Fundamentos de programação e Python

Lucas Araújo Campos Szuster

21 de dezembro de 2025

Sumário [parte 1]

Sumário [parte 2]

O que são variáveis? [parte 1]

As variáveis são elementos fundamentais da programação. Elas funcionam como "caixinhas"utilizadas para armazenar informações na memória do computador, permitindo que os programas manipulem dados durante sua execução. Cada variável é caracterizada por um conjunto de atributos essenciais, que definem como ela é identificada, utilizada e armazenada pelo sistema:

- **Nome**: identifica a variável no programa e permite que ela seja referenciada pelo programador.
- **Tipo**: define o tipo de dado que a variável pode armazenar, como números inteiros, reais, caracteres ou valores lógicos.
- **Valor**: corresponde ao dado efetivamente armazenado na variável em um determinado momento da execução do programa.
- **Endereço de memória**: indica a posição da memória do computador onde o valor da variável está armazenado.

O que são variáveis [parte 2]

Dos diferentes atributos que uma variável pode possuir, apenas três são relevantes para a programação em Python: o **nome**, o **tipo** e o **valor**. Esses atributos permitem que o programador identifique a variável, compreenda o tipo de dado que ela pode armazenar e acompanhe o valor associado a ela durante a execução do programa, sendo suficientes para a grande maioria das aplicações.

O **endereço de memória** é um conceito mais comum em linguagens de baixo nível, como *C*, *C++* e *Assembly*, nas quais o programador precisa lidar diretamente com a alocação e o acesso à memória. Python, por ser uma linguagem de alto nível, abstrai esses detalhes por meio de um gerenciamento automático de memória realizado pelo interpretador, tornando o acesso direto a endereços de memória desnecessário e permitindo que o programador concentre seus esforços na lógica e na resolução do problema.

Como Variáveis Funcionam

Armazenamento na Memória

Quando um programa atribui um valor a uma variável:

1. O computador reserva um espaço de memória
2. Esse espaço é identificado por um endereço único
3. O valor é guardado nesse endereço

Sempre que uma variável é acessada por meio de seu nome, o interpretador do Python utiliza esse identificador para localizar o endereço de memória ao qual ela está associada e, a partir disso, recuperar o valor armazenado correspondente.

Variáveis simples

int

Uma variável do tipo `int` representa um número inteiro, ou seja, um valor numérico sem casas decimais. Diferentemente de muitas linguagens de programação, em que o tipo inteiro possui um intervalo fixo — como, por exemplo, de -2147483648 a 2147483647 em arquiteturas de 32 bits —, o Python não impõe um limite rígido para os valores inteiros.

Em Python, o tipo `int` possui precisão arbitrária, o que significa que os valores podem crescer conforme a necessidade do programa, sendo limitados apenas pela quantidade de memória disponível no computador. Essa característica simplifica o trabalho do programador, pois elimina preocupações com estouro de inteiros (*overflow*) comuns em linguagens de mais baixo nível.

float

Uma variável do tipo `float` representa um número real, ou seja, um valor numérico que **pode** conter casas decimais. Embora um `float` não seja obrigado a possuir uma parte decimal — podendo, inclusive, representar valores inteiros —, esse tipo é utilizado justamente quando se deseja trabalhar com números que envolvem frações ou resultados aproximados.

A precisão da parte decimal é um aspecto fundamental do tipo `float`. Devido à forma como números reais são representados internamente pelos computadores, utilizando o padrão de ponto flutuante, nem todos os valores decimais podem ser representados com exatidão. Em Python, um `float` geralmente oferece cerca de 15 a 16 casas decimais de precisão, embora esse valor possa variar de acordo com o número armazenado. Essa limitação pode resultar em pequenas imprecisões em cálculos numéricos, especialmente em operações envolvendo muitos cálculos sucessivos.

bool

Uma variável do tipo `bool` representa um valor *booleano*, isto é, um tipo de dado lógico que admite apenas dois estados possíveis: verdadeiro (`True`) e falso (`False`). Esses valores são utilizados para representar condições, decisões e resultados de comparações dentro de um programa.

