Python - Guia do Programador

Peter Jandl Junior

2021-06-22

Contents

Prefácio

Este é um livro experimental, organizado com o propósito de incentivar o uso da linguagem de programação Python e, sendo assim, procura cobrir os aspectos mais básicos de sua utilização, bem como alguns conceitos fundamentais e boas práticas da programação. Para tanto, apresenta os elementos básicos da linguagem, ao mesmo tempo que trata dos aspectos básicos da programação, prosseguindo para conceitos e técnicas mais avançados.

Será utilizada a versão 3 do Python, que é substancialmente diferente da versão 2 e, também, considerada mais correta semanticamente falando, além de suportar um conjunto de novas e interessantes características.

Este material foi escrito com **R Markdown** e uso do pacote **bookdown**[1] (Xie, 2021, 2015), portanto emprega o suporte do **Pandoc**, como, por exemplo, para indicar expressões matemáticas, tal como $f(x) = a * x^2 + b * x + c$.

O muitos fragmentos de código e exemplos contidos neste material aparecem como segue. Quando incluída, a saída produzida pelos fragmentos aparecerá precedida por dois caracteres de numeral (`##`).

```
# Uma mensagem de boas vindas
print('Bem vindo ao "Python - Guia do Programador"!')
```

```
## Bem vindo ao "Python - Guia do Programador"!
```

Este material será futuramente armazenado no **GitHub**, no repositório: https://github.com/pjandl/pygp.

[1] Bookdown no GitHub: https://github.com/rstudio/bookdown.

Chapter 1

Introdução

Python é uma linguagem de programação bastante popular, moderna, utilizada tanto no ambiente corporativo, como no meio acadêmico, e que todo programador devia conhecer. Segundo Kopec (Kopec, 2019, pág.1):

> "Python é usada em atividades tão diversas como ciência de dados, produção de filmes, educação em ciência da computação, gerenciamento de tecnologia da informação e muito mais. Realmente não há um ramo da computação que Python não tenha tocado (exceto, talvez, desenvolvimento de *kernel* de sistemas operacionais). Python é amada por sua flexibilidade, sintaxa bela e suscinta, orientação a objetos pura, e uma comunidade movimentada."

Então, é muito necessário falar um pouco sobre ela.

1.1 Breve Histórico

Observando as dificuldades de muitas pessoas em aprender a programar e trabalhar com programação, Guido Van Rossun, que então trabalhava no *Stichting Mathematish Centrum*[1] (Holanda), deu início ao desenvolvimento de uma nova linguagem de programação em 1990. Sua proposta era oferecer um linguagem de *script* simples, fácil de aprender e de programar. O nome **Python** foi inspirado pelo grupo humorístico britânico *Monty Python*, criador do programa de televisão *Monty Python's Flying Circus*.

Em 1995, já no *Corporation for National Research Initiatives* (EUA), Rossun, deu continuidade ao seu projeto de tornar o Python ainda melhor, neste ponto

[1] CWI: http://www.cwi.nl/.

com as opiniões e ajuda de várias outras pessoas contribuíram no desenvolvimento da linguagem e de suas bibliotecas.

Para garantir a evolução contínua da linguagem e, ao mesmo tempo, desvincular o Python da pessoa de Guido Van Rossun, foi criada em 2001 a *Python Software Foundation*[2], uma organização sem fins lucrativos, que detém os direitos de propriedade intelectual do Python, destinada a manter, desenvolver e divulgar a linguagem com base em um modelo de desenvolvimento comunitário, aberto, com participação de membros individuais e corporativos.

Todas as versões do Python são de *código aberto*[3].

1.2 Características

Python é uma linguagem de programação de alto nível, orientada a objetos, interpretada, interativa, de semântica dinâmica, com tipagem forte. Sua sintaxe é bastante compacta e direta, o que possibilita aos seus programas serem mais curtos do que os construídos em C, C++, Java e outras linguagens.

As razões para isso são:

- dispõe de tipos de dados de alto nível que permitem a expressão direta de operações complexas;
- não requer a declaração prévia de variáveis ou argumentos, cujos tipos são inferidos durante a execução do programa;
- a criação de blocos de diretivas nas suas construções requer apenas a indentação simples, sem necessidade de elementos extras, como chaves ou palavras reservadas, para identificação de início e fim de tais blocos.

Foi projetada para ser simples e elegante, ao mesmo que é sofisticada e completa, pois:

- é orientada a objetos, oferece herança simples, herança múltipla e tratamento polimórfico de objetos;
- oferece exceções como mecanismo mais moderno para o tratamento de erros;
- possui coleta automática de lixo, que efetua a reciclagem de memória de objetos descartados, simplificando o desenvolvimento;
- inclui recursos avançados de manipulação de texto, listas e outras estruturas de dados; e
- seus programas podem ser organizados em módulos e pacotes, reusáveis em diferentes programas.

