



# PONTIFÍCIA UNIVERSIDADE CATÓLICA DE CAMPINAS

## Escola Politécnica - Curso de Engenharia de Software

### 12413 - ALGORITMOS DE PROGRAMAÇÃO, PROJETOS E COMPUTAÇÃO

ASSUNTOS:

- funções

*Profa. Angela de Mendonça Engelbrecht*

[angel@puc-campinas.edu.br](mailto:angel@puc-campinas.edu.br)

[profa.angela@gmail.com](mailto:profa.angela@gmail.com)

**Funções** – são módulos de programas (também chamados de subprogramas, sub-rotinas, procedimentos) – que são executados quando “chamados” em um fluxo de programa em execução.

- de modo geral - é um **mecanismo utilizado para estruturar** os programas;
- sua utilização em programação segue alguns princípios, tais como:
  - **maximizar o reuso** de código e minimizar redundâncias;
  - são pedaços/fatias de código, com regras **MUITO** bem definidas, com poucas interações com outras partes do programa;

mecanismo para construir funções:

parênteses – tendo ou não argumentos

*def* **nome** ( $arg_1, arg_2, \dots, arg_n$ ) :  
comandos  
  
*return* variável/constante

dois pontos

- *def* - palavra reservada – definir uma função;
- **nome**: nome escolhido para a função;
- $arg_i$ : lista de argumentos/parâmetros da função – comunicação de informações com o restante do programa
- *return*: **opcional** – depende da função – retorna informação

Existem **variações** nos usos dos **argumentos** da lista de parâmetros:

**Vamos abordar as seguintes variações nos argumentos das funções:**

- sem argumentos;
- com argumentos:
  - “simples”;
  - com valor padrão;
  - nomeados;

## I - Funções sem argumentos:

```
File Edit Format Run Options Window Help
''' exemplo função sem parâmetros '''

''' função para imprimir cabeçalho '''
def cabecalho ():
    print('-'*20)
    print('PONTIFÍCIA UNIVERSIDADE CATÓLICA DE CAMPINAS')
    print('      Curso de Engenharia de Software')
    print('-'*45)
    print(' Aluno: Angela Engelbrecht')
    print(' RF: 831159')
    print('-'*45)
    return

'''início do programa principal
    Calcular a área de uma quadrilátero'''

#chamada da função
cabecalho()

print(' <<< CALCULAO DA ÁREA DE QUADRILÁTERO >>>')
print('      (quadrado/retângulo)')
print(' <<< Entrada de Dados >>>')

while True:
    try:
        base = float(input('Base: '))
        break
    except:
        print('Digite número REAL')

while True:
    try:
        altura = float(input('Altura: '))
        break
    except:
        print('Digite número REAL')

area = base * altura
print('-'*45)
print(' <<< RESULTADO >>>')
print(' Área do '+( 'QUADRADO' if base == altura else 'RETÂNGULO') + f' = {area:.2f}')
print('-'*45)

print(' --- FIM DO PROGRAMA ----')
```

return aqui é opcional

chamada

```
-----
PONTIFÍCIA UNIVERSIDADE CATÓLICA DE CAMPINAS
      Curso de Engenharia de Software
-----
Aluno: Angela Engelbrecht
RF: 831159
-----
<<< CALCULAO DA ÁREA DE QUADRILÁTERO >>>
      (quadrado/retângulo)
<<< Entrada de Dados >>>
Base: 2.5
Altura: 5.6
-----
<<< RESULTADO >>>
Área do RETÂNGULO = 14.00
-----
--- FIM DO PROGRAMA ---
>>>
```