Em Python, variáveis booleanas são amplamente empregadas em estruturas de controle, como comandos condicionais (`if`, `elif`, `else`) e laços de repetição (`while`), permitindo que o fluxo de execução do programa dependa de expressões lógicas. Normalmente, valores do tipo `bool` são gerados a partir de comparações entre dados, como igualdade, diferença ou relações de ordem, desempenhando um papel fundamental na lógica dos programas.

Exercício 1

Complete a tabela:

Valor da variável	Tipo Da variável
15	
1.34	
12	
True	
5.89	
π	
False	
5	
7	
8	

Solução

Aqui está a solução do problema:

Valor da variável	Tipo Da variável
15	`int`
1.34	`float`
12	`int`
True	`bool`
5.89	`float`
π	`float`
False	`bool`
5	`int`
7	`int`
8	`int`

Exercício 2

Complete a tabela:

Dado armazenado	Tipo Da variável
idade de uma pessoa altura exata de um edifício quantidade de pregos em uma obra o número irracional *e* o valor em reais de um pão um indicativo se um carro está lavado a população global o comprimento de um lote a distância entre dois países o número 223	

Solução

Aqui está a solução do problema:

Dado armazenado	Tipo Da variável
idade de uma pessoa	`int`
altura exata de um edifício	`float`
quantidade de pregos em uma obra	`int`
o número irracional e	`float`
o valor em reais de um pão	`float`
um indicativo se um carro está lavado	`bool`
a população global	`int`
o comprimento de um lote	`float`
a distância entre dois países	`float`
o número 223	`int`

Variáveis de conjunto

string

Uma `string` — ou, como é denominada em Python, `str` — é um tipo de variável que representa uma sequência de caracteres, como letras, números, símbolos e espaços. Strings são amplamente utilizadas para armazenar e manipular textos, como nomes, mensagens, palavras ou até mesmo frases completas.

Em Python, uma `string` pode ser entendida como uma estrutura semelhante a uma lista de caracteres, na qual cada posição armazena um único caractere e pode ser acessada individualmente por meio de índices. Essa característica permite realizar diversas operações, como percorrer caracteres, extrair partes do texto (*slicing*) e aplicar transformações, tornando as strings um dos tipos de dados mais importantes e versáteis da linguagem.

list

Uma `list` é uma das estruturas de dados fundamentais da linguagem Python, sendo um tipo nativo que permite armazenar múltiplos valores em uma única variável. Esses valores são organizados de forma sequencial e podem ser acessados individualmente por meio de índices, que indicam a posição de cada elemento dentro da lista.

Observação importante

Em diversas linguagens de programação, incluindo Python, as listas são indexadas a partir do zero. Isso significa que o primeiro elemento de uma lista está na posição 0, o segundo elemento na posição 1, e assim por diante. Consequentemente, para acessar o último elemento de uma lista com n elementos, deve-se utilizar o índice $n - 1$.

dict

Um `dict` (dicionário) é um tipo de variável em Python que permite armazenar dados em pares de **chave** e **valor**. Cada chave está associada a um valor específico, de forma semelhante a como uma palavra em um dicionário tradicional está ligada à sua definição.

Com um `dict`, é possível acessar rapidamente um valor conhecendo sua chave, facilitando o armazenamento e a recuperação de informações que têm algum tipo de relação entre si, como cadastros, tabelas ou configurações de programas.