[2] PSF: https://www.python.org/psf/.
[3] Veja uma definição de *código aberto* ou *open source* em http://www.opensource.org/.

Pode ser usada em múltiplas plataformas (Mac OS, Microsoft Windows, distribuições Linux e Unix), o que evidencia sua portabilidade[4]. Além de ser extensível e dotada de uma ampla e versátil biblioteca.

Como característica de seu projeto, a linguagem Python possui um interpretador que pode funcionar em modo interativo. Isto possibilita que programas escritos em Python, ou mesmo trechos de código, possam ser testados antes de serem compilados ou inclusos em outros programas. Com isso, o Python encoraja a programação de maneira simples, sem requisitar código burocrático, o que a torna muito conveniente para criação rápida de programas.

Por tudo isso é muito utilizada para tratamento, processamento de visualização de dados em aplicações de Análise de Dados (*Data Analysis*) e de Ciência de Dados (*Data Science*). Mas tambem é empregada na construção de aplicações complexas e de grande porte em empresas icônicas como DropBox, Google, IBM, Instagram, Nasa, Reddit, Spotify, Uber, YouTube e outras.

1.3 Habilidades da Programação

Programar[5] significa *fazer o programa de, planejar, incluir em programação*, além de ter como sinônimos *planejar, projetar, delinear, designar e coordenar.*

A programação de computadores exige o domínio de cinco habilidades distintas:

1. computação (Capítulo 2),
2. sequenciação (Capítulo 3),
3. repetição (Capítulo 4),
4. decisão (Capítulo 5), e
5. modularização (Capítulo 6).

A *computação* é habilidade necessária para expressar e realizar cálculos, ou seja, a capacidade de combinar valores, variáveis, operadores e funções para obter os resultados desejados. Esta habilidade explora as capacidades dos computadores em realizar cálculos.

A *sequenciação* se refere a habilidade de organizar as instruções de um programa de maneira tal que seja resolvido um problema específico, ou seja, se trata de estabelecer uma sequência adequada de instruções para solução de um problema, ou seja, a determina a *lógica* com a qual resolvemos um problema. A sequenciação determina como os dados de um problema serão processados.

A *repetição* consiste da identificação de uma instrução ou um conjunto de instruções que devem ser executados mais de uma vez, o que inclui determinar o

[4] *Portabilidade* é a capacidade dos programas gerados por uma linguagem de programação de serem executados em diferentes plataformas operacionais, idealmente sem alterações, ou com um mínimo de modificações.

[5] Dicio (Dicionário On-Line de Português): https://www.dicio.com.br/programar.

número de vezes que tais instruções serão executadas ou a condição que exige sua repetição. Além disso, a repetição permite reduzir a quantidade de instruções necessárias para resolver um problema, e confere maior flexibilidade às soluções criadas.

Outra habilidade importante é a *decisão*, pois possibilita escolher quais instruções serão executadas, isto é, permite que, durante a execução das instruções, sejam selecionadas aquelas que serão executadas. Esta decisão é realizada mediante a avaliação de uma condição que o programador estabelece como critério de escolha ou de seleção.

Finalmente temos a *modularização*, que é a habilidade que trata da divisão das instruções necessárias para resolução de um problema em partes menores, cada uma com responsabilidades distintas. Essa estratégia oferece várias vantagens, entre elas, simplificar a compreensão e solução do problema, além de possibilitar o reuso.

{::options parse_block_html="true" /}

Chapter 2

Computação

A palavra *computar*[1] significa *fazer o cômputo de, calcular, orçar*, assim, a *computação* é a habilidade da programação voltada para a realização de cálculos, o que permite explorar uma das capacidades centrais dos computadores.

A realização de cálculos envolve a construção de *expressões*, que podem conter vários elementos:

- valores literais,
- operadores,
- variáveis, e
- funções.

2.1 Valores Literais

Um *valor literal*, ou apenas *literal*, é uma quantidade, um número, uma palavra, um nome ou um texto que podemos ler diretamente numa instrução, isto é, um elemento que não depende da execução da instrução ou de qualquer outra parte do programa para que possa ser compreendido. Alguns exemplos podem facilitar no entendimento do que são os literais.

O número `2021` é um literal, assim como `15`, `3.14`, `-3` ou `4294967296` também são literais numéricos, que são escritos diretamente no texto do programa Python, tal como em todas as linguagens de programação. Vale notar que o *separador* decimal é o caractere ponto (`.`).

