosto de 2008
9. «Why was Python created in the first place?» (<https://docs.python.org/faq/general#why-was-python-created-in-the-first-place>) (em inglês). Python FAQ. Consultado em 22 de março de 2007
10. «LJ #37: Python 1.4 Update» (<https://web.archive.org/web/20070501080219/http://www.amk.ca/python/writing/12-14>) (em inglês). Consultado em 29 de abril de 2007. Arquivado do original (<http://www.amk.ca/python/writing/12-14>) em 1 de maio de 2007
11. Guido van Rossum. «Computer Programming for Everybody» (<https://web.archive.org/web/20090223101648/http://python.org/doc/essays/cp4e.html>) (em inglês). Consultado em 22 de março de 2007. Arquivado do original (<https://www.python.org/doc/essays/cp4e.html>) em 23 de fevereiro de 2009

12. A.M. Kuchling and Moshe Zadka. «What's New in Python 2.0» (<https://web.archive.org/web/20091214142515/http://www.amk.ca/python/2.0>) (em inglês). Consultado em 22 de março de 2007. Arquivado do original (<http://www.amk.ca/python/2.0>) em 14 de dezembro de 2009
13. «History of the software» (<https://docs.python.org/release/2.5/lib/node951.html>). *Referência da Biblioteca Python* (em inglês). Consultado em 22 de março de 2007
14. Jeremy Hylton. «Statically Nested Scopes» (<https://www.python.org/dev/peps/pep-0227/>) (em inglês). Consultado em 22 de março de 2007
15. «2 PEPs 252 and 253: Type and Class Changes» (<https://docs.python.org/release/2.2.3/whatsnew/sect-rellinks.html>) (em inglês). Python Software Foundation. Consultado em 11 de junho de 2008
16. «4 PEP 255: Simple Generators» (<https://docs.python.org/release/2.2.3/whatsnew/node5.html>) (em inglês). Python Software Foundation. Consultado em 11 de junho de 2008
17. «PEP 282 - A Logging System» (<https://www.python.org/dev/peps/pep-0282/>) (em inglês). Python Software Foundation. Consultado em 11 de junho de 2008
18. «8 PEP 282: The logging Package» (<https://docs.python.org/release/2.3/whatsnew/node9.html>) (em inglês). Python Software Foundation. Consultado em 11 de junho de 2008
19. «PEP 318 - Decorators for Functions and Methods» (<https://www.python.org/dev/peps/pep-0318/>) (em inglês). Python Software Foundation. Consultado em 11 de junho de 2008
20. «5 PEP 318: Decorators for Functions and Methods» (<https://docs.python.org/release/2.4/whatsnew/node6.html>) (em inglês). Python Software Foundation. Consultado em 11 de junho de 2008
21. A.M. Kuchling (1 de outubro de 2008). «What's New in Python 2.6» (<https://docs.python.org/whatsnew/2.6.html>) (em inglês). Python Software Foundation. Consultado em 3 de outubro de 2008
22. «Python 3.0 Release» (<https://www.python.org/download/releases/3.0/>) (em inglês). Python Software Foundation. Consultado em 3 de dezembro de 2008
23. Sarah Stokely (1 de fevereiro de 2008). «Python 3.0 to be backwards incompatible» (<http://www.itnews.com.au/News/69326,breaking-the-python-code.aspx>) (em inglês). iNews. Consultado em 11 de junho de 2008
24. Georg Brandl. «Make print a function» (<https://www.python.org/dev/peps/pep-3105/>) (em inglês). Consultado em 3 de outubro de 2008
25. «Diferenças entre Python 2 e Python 3» (<https://blog.caelum.com.br/quais-as-diferencas-entre-python-2-e-python-3/>). Blog da Caelum. Consultado em 16 de março de 2019
26. «PEP 20 - The Zen of Python» (<https://www.python.org/dev/peps/pep-0020/>) (em inglês). Python - Núcleo de Desenvolvimento. Consultado em 15 de janeiro de 2010
27. «Classes — Random Remarks» (<https://docs.python.org/tutorial/classes.html#random-remarks>). *Python Documentation* (em inglês). Python Software Foundation
28. DOWNEY, Allen B. *Pense em Python*. [S.l.]: Novatec. 38 páginas. ISBN 9788575225080
29. «What's New In Python 3.0» (<https://docs.python.org/release/3.0/whatsnew/3.0.html>) (em inglês). Python Software Foundation. Consultado em 15 de janeiro de 2011
30. Michele Simionato. «The Python 2.3 Method Resolution Order» (<https://www.python.org/download/releases/2.3/mro/>) (em inglês). Python Software Foundation. Consultado em 11 de junho de 2008
31. «EAFP vs LBYL (was Re: A little disappointed so far)» (<https://web.archive.org/web/20070929122422/http://mail.python.org/pipermail/python-list/2003-May/205182.html>). web.archive.org. Consultado em 6 de maio de 2012. Arquivado do original (<http://mail.python.org/pipermail/python-list/2003-May/205182.html>) em 29 de setembro de 2007
32. Przemyslaw Piotrowski (Julho de 2006). «Build a Rapid Web Development Environment for Python Server Pages and Oracle» (<http://www.oracle.com/technology/pub/articles/piotrowski-pythoncore.html>) (em inglês). Oracle. Consultado em 11 de junho de 2008