Exercício 3

Complete a tabela:

Dado armazenado	Tipo Da variável
uma frase motivacional produtos de uma loja e seus valores nomes de alunos em uma sala de aula times de um campeonato preço do feijão em diferentes lojas respostas automáticas para um *bot* populações de cada país do mundo nomes dos edifícios de BH nome de uma empresa os números de uma sequência específica	

Solução

Aqui está a solução do problema:

Dado armazenado	Tipo Da variável
uma frase motivacional	`string`
produtos de uma loja e seus valores	`dict`
nomes de alunos em uma sala de aula	`list`
times de um campeonato	`list`
preço do feijão em diferentes lojas	`dict`
respostas automáticas para um *bot*	`list`
populações de cada país do mundo	`dict`
nomes dos edifícios de BH	`list`
nome de uma empresa	`string`
os números de uma sequência específica	`list`

Variáveis em Python

Declaração de variáveis

Para utilizar os diferentes tipos de variáveis que aprendemos em Python, é importante compreender o que significa declarar uma variável. Declarar uma variável consiste em criar um nome que será utilizado para armazenar um valor na memória do computador. Em Python, a declaração é feita de forma simples e direta: basta atribuir um valor a um nome utilizando o operador de atribuição =, e a variável passa a existir automaticamente.

A seguir, um exemplo de declaração de uma variável chamada "`numero`", do tipo `int`, inicializada com o valor 10:

```
numero = 10
```

Regras gerais para declaração de variáveis

Quando estamos criando variáveis, é importante seguir algumas regras:

- **Nomes sem espaço**: O nome de uma variável deve ser uma única sequência de caracteres sem nenhum espaço, pois senão a variável não é reconhecida.
- **Especificidades de certos tipos de variáveis**: Certos tipos de variáveis devem ser declarados de uma maneira específica (todos serão especificados posteriormente), e isto deve ser respeitado para que o interpretador de Python entenda exatamente que tipo de variável você deseja declarar.

Declaração de variáveis do tipo int

Para declarar uma variável do tipo `int`, basta associar um nome a um valor, como feito anteriormente:

```
numero = 10
```

Entretanto, há uma outra opção de declaração que deixa explícito que a variável é do tipo `int`, a seguinte:

```
numero: int = 10
```

Declaração de variáveis do tipo float

Para declarar uma variável do tipo `float`, basta associar um nome a um valor, como o exemplo abaixo:

```
pi = 3.14
```

Entretanto, há uma outra opção de declaração que deixa explícito que a variável é do tipo `float`, a seguinte:

```
pi: float = 3.14
```

Declaração de variáveis do tipo bool

Para declarar uma variável do tipo `bool`, basta associar um nome a um valor, como o exemplo abaixo:

```
booleano = True
```

Entretanto, há uma outra opção de declaração que deixa explícito que a variável é do tipo `bool`, a seguinte:

```
booleano: bool = True
```

Declaração de variáveis do tipo string

Para declarar uma variável do tipo `string`, basta associar um nome a um valor, entre aspas duplas ou simples, como o exemplo abaixo:

```
texto_um = 'texto'
texto_dois = "texto"
```

Entretanto, há uma outra opção de declaração que deixa explícito que a variável é do tipo `string`, a seguinte:

```
texto_um: str = 'texto'
texto_dois: str = "texto"
```

Declaração de variáveis do tipo list

Para declarar uma variável do tipo `list`, basta associar um nome a um conjunto de valores entre colchetes, separados por vírgulas, conforme ilustrado no exemplo abaixo:

```
lista = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
```

Entretanto, há uma outra opção de declaração que deixa explícito que a variável é do tipo `list`, a seguinte:

```
lista: list[int] = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
```

Observe que é utilizado `list[int]` porque a lista declarada contém somente variáveis do tipo `int`, e isto pode variar de diversas maneiras.

Declaração de variáveis do tipo dict

Para declarar uma variável do tipo `dict`, basta associar um nome a um conjunto de pares chave–valor entre chaves (literais, "{}"), sendo cada par separado por vírgulas, conforme ilustrado no exemplo abaixo:

```
dicionario = {'chave_um' : 1, 'chave_dois' : 2}
```

Entretanto, há uma outra opção de declaração que deixa explícito que a variável é do tipo `dict`, a seguinte:

```
dicionario: dict = {'chave_um' : 1, 'chave_dois' : 2}
```

Exercício 4

Imagine que você é um professor da UFMG e o semestre letivo acabou de terminar. Você deseja criar um programa em Python que permita organizar as notas médias de cada turma, possibilitando consultas posteriores e a realização de diferentes cálculos estatísticos.