Já a inclusão de literais de texto no Python, seja uma palavra, frase ou um caractere individual, requer que sejam dispostos entre aspas simples (`'`) ou aspas

[1] Dicio (Dicionário On-Line de Português): https://www.dicio.com.br/computar.

duplas (`"`). Estes *delimitadores* de texto podem ser usados para indicar palavras, como `'computador'` ou `"programação"`, e caracteres individuais, como `'A'`, `"x"`, `'!'`, `"@"` ou `'+'`. Pequenas frases, que não podem exceder uma linha, são indicadas da mesma maneira, ou seja, com uso destes delimitadores, como no fragmento que segue.

```
'Python é uma linguagem de programação moderna.'
"A área de 'data science' utiliza Python."
'A linguagem Python é "interpretada" e "dinâmica".'
```

Os delimitadores são exigidos para que seja possível distinguir o texto literal fornecidos pelo programador dos demais elementos da linguagem. O delimitador não pode fazer parte do texto delimitado, no entanto, é interessante observar que as aspas simples podem usadas quando delimitadas por aspas duplas e vice-versa.

Também é possível definir texto literal com múltiplas linhas, com o uso triplo de aspas simples ou aspas duplas, o que pode ser conveniente em algumas situações, como por exemplo indicar comandos SQL[2] usados pelo programa. Seguem exemplos diretos.

```
"""Python is used in pursuits as diverse as data science,
film-making, computer science education, IT management,
and much more."""

'''There really is no computing field that Python has not
touched (except maybe kernel development). Python is loved
for its flexibility, beautiful and succinct syntax,
object-oriented purity, and bustling community.

--- Kopec (2019)'''
```

Além de literais numéricos e de texto, o Python também dispõe de dois literais de tipo lógico (ou boleanos), que são `False` e `True`, que respectivamente representam os estados *falso* e *verdadeiro*.

2.2 Tipos de Dados

A linguagem Python é capaz de lidar com vários tipos de dados, ou seja, com categorias distintas de valores, cada uma oferecendo um conjunto próprio de

[2] *Structured Query Language*, linguagem padronizada para consulta e manipulação de dados em *Sistemas Gerenciadores de Bancos de Dados Relacionais* (SGBDR).

possibilidades. Os tipos de dados básicos disponíveis na linguagem são considerados tipos *built-in*[3], ou *nativos*, e estão listados na Tabela 2.1.

Tabela 2.1: Tipos de dados *built-in*

Tipo	Descrição
`int`	Inteiro (ou integral), valor númerico sem parte fracionária.
`float`	Real, valor numérico com parte fracionária em ponto flutuante.
`bool`	Lógico (ou boleano).
`string`	*String* ou cadeia de caracteres.
`complex`	Número complexo, com a parte imaginária identificada pelo sufixo `j`.

O Python dispõe de três tipos de dados numéricos *built-in*: `int`, `float` e `complex`. O tipo `int` possibilita a representação de valores numéricos inteiros, ou seja, números, contagens e quantidades, positivos ou negativos, mas sem uma parte fracionária. A função *built-in* `type()` permite determinar o tipo de quaisquer valores literais (na verdade, de qualquer coisa no Python). Observe o uso de `type()` para o valor inteiro `15`.

```
type(15)
```

```
## <class 'int'>
```

Qualquer valor inteiro, quando avaliado por `type()`, produz como retorno a classe `int`, que representa este tipo de dados.

Analogamente, o tipo `float` possibilita a representação de valores numéricos reais, ou seja, números positivos ou negativos dotados de uma parte fracionária. Como antes, pode ser usada a função *built-in* `type()` para determinar o tipo de literais reais, como segue.

```
type(3.14)
```

```
## <class 'float'>
```

Valores reais avaliados por `type()` produzem como retorno a classe `float`, que representa este tipo de dados.

Diferente da grande maioria das linguagens de programação, Python permite a representação nativa de números complexos, ou seja, valores dotados de uma

[3] O termo *built-in* é utilizado para designar um elemento que faz parte da própria definição da linguagem, ou seja, está disponível em todos os programas, sem necessidade de importação de módulos ou pacotes.

parte real e uma parte imaginária, que utiliza o sufixo `j` para diferenciá-la da parte real. No fragmento que segue, a função `type()` é utilizada para determinar o tipo do valor literal `1.5 - 4.9j`, um número complexo cuja parte real tem valor `1.5` e a parte imaginária vale `4.9j`.

```
type(1.5 - 4.9j)
```

```
## <class 'complex'>
```

A verificação de tipo com `type()` retorna a classe `complex` quando recebe números complexos como argumento.

Outro importante tipo *built-in* é `bool` que representa o tipo lógico ou boleano, que possui apenas dois valores possíveis: `False`, para valores falsos; e `True`, para valores verdadeiros. O uso de `type()` para o literal lógico `True` é mostrado no fragmento que segue.

```
type(True)
```

```
## <class 'bool'>
```

Como esperado, é retornada a classe `bool`.

