

# FUNÇÕES: ESCOPO, PARÂMETROS E DOCUMENTAÇÃO

MC102 - Algoritmos e  
Programação de  
Computadores

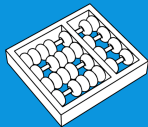
Santiago Valdés Ravelo  
[https://ic.unicamp.br/~santiago/  
ravelo@unicamp.br](https://ic.unicamp.br/~santiago/ravelo@unicamp.br)

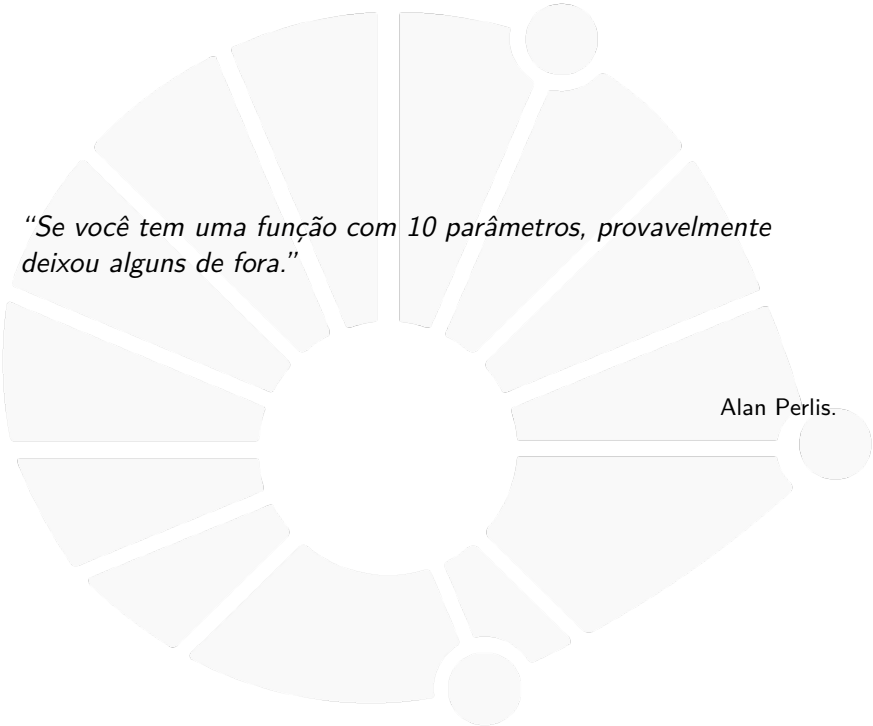
04/24

10



UNICAMP



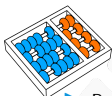


*“Se você tem uma função com 10 parâmetros, provavelmente deixou alguns de fora.”*

Alan Perlis.



# DÚVIDAS DA AULA ANTERIOR

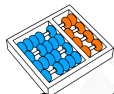


### Dúvidas selecionadas

- ▶ Posso ter funções dentro de funções? Se sim, essas funções são válidas no código todo ou só dentro daquela função?
- ▶ Se eu fizer uma função dentro de uma função, o parâmetro da função mais externa serve pra de dentro também?
- ▶ Então funções são blocos de código que fazem coisas específicas e chamá-las seria como chamar um código todo em uma "palavra"?
- ▶ O return dentro de um laço, quebra ele imediatamente? Ou eu posso continuar o laço e retornar ainda outro valor?
- ▶ Quando a função altera o valor da variável não seria mais interessante criar outra variável com um mesmo valor para usa-la na função afim de manter a variável com seu valor original?
- ▶ Não entendi tão bem quais são as propriedades do "none", que retorna em algumas funções. Pra que ele serve e por que ele é usado em python em vez existir uma função "void"?
- ▶ No exemplo do def imprime(lista), não entendi porque no final ele imprime None.
- ▶ É possível definir uma função embaixo do trecho de código que a utiliza?
- ▶ Ao passar um lista como um parâmetro de uma função, a lista conta como 1 parâmetro ou mais? Por exemplo: l = [1, 2, 3], ao chamar uma função funcao(l), ela recebeu 1 ou 3 parâmetros?
- ▶ Chamar uma função sem parênteses (para que não seja executada) retorna o próprio valor da função? esse valor pode ser atribuído a uma variável e passado como parâmetro para outras funções?
- ▶ Em outras linguagens, se não me engano, na hora de definir a função você também tem que definir o tipo do parâmetro que a função vai receber, já que isso não é necessário em python, é possível receber no mesmo parâmetro variáveis de tipo diferentes?
- ▶ O que diferencia uma função em C e uma função em Python, tal que em C exista o conceito de prototype e em Python não?
- ▶ Como funcionam as demonstrações na programação? Por exemplo, eu entendo o algoritmo e percebo que ele é capaz de dar a resposta correta. Contudo, existe uma forma rigorosa de demonstrar isso? Seria através de pseudocódigo ou matemática?