1. Reflita sobre qual tipo de variável é mais adequada para armazenar, de forma organizada, os nomes das turmas associadas as suas respectivas notas médias.
2. Declare essa variável em Python, considerando as seguintes turmas e suas notas médias:
    - PNS0 — 68.9
    - POLP — 72.3
    - PFL4 — 71.4
    - LKH5 — 75.7
    - TMNP — 65.8
    - TOL1 — 77.0

Solução

A variável mais adequada para esta situação é um `dict`, pois permite associar diretamente as notas médias de cada turma às suas respectivas turmas. A seguir, apresenta-se a declaração desse `dict`:

```
dicionario_notas: dict = {
        'PNSO' : 68.9,
        'POLP' : 72.3,
        'PFL4' : 71.4,
        'LKH5' : 75.7,
        'TMNP' : 65.8,
        'TOL1' : 77.0
}
```

Operadores

O que são operadores?

Operadores são elementos fundamentais das linguagens de programação que permitem manipular, comparar e transformar os valores armazenados em variáveis. Por meio deles, é possível realizar cálculos matemáticos, efetuar comparações lógicas, combinar informações e modificar dados de forma controlada e precisa. Em outras palavras, os operadores definem como as variáveis interagem entre si e com valores constantes, sendo essenciais para a construção de expressões, tomada de decisões e implementação da lógica de um programa.

Operadores aritméticos

Existem 7 operadores aritméticos em Python:

- +: Adição;
- -: Subtração;
- %: Módulo (resto da divisão);
- *: Multiplicação;
- **: Exponenciação;
- /: Divisão;
- //: Divisão inteira.

Estes operadores possuem funções diferentes com base nos tipos de variáveis específicos que são utilizados.

Operadores aritméticos - exemplo

Aqui segue um exemplo simples com operações envolvendo números inteiros:

```
# declaração de ints
a: int = 10
b: int = 5

# soma de ints
c: int = a + b

# multiplicação de ints
d: int = a * b

# subtração de ints
e: int = a - b
```

Operadores de comparação

Existem 6 operadores de comparação diferentes em Python:

- ==: Igual a;
- !=: Diferente de;
- >: Maior que;
- <: Menor que;
- >=: Maior ou igual a;
- <=: Menor ou igual a.

Resultado

Todos retornam valores booleanos: `True` ou `False`, indicando se a comparação realizada é verdadeira ou falsa.

Operadores de comparação - exemplo

Aqui segue um exemplo simples mostrando como as comparações são realizadas em Python:

```
# declaração de ints
a: int = 10
b: int = 5

# comparação de ints
c: bool = a > b

# comparação de ints
d: bool = a < b

# comparação de ints
e: bool = a == b
```

Operadores lógicos

Existem 3 operadores lógicos diferentes:

- `and`: E lógico;
- `or`: OU lógico;
- `not`: NÃO lógico.

Operadores lógicos - and [parte 1]

O operador lógico `and` que avalia se duas variáveis ou expressões "conectadas" são ambas verdadeiras (`True`), e retorna `True` neste caso e `False` caso contrário.

Imagine que temos duas variáveis do tipo `bool` *a* e *b*, e escrevemos a seguinte linha de código:

```
c: bool = a and b
```

Podemos montar uma tabela que indique os possíveis valores de *c*.

Operadores lógicos - and [parte 2]

Aqui está a chamada "tabela verdade" para os possíveis valores de c:

a	b	a `and` b
`False`	`False`	`False`
`False`	`True`	`False`
`True`	`False`	`False`
`True`	`True`	`True`

Operadores lógicos - or [parte 1]

O operador lógico `or` que avalia se ao menos 1 de duas variáveis ou expressões "conectadas" é verdadeira, retornando `True` nestes casos e `False` no caso de nenhuma ser verdadeira.

