



**universidade de aveiro**  
escola superior de tecnologia e  
gestão de águeda

# Introdução à Programação

Variáveis, expressões e statements

# Declaração de atribuição

Uma declaração de atribuição cria uma nova variável e atribui-lhe um valor.

```
>>> message = 'And now for something completely different'
```

Atribuição de uma **string** à variável com o nome *message*.

```
>>> n = 17
```

Atribuição do **integer** 17 à variável com o nome *n*.

```
>>> pi = 3.1415926535897932
```

Atribuição do valor **float** à variável com o nome *pi*.

# Nome de variáveis

Geralmente os programadores escolhem nomes significativos e de certo modo autoexplicativos para as suas variáveis.

Os nomes de variáveis podem ser tão longos quanto se queira e podem conter letras e números, no entanto não podem começar por um número. As letras podem ser tanto minúsculas como maiúsculas.

A utilização do carácter underscore também é permitida sendo frequentemente usado em nomes com várias palavras.

Exemplo: *your\_name* ou *password\_do\_utilizador*

# Nome de variáveis

Se for declarada uma variável com um nome ilegal o interpretador irá gerar um erro de sintaxe.

```
>>> 76trombones = 'big parade'
```

```
SyntaxError: invalid syntax
```

```
>>> more@ = 1000000
```

```
SyntaxError: invalid syntax
```

```
>>> class = 'Advanced Theoretical Zymurgy'
```

```
SyntaxError: invalid syntax
```

Qual o motivo para se obter estes três erros?

# Python keywords

Existem palavras reservadas que não podem ser usadas como nome de variáveis pois são usadas pelo interpretador para reconhecer a estrutura do programa.

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

# Expressões e statements

Uma expressão é uma combinação de valores, variáveis e operadores.

```
>>> n + 25
```

Um statement é uma unidade de código que tem um efeito, como criar uma variável ou exibir um valor.

```
>>> n = 17
```

Atribuição que dá o valor 17 à variável *n*.

```
>>> print(n)
```

Instrução que permite imprimir o valor de *n*.

# Modo script

Existem dois modos de correr o Python.

## ► Modo interativo

Interação direta com o interpretador. Uma boa forma de começar a explorar a linguagem.

```
>>> miles = 26.2  
>>> miles * 1.61  
42.182
```

## ► Modo script

Criação de programas complexos. Permite guardar o código em ficheiros aos quais chamamos **scripts** e que têm a extensão **.py**

```
miles = 26.2  
print(miles * 1.61)
```

# Ordem das operações

Quando uma expressão tem mais do que um operador a sua execução depende da ordem das operações.

O Python segue as convenções matemáticas para executar a ordem dos operadores.

O acrónimo **PEMDAS** permite relembrar a ordem das operações.

## ► Parenteses

$$2 * (3-1) = 4 \mid (1+1)**(5-2) = 8$$

## ► Exponenciação

$$1 + 2**3 = 9$$

## ► Multiplicação e Divisão subpõem-se à Adição e Subtração

$$2*3-1 = 5 \mid 6+4/2 = 8$$



# Operações com strings

Não é permitido efetuar operações matemáticas com string mesmo que estas sejam numéricas.

As seguintes instruções são operações ilegais.

‘Chinese’ - ‘ food’ ‘eggs’ / ‘easy’      ‘third’ \* ‘a charm’

Existem duas exceções, os operadores + e \*.

O operador + permite a **concatenação** de strings, isto significa que permite juntar duas strings.

```
>>> first = 'throat'  
>>> second = 'warbler'  
>>> first + second  
throatwarbler
```

O operador \* permite a repetição da string.

Exemplo: ‘Spam’ \* 3    →    ‘SpamSpamSpam’

À medida que os programas vão ficando maiores e mais complicados, ficam também mais difíceis de ler. Linguagens formais são densas e muitas vezes é difícil olhar para um pedaço de código e descobrir o que ele deverá fazer.

Para facilitar a leitura e compreensão do código é uma boa ideia adicionar notas nos programas para explicar em linguagem natural o que o programa está a fazer. Essas notas são chamadas de comentários e começam com o símbolo #:

**# calcula a percentagem da hora decorrida**

`percentagem = (minuto * 100) / 60`

ou

`percentagem = (minuto * 100) / 60` **#calcula a percentagem da hora decorrida**