# VARIÁVEIS LOCAIS E GLOBAIS



## Variáveis

O que é uma variável?

- ▶ Um lugar da memória.
- ▶ Para o qual demos um nome.

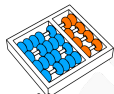
Podemos ter novas variáveis dentro das funções.

Ex:

```
1 def soma_dos_digitos(x):  
2     soma = 0  
3     while x > 0:  
4         soma += x % 10  
5         x //= 10  
6     return soma
```

**soma** é o que chamamos de variável local.

- ▶ Ela existe apenas dentro da função.
- ▶ Perde seu valor quando a função termina.



## Variáveis locais

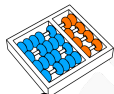
Um exemplo:

```
1 def funcao(x):
2     y = 2 * x
3     return y
4
5 z = funcao(10)
6 print(z)
7 print(y)
```

O que é impresso por esse código?

```
1 20
2 Traceback (most recent call last):
3   File "vars1.py", line 7, in <module>
4     print(y)
5 NameError: name 'y' is not defined
```

**y** não pode ser acessada na linha 7.



## Variáveis globais

Outro exemplo:

```
1 def imprime():  
2     print(z)  
3  
4 z = 10  
5 imprime()  
6 print(z)
```

O que é impresso por esse código?

```
1 10  
2 10
```

**z** pode ser acessada na linha 2.

- ▶ **z** é uma variável **global**.
- ▶ Ela pode ser acessada em qualquer função.





### Começando a confusão...

Mais um exemplo:

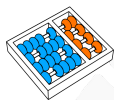
```
1 def imprime():
2     z = 8
3     print(z)
4
5 z = 10
6 imprime()
7 print(z)
```

O que é impresso por esse código?

```
1 8
2 10
```

O que aconteceu?

- ▶ O Python criou uma variável **local** chamada **z**.
- ▶ Que não é a mesma variável **global** chamada **z**.
- ▶ Elas podem ter valores diferentes!
- ▶ Não importa que elas tenham o mesmo nome!



## Resolvendo

“Corrigindo” o exemplo anterior:

```
1 def imprime():
2     global z
3     z = 8
4     print(z)
5
6 z = 10
7 imprime()
8 print(z)
```

O que é impresso por esse código?

- 1 8
- 2 8

Dizemos para o Python que queremos usar a variável **global z**.



# Variáveis Locais e Globais

### Variáveis Locais:

- ▶ São definidas dentro da função:
  - ▶ Na primeira atribuição.
- ▶ Só existem dentro da função:
  - ▶ Dizemos que o escopo da variável é a função.
- ▶ Podem ter o mesmo nome que variáveis globais.
- ▶ Note que um parâmetro é uma variável local.

### Variáveis Globais:

- ▶ São definidas fora de qualquer função.
- ▶ Podem ser escritas dentro das funções:
  - ▶ Mas é necessário usar o **global**.
  - ▶ Melhor devolver o valor do que alterar diretamente.
- ▶ Podem ser lidas dentro das funções:
  - ▶ Não precisa usar o **global**.
  - ▶ Mas não pode ter uma variável local com o mesmo nome.
  - ▶ Não usar o **global** pode levar a bugs!
- ▶ Idealmente não são manipuladas diretamente pelas funções.



