Funções, Lambdas
Módulos e Pacotes

Prof. Saulo Oliveira
Técnico em Informática para Internet
Instituto Federal do Ceará

Funções

> As funções são blocos de instruções que realizam tarefas específicas;

Um problema complexo pode ser simplificado quando dividido em vários problemas menores. Chamamos esta estratégia de **Decomposição**, e com ela temos:

- Redução de complexidade;
- Permite focalizar a atenção em um problema pequeno de cada vez;
- Produz melhor compreensão do todo.

Os programas em geral são executados linearmente, uma linha após a outra, até o fim:

- O código de uma função é carregado uma vez e pode ser executado quantas vezes forem necessárias.
- As funções permitem a realização de desvios na execução dos programas.

Anatomia de uma função

```python
def soma_ate_n(n):
    total = 0
    for i in range( n + 1 ):
        total += i
    return total
```

- Assinatura: `def soma_ate_n(n)`.
- Nome da função: `soma_ate_n`.
- Parâmetros: `(n)`;
- Variáveis locais: `total, i`
- Corpo da função: Tudo que está recuado!
- Valor de retorno: `total`.

Parâmetro ou argumento?

Parâmetro é a variável que irá receber um valor em uma função (ou método) enquanto que um argumento é o valor (que pode originar de uma variável ou expressão) que você passa para a função (ou método). Você não passa parâmetros, você passa argumentos. Você recebe argumentos também, mas recebe em parâmetros.

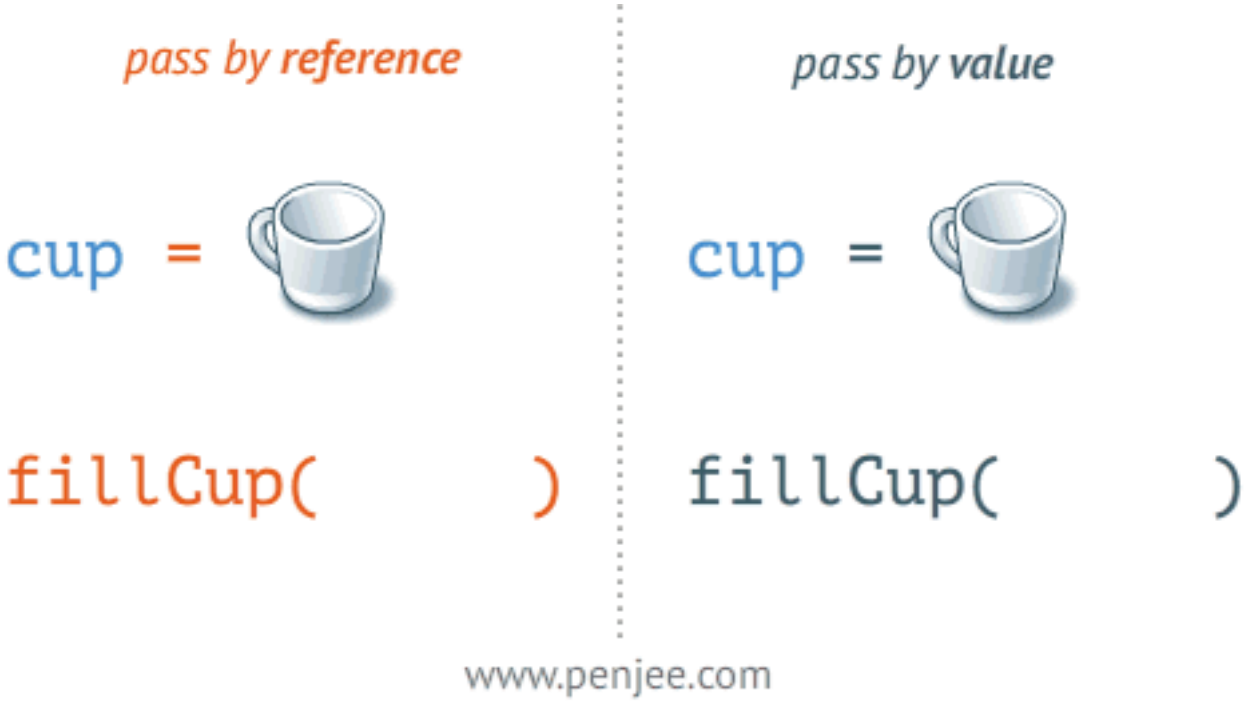
```
def dobro(x):
    return x * 2

n = dobro(5)
m = dobro(n)
o = dobro(5 * 7)
p = dobro(n + 1)

print(n, m, o, p)
```

Passagem de parâmetros

> Existem duas formas, a saber, por **valor** e por **referência**.



Fonte: https://devblog.drall.com.br/wp-content/uploads/2016/06/pass-by-reference-vs-pass-by-value-animation.gif

Passagem de parâmetros

```
x = 10
def muda(x):

  x = 5

muda(x)
print(x)
```

Passagem por valor – permite usar dentro de uma função uma cópia do valor de uma variável, porém não permite alterar o valor da variável original (somente a cópia pode ser alterada).

```
l = [1]
def muda(l):

  l[0] = 2

muda(l)
print(l)
```

Passagem por referência – É passada para a função uma referência da variável, sendo possível alterar o conteúdo da variável original usando-se esta referência.