Finalmente, a representação de texto, sejam caracteres, palavras ou frases, é feita por elementos do tipo `str`, à despeito do delimitador usado (aspas simples, duplas ou triplas). A função `type()` também permite determinar o tipo de literais ou valores de texto, como segue.

```
type('Python: Guia do programador')
```

```
## <class 'str'>
```

Observamos que a classe `str` é retornada quando `type()` verifica o tipo do argumento de texto fornecido.

2.3 Operadores

A utilidade dos computadores se deve, em grande parte, às suas capacidades de realizar cálculos. Então, as linguagens de programação devem suportar essas capacidades e, para isso, deve oferecer operadores que permitam combinar valores e variáveis (seção 2.4) para expressar as sequências de cálculos adequadas à obtenção dos resultados desejados.

Como na matemática, um operador é um símbolo convencionado para representar uma operação específica entre seus operandos, isto é, os valores participantes desta operação. Existem quatro grupos principais de operadores[4], indicados na Tabela 2.2.

Tabela 2.2: Grupos de operadores

Grupo	Descrição
Aritméticos	Destinados às operações algébricas comuns, como adição, subtração e outras.
Relacionais	Possibilitam a comparação entre valores numéricos e não numéricos.
Lógicos	Permitem a combinação de predicados lógicos.
Atribuição	São usados para definir o valor de variáveis e parâmetros de funções.

Com o uso destes operadores, é possível realizar cálculos, comparar valores, avaliar condições e atribuir valores para variáveis, como será tratados nas seções que seguem.

2.3.1 Operadores Aritméticos

Os operadores aritméticos são destinados à realização das operações algébricas de adição, subtração, multiplicação, divisão e potenciação, como relacionado na Tabela 2.3, onde podemos observar que a maior parte dos operadores aritméticos são idênticos aos usados na matemática, exatamente para facilitar sua identificação e emprego.

Tabela 2.3: Operadores aritméticos

Operador	Operação	Aridade[5]
`+`	Adição (soma).	2
`-`	Subtração (diferença).	2
`*`	Multiplicação (produto).	2
`/`	Divisão (quociente).	2
`//`	Divisão inteira (quociente).	2
`%`	Resto da divisão inteira (módulo).	2
`**`	Potenciação (exponenciação).	2
`+`	Sinal positivo.	1
`-`	Sinal negativo.	1

[4] O Python também possui operadores bit-a-bit (ou *bitwise*), que atuam sobre os bits que compõem os valores inteiros, mas que não serão tratados neste material.

[5] Na matemática a *aridade* de uma função ou operação é o número de argumentos ou operandos tomados.

2.3.1.1 Adição

Utilizamos o operador `+` para indicar a adição ou a soma, que requer dois operandos (sua aridade), ou seja, os dois valores que serão adicionados. No fragmento que segue é possível ver que o uso deste operador é simples.

```
123 + 456
```

```
## 579
```

O operador `+` pode ser usado para somar qualquer combinação de valores inteiros e reais, além de obedecer as propriedades *comutativa*[6], *associativa*[7], *distributiva*[8] e do *elemento nêutro*[9] da adição. A soma de valores inteiros produz resultados de tipo `int`, mas se combinados valores inteiros e reais, o resultado será de tipo `float`.

Existe outro uso para o operador `+`, que é como sinal positivo, tal como `+5` ou `+19.12`, mas cujo uso é pouco frequente, pois por padrão, valores sem sinal são considerados positivos.

2.3.1.2 Subtração

O operador `-` permite realizar a subtração ou a diferença entre dois valores, ou seja, requer dois operandos (sua aridade). Seu uso também é simples.

```
654 - 123
```

```
## 531
```

O operador `-` pode efetuar a diferença de qualquer combinação de valores inteiros e reais, além de obedecer as propriedades *distributiva*[10] e do *elemento nêutro*[11] da subtração. A subtração de valores inteiros produz resultados de tipo `int`, mas se combinados valores inteiros e reais, o resultado será de tipo `float`.

[6] Propriedade *comutativa*: a ordem dos operandos não altera o resultado, pois na adição temos que
`A + B = B + A`.

[7] Propriedade *associativa*: a associação dos operandos não modifica o resultado, pois na adição temos que
`A + B + C = (A + B) + C = A + (B + C) = (A + C) + B`.

[8] Propriedade *distributiva*: realizamos o produto do termo externo ao parênteses com seus termos internos, ou seja, na adição `A * (B + C) = A * B + A * C`.

[9] *Elemento nêutro*: valor que não modifica o resultado da operação, na adição ao somar zero não altera o resultado, pois `A + 0 = A`.

[10] Propriedade *distributiva*: realizamos o produto do termo externo ao parênteses com seus termos internos, ou seja, na subtração `A * (B - C) = A * B - A * C`.