Imagine que temos duas variáveis do tipo `bool` *a* e *b*, e escrevemos a seguinte linha de código:

```
c: bool = a or b
```

Podemos montar uma tabela que indique os possíveis valores de *c*.

Operadores lógicos - or [parte 2]

Aqui está a chamada "tabela verdade" para os possíveis valores de c::

a	b	a `or` b
`False`	`False`	`False`
`False`	`True`	`True`
`True`	`False`	`True`
`True`	`True`	`True`

Operadores lógicos - not [parte 1]

O operador lógico `not` simplesmente inverte o valor booleano de uma variável ou expressão.

Imagine que temos uma variável do tipo `bool` *a*, e escrevemos a seguinte linha de código:

```
c: bool = not a
```

Podemos montar uma tabela que indique os possíveis valores de *c*.

Operadores lógicos - not [parte 2]

Aqui está a chamada "tabela verdade" para os possíveis valores de c:

a	`not` a
`False`	`True`
`True`	`False`

Exercício 5

Escreva o valor da variável `r` nas seguintes expressões:

(a) `r: bool = False or False`

(b) `r: bool = True or True`

(c) `r: bool = True and True`

(d) `r: bool = False and True`

(e) `r: bool = not(True and (False or False))`

Observação importante

Os parênteses em Python indicam, assim como na matemática, a ordem de precedência das operações.

Por exemplo, na expressão `a + (b * c)`, temos que `(b * c)` é avaliado e seu resultado é somado à `a`.

Solução (a)

Para definir o valor de `r` na expressão:

```
r: bool = False or False
```

Basta checar na tabela verdade para o operador `or`:

a	b	a or b
False	False	False
False	True	True
True	False	True
True	True	True

Como temos `False or False`, o resultado será `False`.

Solução (b)

Para definir o valor de `r` na expressão:

```
r: bool = True or True
```

Basta checar na tabela verdade para o operador `or`:

a	*b*	*a* or *b*
False	False	False
False	True	True
True	False	True
True	True	True

Como temos `True or True`, o resultado será `True`.

Solução (c)

Para definir o valor de `r` na expressão:

```
r: bool = True and True
```

Basta checar na tabela verdade para o operador `and`:

a	b	a and b
False	False	False
False	True	False
True	False	False
True	True	True

Como temos `True and True`, o resultado será `True`.

Solução (d)

Para definir o valor de `r` na expressão:

```
r: bool = False and True
```

Basta checar na tabela verdade para o operador `and`:

a	b	a and b
False	False	False
False	True	False
True	False	False
True	True	True

Como temos `False and True`, o resultado será `False`.

Solução (e) [Parte 1]

Vamos determinar o valor de `r` na seguinte expressão:

```
r: bool = not(True and (False or False))
```

Para isso, é útil decompor a expressão seguindo a sua ordem de avaliação:

1. Avaliamos primeiro os parênteses internos: `(False or False)`. Denotemos o resultado desta operação por α.
2. Em seguida, avaliamos os parênteses externos: `(True and` α`)`. Chamaremos o resultado desta expressão de β.
3. Por fim, aplicamos o operador `not` sobre β para obter o valor final de `r`.

Solução (e) [Parte 2]

Para determinar o valor de α na expressão:

```
alpha: bool = False or False
```

Basta checar na tabela verdade para o operador `or`:

a	b	a `or` b
`False`	`False`	`False`
`False`	`True`	`True`
`True`	`False`	`True`
`True`	`True`	`True`

Como temos `False or False`, o resultado de α será `False`.

Solução (e) [Parte 3]

Para determinar o valor de β na seguinte expressão, sabendo que α é igual a `False`:

```
beta: bool = True and alpha
```

Basta checar na tabela verdade para o operador `and`:

a	b	a and b
False	False	False
False	True	False
True	False	False
True	True	True

Como temos `True and False`, o resultado de β será `False`.