Passagem por valor e por referência

> A maioria das linguagens (C, Java, ...) fazem distinção de uma passagem por valor e por referência.

No Python, tal distinção é, de certa forma, artificial, e é um pouco sutil quando suas variáveis serão modificadas ou não. Felizmente, existem regras claras.

Parâmetros para funções são referências a objetos que são passados por valor. Quando se passa uma variável a uma função, o Python passa a referência ao objeto ao qual a variável se refere (o valor), e não a variável propriamente dita.

- Se o valor é **imutável**, a função não modifica a variável chamada.
- Se o valor é **mutável**, a função pode modificar a variável chamada.

Tipos mutáveis e imutáveis

Em **tipos imutáveis**, você não pode alterar o valor durante a execução do código.

Tipos de dados **mutáveis** são outra face dos tipos integrados, eles permitem alterar o valor de itens específicos sem afetar a identidade do objeto contêiner.

Ao lado, está um resumo de quais tipos integrados são mutáveis e quais são imutáveis:

Tipo de dados	Classe integrada	Mutável
Números	`int`, `float`, `complex`	❌
Strings	`str`	❌
Tuplas	`tuple`	❌
Bytes	`bytes`	❌
Booleanos	`bool`	❌
Conjuntos congelados	`frozenset`	❌
Listas	`list`	✅
Dicionários	`dict`	✅
Conjuntos	`set`	✅
Matrizes de bytes	`bytearray`	✅

Namespaces

> Um *namespace* é uma coleção de nomes simbólicos atualmente definidos junto com informações sobre o objeto ao qual cada nome faz referência.

Em um programa Python, existem quatro tipos de *namespaces*: **integrado** (*built-in*), **global**, **local** e **fechado** (*enclosing*). Você pode pensar em um *namespace* como um dicionário no qual as chaves são os nomes dos objetos e os valores são os próprios objetos. Cada par de valores-chave mapeia um nome para seu objeto correspondente.

```
>>> nome = 'Saulo'
>>> dir()
['__annotations__', '__builtins__', '__doc__',
'__loader__', '__name__', '__package__', '__spec__', 'nome']
```

Namespaces integrado

O *namespace* integrado contém os nomes de todos os objetos integrados do Python e estão disponíveis sempre que o Python está em execução.

Você pode listar os objetos no *namespace* integrado com o seguinte comando:

```
>>> dir(__builtins__)
['ArithmeticError', 'AssertionError', 'AttributeError', ..., 'len', 'license', 'list',
 'locals', 'map', 'max', 'memoryview', 'min', 'next', 'object', 'oct',
 'open', 'ord', 'pow', 'print', 'property', 'quit', 'range', 'repr',
 'reversed', 'round', 'set', 'setattr', 'slice', 'sorted', 'staticmethod',
 'str', 'sum', 'super', 'tuple', 'type', 'vars', 'zip']
```

Namespaces global

O **namespace global** contém quaisquer nomes definidos no nível do programa principal e ele permanece existente até que o interpretador termine.

```
>>> globals()
{'__name__': '__main__', '__doc__': None,
 '__package__': None, '__loader__': <class '_frozen_importlib.BuiltinImporter'>,
 '__spec__': None, '__annotations__': {},
 '__builtins__': <module 'builtins' (built-in)>,
 'nome': 'Saulo'}
```

Namespace local

Python também fornece uma função integrada correspondente chamada `locals()`. É semelhante `globals()`, mas acessa objetos no ***namespace local***:

```
>>> def f(x, y):
...     s = 'foo'
...     print(locals())
...

>>> f(10, 0.5)
{'s': 'foo', 'y': 0.5, 'x': 10}
```

> ⚠️ Se você chamar `locals()` fora de uma função no programa principal, ela se comportará da mesma forma que `globals()`.

Exercícios

1. Qual é o resultado da execução deste código?

```
def funcao1():
    x = 10

def funcao2():
    print(x)

funcao1()
funcao2()
```

2. O que será impresso ao executar este código?

```
y = 20

def funcao3():
    y = 30
    print(y)

funcao3()
print(y)
```

Exercícios

3. O código abaixo apresentará um erro. Identifique o problema.

```
def funcao4():
    z = 5

funcao4()
print(z)
```

4. Qual é a saída esperada deste código?

```
def funcao5():
    a = 15
    print(a)

a = 5
funcao5()
print(a)
```

Lambdas, clojures e afins

Módulo

> Um módulo é um arquivo .py contendo definições e comandos Python:

- Funções;
- Variáveis e constantes; e
- Classes.

Pacote

Qualquer diretório do sistema operacional que contém um arquivo `__init__.py` dentro é considerado um pacote. Python dá suporte à hierarquia de pacotes onde podemos ter uma árvore de pacotes separadas por pastas.

Referências

https://python-textbok.readthedocs.io/en/1.0/Variables_and_Scope.html

https://estruturas.ufpr.br/disciplinas/pos-graduacao/introducao-a-computacao-cientifica-com-python/introducao-python/1-4-funcoes/

https://realpython.com/python-mutable-vs-immutable-types/