33. Phillip J. Eby (7 de dezembro de 2003). «PEP 333 -- Python Web Server Gateway Interface v1.0» (<https://www.python.org/dev/peps/pep-0333/>) (em inglês). Python Software Foundation. Consultado em 11 de junho de 2008
34. Guido van Rossum (5 de julho de 2001). «PEP 7 -- Style Guide for C Code» (<https://www.python.org/dev/peps/pep-0007/>) (em inglês). Python Software Foundation. Consultado em 11 de junho de 2008
35. «Quotes about Python» (<https://python.org/about/quotes/>) (em inglês). Python Software Foundation. Consultado em 11 de junho de 2008
36. Daniel G. Shafer (17 de janeiro de 2003). «Python Streamlines Space Shuttle Mission Design» (<https://www.python.org/about/success/usa/>) (em inglês). Python Software Foundation. Consultado em 11 de junho de 2008
37. Darryl K. Taft (5 de março de 2005). «Python Slithers into Systems» (<http://www.eweek.com/c/a/Application-Development/Python-Slithers-into-Systems/>) (em inglês). eWEEK. Consultado em 11 de junho de 2008
38. «Introduction to Maya Python API» (http://www.autodesk.com/us/maya/docs/Maya85/whelp/wwhimpl/common/html/wwhelp.htm?context=DeveloperResources&file=Introduction_to_Maya_Python_API.html). *Documentação do Maya* (em inglês). Autodesk. Consultado em 18 de julho de 2008
39. «Python Scripts» (<https://web.archive.org/web/20120618153326/http://wiki.blender.org/index.php/Extensions:Py/Scripts>) (em inglês). Blender. Consultado em 18 de julho de 2008. Arquivado do original (<http://wiki.blender.org/index.php/Extensions:Py/Scripts>) em 18 de junho de 2012
40. James Henstridge (16 de maio de 2006). «GIMP Python Documentation» (<http://www.gimp.org/docs/python/index.html>). *Documentação do GIMP* (em inglês). GIMP. Consultado em 18 de julho de 2008
41. Robin Rowe (1 de julho de 2002). «Industrial Light & Magic» (<http://www.linuxjournal.com/article/6011>) (em inglês). Linux Journal. Consultado em 18 de julho de 2008

Bibliografia

- Pilgrim, Mark (2004). *Dive into Python* (<http://diveintopython.net>) (em inglês) 2 ed. Nova Iorque: Apress. 413 páginas. ISBN 978-1-5905-9356-1
- Pilgrim, Mark (2009). *Dive into Python 3* (<http://www.diveinto.org/python3/>) (em inglês) 2 ed. Nova Iorque: Apress. 360 páginas. ISBN 978-1-4302-2415-0
- Downey, Allen B. (2012). *Think Python* (<http://shop.oreilly.com/product/0636920025696.do>) (em inglês). Sebastopol (Califórnia): O'Reilly. 300 páginas. ISBN 978-1-4493-3072-9
- Lutz, Mark (2013). *Learning Python* (<http://shop.oreilly.com/product/0636920028154.do>) (em inglês) 5 ed. Sebastopol (Califórnia): O'Reilly. 1600 páginas. ISBN 978-1-4493-5573-9
- Lutz, Mark (2010). *Programming Python* (<http://shop.oreilly.com/product/9780596158118.do>) (em inglês) 4 ed. Sebastopol (Califórnia): O'Reilly. 1632 páginas. ISBN 978-0-596-15810-1
- David Beazley e Brian K. Jones (2013). *Python Cookbook* (<http://shop.oreilly.com/product/0636920027072.do>) (em inglês) 3 ed. Sebastopol (Califórnia): O'Reilly. 706 páginas. ISBN 978-1-4493-4037-7

Ligações externas

- [Sítio oficial](https://www.python.org/) (<https://www.python.org/>) (em inglês)
- [Python](https://github.com/python/cpython) (<https://github.com/python/cpython>) no GitHub
- «Wiki da comunidade brasileira de usuários» (<https://www.python.org.br/>)
- «Site da comunidade portuguesa de usuários» (<https://python.pt/>)

- [Python \(https://dmoztools.net/Computers/Programming/Languages/Python\)](https://dmoztools.net/Computers/Programming/Languages/Python) no [DMOZ](#)
-

Obtida de "<https://pt.wikipedia.org/w/index.php?title=Python&oldid=65454645>"