Solução (e) [Parte 4]

Por fim, para determinar o valor de `r` na seguinte expressão, sabendo que β é igual a `False`:

```
r: bool = not(beta)
```

Basta checar na tabela verdade para o operador `not`:

a	`not` *a*
`False`	`True`
`True`	`False`

Como temos `not False`, o resultado de `r` será `True`.

Operadores de atribuição

Existem 5 operadores de atribuição diferentes em Python:

- =: Atribuição simples;
- +=: Soma e atribuição;
- -=: Subtração e atribuição;
- *=: Multiplicação e atribuição;
- /=: Divisão e atribuição.

O princípio de funcionamento básico de todos esses operadores (exceto o operador =) é o mesmo. Quando uma expressão como `a += 5` é executada em Python, a variável a tem seu valor incrementado em 5, sendo diretamente equivalente à expressão `a = a + 5`. Já o operador de atribuição simples = apenas define o valor da variável como um valor específico, independentemente do que estava armazenado anteriormente.

Operadores de atribuição - exemplo

Aqui segue um exemplo simples mostrando como atribuições são realizadas em Python:

```
# declaração de ints
a: int = 10
b: int = 5

# incremento por 5
a += 5

# decremento por 5
b -= 5

# atribuição com multiplicação por 2
a *= 2
```

Exercício 6

Determine o valor de a ao fim do código:

```
# declaração inicial
a: float = 0

# operações
a += 5
a *= 2
a -= 3
a /= 7
```

Solução [parte 1]

Para descobrir o valor final de a, precisamos analisar o código linha por linha. A segunda operação de atribuição é:

```
a += 5
```

Essa linha é equivalente a:

```
a = a + 5
```

Sabemos que, neste ponto, a possui valor 0. Substituindo no código, temos:

```
a = 0 + 5
```

Portanto, após essa atribuição, o valor de a é 5.

Solução [parte 2]

Continuando, a terceira atribuição realizada é:

```
a *= 2
```

Essa linha é equivalente a:

```
a = a * 2
```

Sabemos que, neste ponto, a possui valor 5. Substituindo no código, temos:

```
a = 5 * 2
```

Portanto, após essa atribuição, o valor de a é 10.

Solução [parte 3]

Continuando, a quarta atribuição realizada é:

```
a -= 3
```

Essa linha é equivalente a:

```
a = a - 3
```

Sabemos que, neste ponto, a possui valor 10. Substituindo no código, temos:

```
a = 10 - 3
```

Portanto, após essa atribuição, o valor de a é 7.

Solução [parte 4]

Por fim, a última atribuição realizada é:

```
a /= 7
```

Essa linha é equivalente a:

```
a = a / 7
```

Sabemos que, neste ponto, a possui valor 7. Substituindo no código, temos:

```
a = 7 / 7
```

Portanto, após essa atribuição, o valor de a é 1.

Acesso a coleções

Para acessar um elemento específico de uma coleção de dados, como um caractere de uma `string` ou um item de uma `list`, utiliza-se o operador de acesso de Python. Esse operador consiste em colchetes (`[]`), que devem ser escritos imediatamente após o nome da variável.

Dentro dos colchetes, informa-se o índice ou a chave do elemento que se deseja acessar, de forma compatível com o tipo da variável. Por exemplo, em sequências ordenadas, como `strings` e `lists`, o acesso é feito por meio de índices numéricos; já em estruturas como `dict`, utiliza-se a chave associada ao valor desejado.

Importante

Para uma `string` ou `list` com n elementos, os índices se iniciam em 0 e terminam em $n - 1$!

Acesso a coleções - strings [parte 1]

Imagine que temos a seguinte `string` em nosso código:

```
string_teste: str = 'Um texto qualquer'
```

Para acessar a letra "x"na palavra texto, basta escrever a seguinte linha de código com o operador de acesso à variáveis de conjunto:

```
string_teste[5]
```

Note que o número utilizado é o 5 pois esta é a indexação completa da `string`:

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
U	m		t	e	x	t	o		q	u	a	l	q	u	e	r

Acesso a coleções - strings [parte 2]

Algo de extrema importância a ser destacado é que, no caso das `strings` (e veremos posteriormente o motivo disso), o operador de acesso permite apenas a leitura dos valores armazenados, não sendo possível modificar diretamente um caractere em uma posição específica.

