



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

# Introdução à Programação

Funções

# Funções

Exemplo de chamada de função:

```
>>> type(42)
```

O nome da função é **type**.

A expressão entre parênteses é chamada de argumento da função.

É comum dizer que uma função “pega” num argumento e “devolve” um resultado. O resultado também é chamado de valor de retorno.

# Funções Math

O Python tem um módulo matemático que fornece a maioria das funções matemáticas.

Um módulo é um arquivo que contém uma coleção de funções relacionadas.

Antes de podermos usar as funções de um módulo, temos que importá-lo com uma declaração de importação:

```
>>> import math
```

Essa instrução cria um objeto do módulo chamado `math`.

O objeto do módulo contém as funções e variáveis definidas no módulo.

Para aceder a uma das funções, devemos especificar o nome do módulo e o nome da função, separados por um ponto.

```
>>> radians = 0.7
```

```
>>> height = math.sin(radians)
```

# Funções Math

```
>>> degrees = 45
>>> radians = degrees / 180.0 * math.pi
>>> math.sin(radians)
0.707106781187
```

A expressão `math.pi` obtém a variável *pi* do módulo *math*.

O argumento de uma função pode ser qualquer tipo de expressão, incluindo operadores aritméticos:

```
x = math.sin(degrees / 360.0 * 2 * math.pi)
```

E até chamadas de outras funções:

```
x = math.exp(math.log(x+1))
```

# Definição de Funções

Até agora, vimos apenas as funções que vêm com o Python, mas também é possível adicionar novas funções.

Uma definição de função especifica o nome de uma nova função e a sequência de instruções executadas quando a função é chamada.

```
def print_lyrics():  
    print("I'm a lumberjack, and I'm okay.")  
    print("I sleep all night and I work all day.")
```

**def** é uma palavra-chave que indica que esta é uma definição de função. O nome da função é *print\_lyrics*.

As regras para nomes de funções são as mesmas que para nomes de variáveis.

Devemos evitar ter uma variável e uma função com o mesmo nome.

Os parênteses vazios após o nome indicam que esta função não aceita nenhum argumento.

# Definição de Funções

A primeira linha da definição da função é chamada de header (cabeçalho) e o resto é chamado de corpo (body).

O cabeçalho deve terminar com dois pontos e o corpo deve ser indentado.

O corpo pode conter qualquer número de instruções.

Para finalizar a função, devemos inserir uma linha vazia.

Depois de definir uma função, podemos usá-la dentro de outra função.

```
def repeat_lyrics():
```

```
    print_lyrics()
```

```
    print_lyrics()
```

**A definição da função deve ser executada antes que a função seja chamada.**

# Parâmetros e argumentos

Algumas funções que vimos requerem argumentos.

Dentro da função, os argumentos são atribuídos a variáveis chamadas **parâmetros**.

```
def print_twice(bruce):  
    print(bruce)  
    print(bruce)
```

Esta função atribui o argumento a um parâmetro chamado *bruce*. Quando a função é chamada, ela imprime o valor do parâmetro (seja ele qual for) duas vezes.

```
>>> print_twice('Spam '*4)
```

```
Spam Spam Spam Spam
```

```
Spam Spam Spam Spam
```

```
>>> print_twice(math.cos(math.pi))
```

```
-1.0
```

```
-1.0
```

O argumento é avaliado antes da chamada da função, portanto, nos exemplos, as expressões 'Spam'\*4 e math.cos(math.pi) são avaliadas apenas uma vez.

# Parâmetros e argumentos

Também podemos usar uma variável como argumento:

```
>>> michael = 'Eric, the half a bee.'
```

```
>>> print_twice(michael)
```

```
Eric, the half a bee.
```

```
Eric, the half a bee.
```

O nome da variável que passamos como argumento (michael) não tem nada a ver com o nome do parâmetro (bruce).

# Variáveis e parâmetros são locais

Quando criamos uma variável dentro de uma função, ela é local, ou seja, só existe dentro da função.

Por exemplo:

```
def cat_twice(part1, part2):  
    cat = part1 + part2  
    print_twice(cat)
```

Essa função recebe dois argumentos, concatena-os e imprime o resultado duas vezes. Aqui está um exemplo que a usa:

```
>>> line1 = 'Bing tiddle '  
>>> line2 = 'tiddle bang.'  
>>> cat_twice(line1, line2)  
Bing tiddle tiddle bang.  
Bing tiddle tiddle bang.
```

Quando `cat_twice` termina, a variável `cat` é destruída. Se tentarmos imprimi-lo, obteremos uma exceção:

```
>>> print(cat)  
NameError: name 'cat' is not defined
```

# Funções

Existem dois tipos de funções, as que retornam um valor e as que não retornam valores (void).

O valor de retorno pode ser atribuído a uma variável ou usado como parte de uma expressão

```
x = math.cos(radians)
```

```
golden = (math.sqrt(5) + 1) / 2
```

As funções void podem exibir algo no ecrã ou ter algum outro efeito, mas não têm um valor de retorno.

Se se atribuir o resultado a uma variável, obterá um valor especial chamado None.

```
>>> result = print_twice('Bing')
```

```
Bing
```

```
Bing
```

```
>>> print(result)
```

```
None
```

# Por que funções?

Existem vários motivos:

- A criação de uma nova função oferece a oportunidade de nomear um grupo de instruções, o que torna o programa mais fácil de ler e de fazer debug.
- As funções podem tornar um programa menor eliminando o código repetitivo. Mais tarde, se fizermos uma alteração, basta fazê-la num sitio só.
- Dividir um programa longo em funções permite fazer debug a cada uma das partes, uma de cada vez.
- Funções bem projetadas geralmente são úteis para muitos programas. Depois de escrever e fazer debug a uma, podemos reutilizá-la.