Variáveis e tipos em Python

April 28, 2020

1 Variáveis e tipos em Python

Jorge Gustavo RochaDepartamento de Informática, Universidade do Minho27 de abril de 2020 O Python é uma linguagem tipada, isto é, as operações só fazem sentido para um determinado tipo de operando. Se o tipo do operando não é o adequado, o Python interrompe a execução.

Nestes exercícios vamos introduzir as variáveis, os tipos e os objectos. São alguns dos conceitos fundamentais para se programar em Python.

1.1 Variáveis

As variáveis são usadas para guardar dados, para posterior utilização. No exemplo seguinte definem-se duas variáveis `g` e `nome`que posteriormente são usadas na função `print()`.

[4]:
```
g = 9.8
nome = 'Galileu Galilei'
print( "A aceleração da gravidade é {} m/s^2 e já era do conhecimento do {}".
 ↪format( g, nome ))
```

```
A aceleração da gravidade é 9.8 m/s^2 e já era do conhecimento do Galileu
Galilei
```

1.1.1 Nomes das variáveis

Como em qualquer linguagem, o Python tem regras muito bem definidas para os identificadores.

1. As variáveis têm que começar com uma letra ou `_`. Não podem começar com algarismos.
2. As variáveis só podem conter letras, algarismos e `_`. Boa notícia para os portugueses: as letras acentuadas são aceites.
3. As letras maiúsculas são diferentes das minúsculas. Por isso, as variáveis `nome`, `Nome` e `NOME` são todas diferentes.

Exemplos de identificadores para variáveis:

Válido	Inválido
`data`	`2vezes`
`data2`	`%perc`
`ângulo`	`o ângulo`
`vel_inicial`	`vel-inicial`

Exercício Complete o código seguinte, de forma a provar que as duas variáveis são diferentes (por diferirem em maiúsculas e minúsculas).

[1]:
```
nome = 'Manuel'
Nome = 'Maria'
```

1.1.2 Atribuição

A atribuição de uma valor a uma variável, como já se viu, na sua forma mais simples é:

```
g = 9.8
```

Como usamos o símbolo `=` para atribuir valores a variáveis, para comparar dois valores, temos que usar um símbolo diferente, que é o `==`. Ou seja, as seguintes expressões são completamente diferentes:

```
g = 9.8
g == 9.8
```

A expressão `g == 9.8` dá sempre um resultado booleano (`True` ou `False`).

Em Python, pode-se fazer várias atribuições numa só linha. Por exemplo:

[10]:
```
r, g, b = "Vermelho", "Verde", "Azul"
print("Pode-se formar qualquer cor juntando: {}, {} e {}".format( r, g, b) )
```

```
Pode-se formar qualquer cor juntando: Vermelho, Verde e Azul
```

Pode-se, também, atribuir o mesmo valor a várias variáveis.

[2]:
```
i = k = j = 0
print(i, j, k)
```

```
0 0 0
```

Exercício Use a função predefinida divmod para calcular o quociente e o resto da divisão de 17 por 5.

Nota: A função `divmod()` retorna um `tuple` (um par).

[ ]:
```

```

1.2 Tipos

Os principais tipos predefinidos do Python são:

Categoria	Tipo
Numéricos	`int`, `float`, `complex`
Texto	`str`
Coleções	`list`, `tuple`, `range`
Dicionário	`dict`

Categoria	Tipo
Conjuntos	`set`, `frozenset`
Booleano	`bool`
Binários	`bytes`, `bytearray`, `memoryview`
Funções	`builtin_function_or_method`

Como o Python é tipado (como já se disse), temos que ter cuidado com as operações e o tipo dos operandos. Por exemplo, tente correr o seguinte código:

[19]:
```
from datetime import date
ano_atual = date.today().year

ano_nascimento = input()
print("Você tem {} anos.".format(ano_atual - ano_nascimento) )
```

```
1969
```

```
␣
↪---------------------------------------------------------------------------

TypeError                                 Traceback (most recent call␣
↪last)

<ipython-input-19-5e145f15d846> in <module>
  3 ano_atual
  4 ano_nascimento = input()
----> 5 print("Você tem {} anos.".format(ano_atual - ano_nascimento) )


TypeError: unsupported operand type(s) for -: 'int' and 'str'
```

Porque é que o exemplo anterior corre mal? Porque a variável `ano_nascimento` que fica com o resultado da função `input()` é do tipo `str`. Por isso, o Python não consegue fazer a subtração: `ano_atual - ano_nascimento` entre um `int` e uma `str`.

No código seguinte, assegura-se que a variável `ano_nascimento` é convertida em `int` com `int(ano_nascimento)`.

[22]:
```
from datetime import date
ano_atual = date.today().year

ano_nascimento = input()
print("Você tem {} anos.".format(ano_atual - int(ano_nascimento)) )
```

```
1969
Você tem 51 anos.
```

1.2.1 Tipo de uma variável

Para investigar o tipo de uma variável, usa-se a função `type()`.