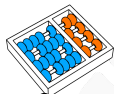
ESCOPO



## O que é?

O **escopo** de um nome (de variável, de função, etc):

- ▶ É a região do programa onde esse nome é válido.
- ▶ Isto é, onde esse nome pode ser acessado.
- ▶ Vimos o escopo global:
  - ▶ Variável é acessível em qualquer parte do programa.
- ▶ O escopo local:
  - ▶ Variável é acessível apenas dentro da função onde foi criada.
- ▶ Temos também o escopo *built-in*:
  - ▶ Funções como **int**, **input**.
- ▶ Mas também temos o escopo *enclosing*.

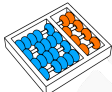


## Escopo *enclosing*

O seguinte código é válido em Python (e imprimirá 30):

```
1 def f(x):  
2     a = 10  
3     def g(y):  
4         print(a * y)  
5         g(x + 1)  
6  
7 f(2)
```

- ▶ **g** é uma função local de **f**.
- ▶ **a** e **x** são variáveis locais de **f**.
- ▶ **y** é variável local de **g**.
- ▶ **a** não é variável local de **g** e não é variável global...
  - ▶ **a** é acessível em **g**.
  - ▶ Está *enclosing*.



## Objetos, Classes e Métodos

O que as variáveis armazenam?

- ▶ **Objetos** de um certo **tipo** (ou **classe**).

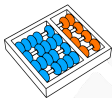
As classes definem **métodos** que podem ser utilizados pelos objetos:

- ▶ Funções que acessam ou modificam os objetos.

Exemplo:

- ▶ `l = [1, 2, 3]` cria um objeto do tipo `list`.
- ▶ `append` é um método de `list` que adiciona um elemento ao final da lista.
- ▶ Ou seja, podemos escrever `l.append(4)`.

Em breve aprenderemos a criar nossas próprias classes!



## Funções e Listas

Qual o resultado desse código?

```
1 def soma_um(x):  
2     x += 1  
3 x = 10  
4 soma_um(x)  
5 print(x)
```

E deste?

```
1 def soma_um(lista):  
2     for i in range(len(lista)):  
3         lista[i] += 1  
4 lista = [1, 2, 3]  
5 soma_um(lista)  
6 print(lista)
```

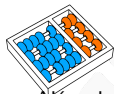
A função altera a lista

- ▶ Também alteraria se fizéssemos `append`.
- ▶ Mais sobre isso em breve.





# DOCUMENTAÇÃO



## Documentando funções

Além dos comentários normais, devemos usar **docstrings**:

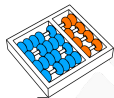
- ▶ Uma string que começa e termina com `"""`.
- ▶ Sempre na primeira linha da função.
- ▶ E que diz o que a função faz.

Ex:

```
1 def soma_dos_digitos(x):
2     """Calcula a soma dos digitos do número x na base 10."""
3     soma = 0
4     while x > 0:
5         soma += x % 10
6         x //= 10
7     return soma
```

Isso permite usar a função **help** (e é usado pelos editores).

```
1 >>> help(soma_dos_digitos)
2 Help on function soma_dos_digitos in module
   exemplo_docstring:
3
4 soma_dos_digitos(x)
5     Calcula a soma dos digitos do número x na base 10.
```



## Docstrings de várias linhas

Em geral, precisamos de várias linhas na documentação.

```
1 def soma_dos_digitos(x):
2     """Calcula a soma dos digitos do número x na base 10.
3
4     Funciona apenas para números inteiros não negativos.
5
6     Parâmetros:
7     x -- número inteiro positivo
8     """
9     soma = 0
10    while x > 0:
11        soma += x % 10
12        x //= 10
13    return soma
```

Convenção de Estilo do Python (PEP 257):

- ▶ Primeira linha diz o que a função faz brevemente.
- ▶ Deve haver uma linha em branco após a primeira linha.
- ▶ Terminamos com `"""` sozinho na última linha.