[11] *Elemento nêutro*: valor que não modifica o resultado da operação, subtrair zero não altera o resultado, pois `A - 0 = A`.

Como para o operador `+`, existe um segundo uso para o operador `-` como sinal negativo, por exemplo, `-7` ou `+20.06`, e cujo uso é mais comum, para explicitar valores considerados negativos.

2.3.1.3 Multiplicação

O operador `*` permite efetuar a multiplicação ou o produto de dois valores, tomando dois operandos, com uso como segue.

```
537 * 215
```

```
## 115455
```

Este operador pode efetuar o produto de qualquer combinação de valores inteiros e reais, além de obedecer as propriedades *comutativa*[12], *associativa*[13] e do *elemento nêutro*[14] da multiplicação. Como antes, o produto de valores inteiros produz resultados de tipo `int`, mas se multiplicados valores inteiros e reais, o resultado será de tipo `float`.

2.3.1.4 Divisão real, divisão inteira e resto da divisão

O operador `/` realiza a divisão de dois valores, obtendo um quociente a partir de dois operandos, com uso como segue.

```
537 / 215
```

```
## 2.4976744186046513
```

Podem ser combinados valores inteiros e reais com este operador, que também possui um *elemento nêutro*[15]. Deve-se tomar cuidado com a divisão por zero, que provoca o erro `ZeroDivisionError`. Também deve ser destacado que este operador realiza a divisão real dos operandos indicados, produzindo um resultado de tipo `float`, ou seja, que pode conter uma parte fracionária, com uma ou mais casas decimais. Mesmo que o resultado da divisão seja exato e não possua uma parte fracionária, seu tipo será `float`.

[12] Propriedade *comutativa*: a ordem dos operandos não altera o resultado, pois na multiplicação `A * B = B * A`.

[13] Propriedade *associativa*: a associação dos operandos não modifica o resultado, pois na multiplicação `A * B * C = (A * B) * C = A * (B * C) = (A * C) * B`.

[14] *Elemento nêutro*: valor que não modifica o resultado da operação, a multiplicação por um não modifica o resultado, pois `A * 1 = A`.

[15] *Elemento nêutro*: valor que não modifica o resultado da operação, a divisão por um não modifica o resultado, pois `A / 1 = A`.

Se desejado, pode ser utilizado o operador `//`, que realiza a divisão inteira (*floor division*) de seus operandos, descartando a parte fracionária, retornando um resultado sempre do tipo `int`.

```
537 // 215
```

```
## 2
```

Também é possível obter o resto da divisão inteira com o operador `%`, ou seja, a parcela inteira descartada pela divisão inteira. Por exemplo a divisão `6 / 4` produz `1.5`, um valor real; enquanto a divisão inteira `6 // 4` resulta `1`, sendo que o resto desta divisão `6 % 4` permite obter `2`. Este operador também é conhecido como *módulo*.

2.3.1.5 Potenciação

Python oferece um operador para realização da *potenciação* (ou da *exponenciação*) que é `**` (duplo asterisco, sem espaço em branco), usado na forma `base ** expoente`, onde tanto a base, como o expoente, podem ser números inteiros ou reais, como segue:

```
2 ** 10
```

```
## 1024
```

Assim, `2 ** 10` representa dois elevado à décima potência e `10 ** 3` calcula dez elevado ao cubo.

Como na matemática, expoentes negativos representam potências inversas, por exemplo `2 ** -3` equivale à `1 / (2 ** 3)`; e expoentes entre `0` e `1` permitem efetuar a *radiciação* (obter raízes), ou seja, `16 ** (1/2)` e `16 ** 0.5` permitem calcular a raíz quadrada de `16`, enquanto `5 ** (1/3)` e `5 ** 0.3333` calculam a raiz cúbica de `5`.

2.3.2 Operadores Relacionais

Os operadores relacionais permitem comparar valores determinando as existência de relações específicas entre eles, tal como mostra a Tabela 2.4. Vários dos operadores relacionais são compostos por dois caracteres, entre os quais *não pode existir espaços em branco*.

Tabela 2.4: Operadores relacionais

Operador	Relação	Aridade
`>`	Maior que.	2
`>=`	Maior ou igual a.	2
`<`	Menor que.	2
`<=`	Menor ou igual a.	2
`==`	Igual.	2
`!=`	Diferente.	2

Todos os operadores relacionais tomam dois operandos e retornam como resultado um valor do tipo `bool`, ou seja, um resultado que só pode ser `False`, quando a relação indicada não existe (é falsa), ou `True`, quando se confirma a relação indicada (ou seja, é verdadeira). Um exemplo simples do uso destes operadores está no fragmento que segue, no qual se verifica se o valor `1964` *é menor* que `1995` e produz um retorno `True`.

```
1964 < 1995
```

```
## True
```

O próximo fragmento mostra outro uso simples destes operadores, onde se compara `1931` e `2021` em relação a sua igualdade, o que produz um retorno `False`.

```
1931 == 2021
```

```
## False
```

Como será visto na seção 2.5, os operadores relacionais pode ser combinados com operadores aritméticos e lógicos para formar expressões compostas capazes de verificar relações mais complexas.