[25]:
```
ano_nascimento = input()
type(ano_nascimento)
```

```
2001
```

[25]:
```
str
```

[29]:
```
planetas = [ 'Mercúrio', 'Vénus', 'Terra', 'Marte', 'Júpiter', 'Saturno',␣
 ↪'Urano', 'Neptuno' ]
type(planetas)
```

[29]:
```
list
```

[30]:
```
planetas[0]
```

[30]:
```
'Mercúrio'
```

[31]:
```
type(planetas[0])
```

[31]:
```
str
```

Uma variável em Python não tem que ter sempre o mesmo tipo. A qualquer momento pode-se atribuir um valor de outro tipo à mesma variável.

[34]:
```
x = 3.14159
x = 'Salazar Slytherin'
type(x)
```

[34]:
```
str
```

Caso seja preciso confirmar se uma variável é de um determinado tipo, pode-se usar a função `isinstance()`.

[5]:
```
x = 3.14159
isinstance(x,int)
```

[5]:
```
False
```

Exercício Prove que `'Amor de perdição'` e `"Amor de perdição"`são ambas do tipo `str` e que são igual entre si, sendo uma escrita com `'` e outra com `"`.

[ ]:
```

```

1.3 Funções

Nos exemplos anteriores, estamos recorrentemente a usar funções predefinidas do Python, como `print()`, `type()`, `len()` etc. As funções são uma peça fundamental para construir programas, pois permitem agregar uma sequência de instruções muito utilizada numa função, que depois se usa quantas vezes for preciso.

1.4 Classes e métodos

O Python é uma linguagem orientada a objectos. Isso significa que se podem desenvolver programas à custa de criar **classes**. As classes são uma forma de juntar dados e código numa única unidade. Enquanto que uma função agrega apenas código que se pode reutilizar, as classes permitem reutilizar dados e código.

É complicado começar a criar classes e a desenvolver programas orientados a objectos, mas não é difícil perceber um exemplo. Vamos tentar ilustrar a programação orientada a objectos com uma classe `Cão`.

Vamos fazer esta introdução para percebermos que existem **métodos** para além das **funções**, já que terá que usar muitos destes métodos, mesmo nos programas mais simples. Os métodos são muito utilizados em Python.

[40]:
```
from datetime import date

class Cão:

    def __init__(self, nome, nascimento):
        self.nome = nome
        self.nascimento = nascimento

    def ladra(self):
        print("Au, au!")

    def idade(self):
        return date.today().year - nascimento;
```

Já está criada a classe `Cão`. A classe dá-nos agora a possibilidade de criar objectos (dessa classe). Vamos criar então um programa com dois cães diferentes .

[43]:
```
x = Cão('Atena', 2007)
y = Cão('Apolo', 2018)
```

`x` e `y` são **objectos**. Estes objetos tem propriedades (atributos) e métodos associados (código). Vamos exemplificar:

[45]:
```
x.nome
```

[45]:
```
'Atena'
```

[46]:
```
y.ladra()
```

```
Au, au!
```

1.4.1 Métodos

Vamos focar-nos nos métodos por agora, só para reforçar que estes são ligeiramente diferentes das funções. Enquanto que uma função se invoca através do nome da função, os métodos são sempre invocados no contexto de um objecto. No exemplo anterior, usou-se a sintaxe `y.ladra()` para invocar o método `ladra()`.

[57]:
```
y.ladra()
```

```
Au, au!
```

Este introdução à programação orientada a objectos ajuda a perceber porque é que já encontrou nos exemplos anteriores situações em que se usam métodos em vez de funções.

Um exemplo muito simples é o método `str.format()` que já viu ser utilizado. Se é um método da classe `str`, usa-se no contexto de um objecto dessa classe, da mesma forma que o método `ladra()` se usou no contexto do um objeto da classe `Cão`.

Vamos aplicar o método `str.format()` a uma objeto da classe `str`. Começamos por confirmar que o objecto é uma instância da classe `str` (só por curiosidade):

[9]:
```
type("O gasóleo está a {:0.2f} €")
```

[9]:
```
str
```

Invocação do método `str.format()`:

[10]:
```
"O gasóleo está a {:0.2f} €".format(1.214)
```

[10]:
```
'O gasóleo está a 1.21 €'
```

Exercício Use o método `str.upper()` aplicado ao exercício anterior, para apresentar a mensagem em maiúsculas.

[ ]:
```

```

Da mesma forma, começará a usar outros métodos da classe `str`, como `str.upper()`, por exemplo. Começará também a usar métodos de muitas outras classes. Fica, para já, com a noção de que se a invocação for desta forma, trata-se de um método e não de uma função.

Nota: Complicando as coisas, na verdade o método `ladra()` é uma função da classe `Cão`. Por essa razão, é equivalente usar `y.ladra()` e `Cão.ladra(y)`. Mas isto já são coisas complicadas