Dessa forma, o código apresentado a seguir não é válido em Python:

```
# declaração da string
texto: str = 'Olá, mundo!'

# operação inválida
texto[2] = 'x'
```

Acesso a coleções - lists [parte 1]

Imagine que temos a seguinte `list` em nosso código:

```
lista_nums: list[int] = [34, 59, 23, 45, 67, 12, 45, 33]
```

Para acessar o número 23 na lista, basta escrever a seguinte linha de código com o operador de acesso à variáveis de conjunto:

```
lista_nums[3]
```

Note que o número utilizado é o 3 pois esta é a indexação completa da `list`:

0	1	2	3	4	5	6	7
34	59	23	45	67	12	45	33

Acesso a coleções - lists [parte 2]

Algo de extrema importância a ser destacado é que, no caso das `lists`, diferentemente das `strings`, é possível alterar o valor de uma posição específica utilizando o operador de acesso.

Dessa forma, o código apresentado a seguir é válido em Python:

```
# declaração da lista
lista_nums: list[int] = [21, 22, 23, 24, 25]

# operação válida
lista_nums[2] = 99
```

Acesso a coleções - dicts [parte 1]

Imagine que temos a seguinte `dict` em nosso código:

```
dicionario: dict = {'chave_um' : 1, 'chave_dois' : 2}
```

Para acessar o número 2 no `dict`, basta escrever a seguinte linha de código com o operador de acesso à variáveis de conjunto:

```
dicionario['chave_um']
```

Note que não utilizamos um índice, mas sim uma chave que está associada exatamente ao valor 2.

Acesso a coleções - dicts [parte 2]

Algo de extrema importância a ser destacado é que, no caso dos `dicts`, diferentemente das `strings` e de maneira semelhante às variáveis do tipo `list`, é possível alterar o valor de uma posição específica utilizando o operador de acesso.

Dessa forma, o código apresentado a seguir é válido em Python:

```
# declaração do dict
dicionario: dict = {
    'a' : 1,
    'b' : 2,
    'c' : 3
}

# operação válida
dicionario['a'] = 90
```

Acesso a coleções - dicts [parte 3]

Além disso, no caso dos `dicts`, além de ser possível alterar um elemento já existente, também é possível utilizar o operador de acesso para inserir um novo elemento, acessando uma chave que ainda não existe e associando-a a um valor recém-definido.

Dessa forma, o código apresentado a seguir é válido em Python:

```
# declaração do dict
dicionario: dict = {'a' : 1}

# operação válida
dicionario['b'] = 10
```

E no fim da execução deste código, `dicionario` será o seguinte:
`{'a' : 1, 'b' : 10}`

Exercício 7

Escreva o código capaz de acessar os elementos indicados da seguinte `string`:

```
texto: str = 'a b c d e'
```

(a) A Letra “a”;

(b) A Letra “c”;

(c) A Letra “e”.

Solução

Considerando a seguinte indexação para a `string`:

0	1	2	3	4	5	6	7	8
a		b		c		d		e

Aqui está a solução do problema:

(a) `texto[0]`

(b) `texto[4]`

(c) `texto[8]`

Comportamento de operadores

O mesmo operador pode ter diferentes comportamentos baseados no tipo de variável que está sendo utilizado, e é extremamente importante se atentar a isto para que não ocorra nenhum comportamento inesperado na execução do seu programa.

Exemplos:

- `+` soma números (`ints` e `floats`), mas **concatena** strings e listas
  - `3 + 2 = 5`
  - `"Olá"+ "mundo" = "Olá mundo"`
  - `[1, 2, 3] + [4, 5, 6] = [1, 2, 3, 4, 5, 6]`
- `*` multiplica números (`ints` e `floats`), mas **repete** strings
  - `3 * 2 = 6`
  - `"Oi"* 3 = "OiOiOi"`

Entradas e saídas

Entradas e saídas de um programa

Já discutimos anteriormente o conceito de entradas e saídas de um programa. No entanto, surge a seguinte questão: como podemos, de fato, utilizá-las em nossos programas escritos em Python?