2.3.3 Operadores Lógicos

Os operadores lógicos, listados da Tabela 2.5, permitem realizar as operações fundamentais da álgebra de Boole que são a conjunção (*e-lógico*), a disjunção (*ou-Lógico*) e a negação (*não-lógico*).

Tabela 2.5: Operadores lógicos

Operador	Operação	Aridade
`and`	E-lógico (conjunção).	2
`or`	Ou-lógico (disjunção).	2

Operador	Operação	Aridade
`not`	Não-lógico (negação).	1
`in`	Está em.	2
`is`	È.	2

A operação de conjunção ou *e-lógico* verifica o estado lógico de seus dois operandos e retorna um resultado verdadeiro (`True`) apenas se ambos os operandos são verdadeiros, como mostra a Tabela 2.6.

Tabela 2.6: Tabela-verdade[16] do *e-lógico* (conjunção)

A	B	A and B
`False`	`False`	`False`
`False`	`True`	`False`
`True`	`False`	`False`
`True`	`True`	`True`

A operação de disjunção ou *ou-lógico* também verifica o estado lógico de seus dois operandos, mas retorna um resultado falso (`False`) apenas se ambos os operandos são falsos, como mostra a Tabela 2.7.

Tabela 2.7: Tabela-verdade do *ou-lógico* (disjunção)

A	B	A or B
`False`	`False`	`False`
`False`	`True`	`True`
`True`	`False`	`True`
`True`	`True`	`True`

Finalmente, a operação de negação ou *não-lógico* retorna o oposto (ou inverso) de seu único operando, ou seja, quando este tem valor `False`, sua negação retorna `True`, e vice-versa, como na Tabela 2.8.

Tabela 2.8: Tabela-verdade do *não-lógico* (negação)

A	not A
`False`	`True`
`True`	`False`

[16] Uma *tabela-verdade* mostra todas as combinações possíveis dos operandos de uma função lógica e seus resultados. O número de combinações possíveis sempre é 2operandos.

Os operadores lógicos `and`, `or` e `not` permitem conectar logicamente o resultado de diferentes expressões aritméticas, relacionais ou lógicas, o que permite construir expressões compostas de várias partes e, portanto, mais complexas, como será visto na seção 2.5.

Os operadores `in` (*teste de membro*) e `is` (*teste de identidade*) serão vistos nas seções que tratam de estruturas de dados e uso de objetos.

2.4 Variáveis

Um programa de computador requer o uso de alguns ou de muitos dados para que possa produzir os resultados desejados. Durante a execução do programa, os dados necessários são armazenados na memória do computador. Para evitar que o programador tenha que lidar com os endereços de memória, isto é, com as posições onde os dados ficam efetivamente armazenados, são utilizadas *variáveis*.

Uma *variável* é um espaço em memória, reservado para guardar um valor, ao qual se associa um *identificador*, ou seja, um nome por meio do qual se define e se recupera o valor armazenado. O uso de variáveis simplifica a programação, pois o programador não precisa se preocupar com os endereços de memória utilizados, nem com o espaço necessário (número de bytes) para armazenar tais valores, tão pouco com a organização dos dados e das instruções do programa.

Por meio do uso das variáveis, o programador pode armazenar valores literais ou o resultado de cálculos diversos, que podem ser utilizados em etapas posteriores do programa, evitando sua repetição e o processamento destes cálculos.

Além disso, o uso de variáveis constitui um importante mecanismo de abstração[17], pois o uso de nomes significativos melhora a legibilidade do programa e permite que suas ações sejam compreendidas mais facilmente.

A criação de variáveis em Python é bastante simples e direta, empregando a sintaxe que segue:

identificador = valor_inicial

O *identificador* é o *nome* que o programador escolhe para uma variável, o símbolo `=` é o operador de atribuição e `valor_inicial` é o valor que será inicialmente armazenado por esta variável. Por exemplo, a criação da variável de nome `a` com valor inicial definido pelo literal `15`:

[17] Segundo o Dicio (Dicionário On-Line de Português) *abstrair* é a ação de analisar isoladamente um aspecto, contido num todo, sem ter em consideração sua relação com a realidade. Fazer a abstração de uma coisa permite simplificar, pois observamos seu aspecto principal, sem levar em conta seus detalhes (https://www.dicio.com.br/).

```
a = 15
```

Esta construção é lida como *variável* `a` *recebe o valor 15* ou, resumidamente, `a` *recebe 15*.