# PARÂMETROS



## Valor padrão

É possível definir um valor padrão para um parâmetro.

Ex:

```
1 def soma(x, y = 2, z = 7):
2     return x + y + z
3
4 print(soma(3))           # imprime 3 + 2 + 7 = 12
5 print(soma(3, 4))       # imprime 3 + 4 + 7 = 14
6 print(soma(3, 4, 2))    # imprime 3 + 4 + 2 = 9
```

Se você quiser passar um parâmetro na posição *i*:

- ▶ Precisa passar todos os parâmetros anteriores.
- ▶ Mesmo com os valores padrão.



## Parâmetros nominais

É possível usar o nome do parâmetro na chamada da função.

Ex:

```
1 def soma(x, y=2, z=7):  
2     return x + y + z  
3  
4 print(soma(x=3))           # imprime 3 + 2 + 7 = 12  
5 print(soma(x=3, y=4))     # imprime 3 + 4 + 7 = 14  
6 print(soma(x=3, z=2))     # imprime 3 + 2 + 2 = 7  
7 print(soma(x=3, y=4, z=2)) # imprime 3 + 4 + 2 = 9
```



## Combinação

Você até pode misturar os dois:

- ▶ Isto é, ter parâmetros posicionais e nominais.
- ▶ Porém, precisa começar com os parâmetros posicionais.

Ex:

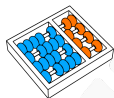
```
1 def soma(x, y=2, z=7):
2     return x + y + z
3
4 print(soma(3))           # imprime 3 + 2 + 7 = 12
5 print(soma(x=3))        # imprime 3 + 2 + 7 = 12
6 print(soma(3, y=4))     # imprime 3 + 4 + 7 = 14
7 print(soma(x=3, y=4))   # imprime 3 + 4 + 7 = 14
8 print(soma(3, z=2))     # imprime 3 + 2 + 2 = 7
9 print(soma(x=3, z=2))   # imprime 3 + 2 + 2 = 7
10 print(soma(x=3, y=4, z=2)) # imprime 3 + 4 + 2 = 9
11 print(soma(3, 4, z=2))  # imprime 3 + 4 + 2 = 9
```

Há formas de forçar parâmetros serem apenas posicionais ou apenas nominais.



# EXERCÍCIOS

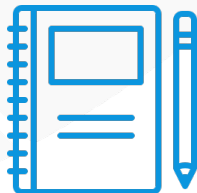


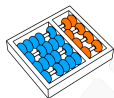


## Listas e repetição



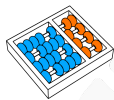
**Vamos fazer alguns exercícios?**





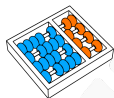
## Exercício 1

Faça uma função que, dado um número **n** e um valor **v** (por padrão, valendo **0**), cria uma lista de **n** posições com cada posição valendo **v**. Documente sua função.



## Exercício 2

Faça uma função que, dada uma lista **l** e um valor **v** (por padrão, valendo **0**), modifica todas as entradas da lista para **v**. Documente sua função.



### Exercício 3

Faça uma função que, dados dois números inteiros  $m$  (por padrão valendo **1**) e  $n$  (por padrão valendo **5**), desenhe  $m$  triângulos invertidos na tela, cada um com base  $n$ , usando caracteres '\*'.  
▶ Com os valores padrão deve imprimir:

```
* * * * *
 * * *
  * *
   *
```

# FUNÇÕES: ESCOPO, PARÂMETROS E DOCUMENTAÇÃO

MC102 - Algoritmos e  
Programação de  
Computadores

Santiago Valdés Ravelo  
[https://ic.unicamp.br/~santiago/  
ravelo@unicamp.br](https://ic.unicamp.br/~santiago/ravelo@unicamp.br)

04/24

10



UNICAMP