Para isso, precisamos de ferramentas que nos permitam coletar informações provenientes de diferentes dispositivos de entrada do computador, como o teclado, e exibir os resultados por meio de dispositivos de saída, como a tela.

Neste contexto, utilizaremos dois elementos fundamentais da programação em Python, que serão estudados com mais detalhes posteriormente: a função `input()`, responsável pela coleta de dados fornecidos pelo usuário, e a função `print()`, utilizada para exibir informações no terminal.

Função input() [parte 1]

A função `input()` tem como principal objetivo ler informações digitadas no terminal, retornando-as **sempre** no formato de uma `string`.

Sua sintaxe é bastante simples:

```
input(<texto que deve ser exibido no terminal>)
```

Nesse contexto, o trecho "`texto que deve ser exibido no terminal`" deve ser substituído por uma `string` contendo a mensagem que será exibida ao usuário, de modo a indicar claramente qual informação deve ser digitada. O valor lido pelo `input()` pode então ser armazenado em uma variável, conforme ilustrado a seguir:

```
texto: str = input('Escreva: ')
```

Função input() [parte 2]

Entretanto, há um certo problema com a função `input()`, pois nem tudo que nos interessa é uma `string`. Imagine a seguinte situação, queremos fazer um programa que leia dois números do terminal e realize a operação de adição entre eles, guardando o resultado final em uma variável. Você pode imaginar que este código realiza exatamente isto:

```
# ler os números
a = input('Insira um número: ')
b = input('Insira um número: ')

# somar os números
c = a + b
```

Função input() [parte 2]

```
# ler os números
a = input('Insira um número: ')
b = input('Insira um número: ')

# somar os números
c = a + b
```

Este código apenas soma duas `strings`, pois *a* e *b* são `strings`, ou seja, as concatena. Se *a* é igual a 1 e *b* é igual a 2, c terá um valor igual 12, não 3!

A solução para este problema é realizar uma conversão de tipos, e transformar a entrada de uma `string` para um `int` ou `float`.

Função input() [parte 3]

Para converter uma `string` para um `int` ou `float`, basta realizar `int()` ou `float()` na `string` desejada. Aqui está a correção do exemplo anterior:

```
# ler os números
a = int(input('Insira um número: '))
b = int(input('Insira um número: '))

# somar os números
c = a + b
```

Agora sim, c será o resultado correto de uma soma entre dois números inteiros.

Função print()

A função `print()` tem como principal objetivo exibir informações no terminal. Para isso, basta chamá-la passando como argumento o conteúdo que deve ser impresso. Ao executar o programa, essa informação será exibida na tela.

Sua sintaxe é bastante simples:

```
print(<texto que deve ser exibido no terminal>)
```

Essa função não retorna nenhum valor, ou seja, não é possível associá-la a uma variável. Além disso, é importante destacar que a função `print()` não se limita à impressão de `strings`, sendo também capaz de exibir outros tipos de dados, como `ints` e `floats`.

Função print() - exemplo

Podemos incrementar o exemplo utilizado para a função `input()` a fim de imprimir no final do programa o resultado da variável c. Para isto, podemos utilizar a função `print()`:

```
# ler os números
a = int(input('Insira um número: '))
b = int(input('Insira um número: '))

# somar os números
c = a + b

# imprimir resultados
print(c)
```

Pontos chave

- Variáveis armazenam valores para posterior acesso;
- Variáveis podem ser manipuladas através de diferentes operadores;
- Índices em Python se iniciam em 0;
- Um mesmo operador pode ter diferentes efeitos em diferentes tipos de variáveis;
- `input()` lê dados e `print()` os exibe.

Pratique muito!