Desta forma, para criar uma nova variável em um programa Python basta atribuir um valor para um novo identificador. A criação de uma variável desta maneira é chamada *inicialização*. A partir de sua inicialização, a variável criada se torna disponível no escopo onde foi declarada.

Para utilizar uma variável, em Python e outras linguagens de programação, basta utilizar seu nome, de maneira que este é automaticamente substituído pelo valor atual (ou corrente) da variável. Ou seja. apenas escrever seu nome.

```
a
```

```
## 15
```

Como esperado, o uso do nome de variável `a` recupera seu valor, no caso `15`.

Observe que o Python não requer que o tipo da variável seja declarado, pois este é inferido conforme o tipo do valor atribuído , assim a variável `a` será do tipo `int`, como mostrado pelo uso da função *built-in* `type()`:

```
type(a)
```

```
## <class 'int'>
```

Cada vez que a variável recebe um valor, o tipo da variável é novamente inferido, de maneira que, se atribuído um valor de tipo diferente do previamente armazenado na variável, seu tipo é *alterado dinamicamente*, sem produzir qualquer tipo de erro. Assim, a variável `a`, do tipo `int`, pode receber um valor real como segue:

```
a = 7.45
```

A alteração do tipo da variável pode ser visto por meio da função `type()`:

```
type(a)
```

```
## <class 'float'>
```

A valor da variável `a` pode ser recuperado com uso de seu nome, permitindo verificar a alteração em seu conteúdo.

```
a
```

```
## 7.45
```

Em conjunto, tudo isto confere grande simplificade e flexibilidade ao Python em relação a criação e utilização de variáveis.

2.4.1 Denominação de Variáveis

Os nomes de variáveis em Python podem ser compostos de uma ou mais letras, números e também símbolos _ (sublinhado ou *underscore*), desde que iniciados por uma letra ou sublinhado. É recomendado que usem apenas letras minúsculas e, caso sejam compostos de mais de uma palavra, estas sejam separadas por um sublinhado. Esta convenção é conhecida como *snake case*.

São exemplo válidos: `x`, `s3`, `total`, `quadra03`, `posicao_absoluta`, `_media_parcial`.

Desde que seguida esta regra de formação, os nomes podem quaisquer, exceto das *palavras reservadas* da linguagem (seção 2.4.2), e arbitrariamente longos, assim sugere-se o uso de denominações representativas do propósito das variáveis, melhorando a legibilidade dos programas. Caracteres acentuados podem ser usados, embora desaconselhado. Em hipótese alguma os nomes podem conter espaços em branco, tabulações ou quaisquer operadores.

2.4.2 Palavras Reservadas

O Python possui um conjunto de *palavras reservadas* que tem significado pré-definido, pois indicam as diretivas da linguagem e outros elementos de sua sintaxe. As *palavras reservadas*, ou as *keywords*, listadas na Tabela 2.6 não podem ser utilizadas como identificadores ou para qualquer outro fim, exceto o determinado pela linguagem.

Tabela 2.6: Palavras reservadas (*keywords*)

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

A maioria das palavras reservadas do Python é comum à outras linguagens de programação. Por exemplo, dentre as 35 *keywords*, 12 são comuns ao Java e ao C#.

2.5 Expressões

Uma *expressão* é uma combinação de valores literais (seção 2.1), variáveis (seção 2.4) e operadores (seção 2.3), que produz um resultado como consequência do encadeamento dos elementos nela indicados. A determinação do valor resultante de uma expressão é que se denomina *avaliação da expressão*.

O Python avalia as expressões da *esquerda para direita*, ou seja, no sentido usual de leitura, por exemplo, considere as variáveis `x` e `y` inicializadas como seguem;

```
x = 2
y = 3
```

Uma expressão simples pode combinar valores literais e variáveis, como segue:

```
1 + x + y + 4
```

```
## 10
```

O resultado, `10`, é obtido da soma dos valores expressos diretamente pelos literais e recuperados das variáveis indicadas, ou seja, `1 + 2 + 3 + 4`.

Mas as expressões podem combinar e utilizar operadores diferentes, como:

```
2 * x + y / 4
```

```
## 4.75
```

Aqui, o resultado `4.75` mostra que, quando operadores diferentes são misturados, a ordem de avaliação esquerda-para-direita é modificada. Isto ocorre porque alguns operadores possuem maior *prioridade* (ou *precedência*).

A *prioridade* dos operadores, ou sua *precedência*, é o critério matemático que estabele a ordem com que os operadores serão executados em uma expressão, além de como os operadores envolvidos serão tomados (sua *associatividade*).

Para toda e qualquer expressão, sempre são aplicadas as regras de precedência da linguagem., garantindo que a expressão produza sempre o mesmo resultado, independentemente da plataforma ou do computadore utilizado.

2.5.1 Prioridade dos operadores

Como uma expressão pode combinar operadores diferentes e com aridade distinta, é necessário estabelecer um critério para determinar qual a ordem de execução dos operadores, garantindo resultados consistentes na avaliação da expressões, seja qual for a combinação empregada.

A Tabela 2.9 relaciona a prioridade dos operadores em Python, da maior (nível 1) para a menor (nível 18). Várias das indicações tratam de construções que serão vistas nos próximos capítulos deste material.

Tabela 2.9: Prioridade (precedência) dos operadores em Python

Nível	Operadores	Descrição
1	(*expr*), [*expr*,...],{*ch*:*val*}, {*expr*,...}	Expressões parentisadas, listas, dicionários e conjuntos
2	[*idx*], [*ini*:*fim*],x(*args*), `.`	Subscrição, fatiamento,passagem de argumentos, seleção
3	`await x`	Expressão `await`
4	`**`	Potenciação
5	`+x`, `-x`, `~`	Positivo, negativo, não *bitwise*
6	`*`, `/`, `//`, `%`	Multiplicação, divisão, divisão inteira, módulo
7	`+`, `-`	Adição, subtração
8	`<<`, `>>`	Deslocamento à esquerda e à direita
9	`&`	E *bitwise*
10	`^`	Ou-exclusivo *bitwise*
11	`|`	Ou *bitwise*
12	`<`, `<=`, `>`, `>=`, `==`, `!=`,`in`, `not in`, `is`, `not is`	Relacionais, teste de membro e teste de identidade
13	`not x`	Não lógico
14	`and`	E lógico
15	`or`	Ou lógico
16	`if - else`	Expressão condicional
17	`lambda`	Expressão lambda
18	`=`	Atribuição

Os operadores existentes num mesmo nível possuem a mesma precedência, sendo avaliados conforme encontrados da esquerda para a direita.

Os parênteses são operadores especiais, que podem ser utilizados para alterar a precedência pré-estabelecida de avaliação dos operadores, permitindo determinar uma sequência específica para o cálculo de uma expressão. Sempre é avaliado o conteúdo dos parênteses mais internos, prosseguindo com o conteúdo dos mais externos, até que a expressão seja completamente avaliada. Dentro de cada parênteses, a prioridade dos operadores é aplicada normalmente. A

expressão `9 * 5 + 2` equivale à `(9 * 5) + 2`, pois os parênteses não alteram a prioridade das operações, mas que é diferente de `9 * (5 + 2)`, cujo parânteses *força* que a soma `5 + 2` ocorra *antes* da multiplicação.

$9 * 5 + 2 = (9 * 5) + 2 = 47$

Mas a próxima expressão só poderá ser avaliada corretamente se estiverem presentes os parêntesis indicados, caso contrário será obtido o resultado das expressões anteriores:

$9 * (5 + 2) = 9 * 7 = 63$

O uso de parênteses, mesmo quando indicando a ordem natural de precedência, permite construir expressões cuja leitura é mais fácil, além de evitar alguns erros, constituindo assim uma boa prática de programação.

2.6 Funções

Blá blá.

Chapter 3

Sequenciação

A *sequenciação* é a habilidade requerida para organizarmos uma sequência de instruções que permita resolver um problema específico.

> Um algoritmo é uma sequência organizada e finita de instruções que permite a solução de um problema específico ou de uma classe de problemas.

Chapter 4

Repetição

Mais uma habilidade...

4.1 Repetição Condicional

Dá-lhe *while*!

4.2 Repetição Automática

Dá-lhe *for*!

Chapter 5

Decisão

5.1 Decisão Simples

Aqui tratamos do *if*!

5.2 Decisão Completa

Aqui tratamos do *if/else*!

5.3 Decisões Encadeadas

Aqui tratamos do *if/elif/else*!

Chapter 6

Modularização

6.1 Programa Principal

6.2 Funções

6.2.1 Tipos de funções

6.2.2 Retorno de Valor

6.2.3 Passagem de Parâmetros

6.2.4 Parâmetros *Default*

6.2.5 Parâmetros Variáveis

6.3 Importação

6.4 Criação de pacotes e módulos

Como incluir as referências bibliográficas aqui?

Bibliography

Kopec, D. (2019). *Classic Computer Science Problems in Python*. Manning Publications Co., Shelter Island, New York, 1st edition. ISBN 978-1617295980.

Xie, Y. (2015). *Dynamic Documents with R and knitr*. Chapman and Hall/CRC, Boca Raton, Florida, 2nd edition. ISBN 978-1498716963.

Xie, Y. (2021). *bookdown: Authoring Books and Technical Documents with R Markdown*. R package version 0.22.