## II - Funções argumentos “simples”:

opcional – quando não tem retorno de valor pode ser omitido

resultado retorna pelo comando return

chamada

File Edit Format Run Options Window Help

```
''' exemplo função sem parâmetros '''

''' função para imprimir cabeçalho '''

def cabecalho ():
    print('-'*20)
    print('PONTIFÍCIA UNIVERSIDADE CATÓLICA DE CAMPINAS')
    print('      Curso de Engenharia de Software')
    print('-'*45)
    print(' Aluno: Angela Engelbrecht')
    print(' RF: 831159')
    print('-'*45)
    return

''' função para cálculo da área do quadrilátero'''

def calc_area (b, h):
    area = b * h
    return area
```

```
'''início do programa principal
    Calcular a área de uma quadrilátero'''

#chamada da função
cabecalho()

print(' <<< CALCULO DA ÁREA DE QUADRILÁTERO >>>')
print('          (quadrado/retângulo)')
print(' <<< Entrada de Dados >>>')

while True:
    try:
        base = float(input('Base: '))
        break
    except:
        print('Digite número REAL')

while True:
    try:
        altura = float(input('Altura: '))
        break
    except:
        print('Digite número REAL')

area = calc_area(base, altura)
print('-'*45)
print(' <<< RESULTADO >>>')
print(' Área do '+'(QUADRADO' if base == altura else 'RETÂNGULO') + f' = {area:.2f}')
print('-'*45)

print(' --- FIM DO PROGRAMA ----')
```

```
-----
PONTIFÍCIA UNIVERSIDADE CATÓLICA DE CAMPINAS
      Curso de Engenharia de Software
-----
Aluno: Angela Engelbrecht
RF: 831159
-----
<<< CALCULO DA ÁREA DE QUADRILÁTERO >>>
      (quadrado/retângulo)
<<< Entrada de Dados >>>
Base: 10
Altura: 10
-----
<<< RESULTADO >>>
Área do QUADRADO = 100.00
-----
--- FIM DO PROGRAMA ---
>>>
```

## II - Funções com valor padrão

quando o argumento **tem um valor atribuído** – usado **quando** não está especificado um valor na chamada

### EXEMPLO 1 apenas 1 parâmetro com valor

```
File Edit Format Run Options Window Help
''' função com valor padrão '''
def soma (a , b = 5):
    return a + b
''' programa principal '''
x = 10
y = 20
print('Soma1 = ', soma(x,y))
print('Soma2 = ', soma(x))
print('Soma3 = ', soma(y))
```

```
r padrão.py
Soma1 = 30
Soma2 = 15
Soma3 = 25
>>> |
```

### EXEMPLO 2 os 2 parâmetros com valor

```
File Edit Format Run Options Window Help
''' função com valor padrão '''
def soma (a = 3 , b = 5):
    return a + b
''' programa principal '''
x = 10
y = 20
print('Soma1 = ', soma(x,y))
print('Soma2 = ', soma(x))
print('Soma3 = ', soma(y))
print('Soma4 = ', soma())
```

```
r padrão.py
Soma1 = 30
Soma2 = 15
Soma3 = 25
Soma4 = 8
>>> |
```

Funções **com** valor padrão - continuação

**Obs:** os argumentos com valores, **devem ser colocados no final**, uma vez que os valores que não estão na chamada devem ser os que possuem valor default.

veja erro:

```
>>> def soma (a = 3, b): return a+b
SyntaxError: non-default argument follows default argument
>>> |
```

Outra situação **de erro** – argumentos que **não possuem valor** devem receber valor da chamada:

```
>>> def soma (a , b, c = 5):
    return a + b + c

>>> soma (10,20,30)
60
>>> soma (10, 20)
35
>>> soma (10)
Traceback (most recent call last):
  File "<pyshell#21>", line 1, in <module>
    soma (10)
TypeError: soma() missing 1 required positional argument: 'b'
>>> |
```

erro – argumentos **a e b** não possuem valor e na chamada não foi passado valor



## Estruturas (string, lista e dict) e as funções

como os **números** e a **string** são **imutáveis**, seus valores originais, **quando na lista de parâmetros**, não são modificados

```
def leia_str (f):  
    f = input('Digite uma frase: ')  
    pal = input('Digite uma palavra: ')  
    return pal
```

```
''' principal '''  
# leitura de uma frase e uma palavra  
frase = 'Boa tarde'  
palavra = "Mesa"  
palavra = leia_str(frase)  
print(f'Frase: {frase} \n Palavra: {palavra}')
```

repare que a **string** como **argumento**, não **muda o valor** e somente a que **recebe o valor do retorno**!

comando de leitura - dentro da função

```
Digite uma frase: Tudo bem com você?  
Digite uma palavra: Oi  
Frase: Boa tarde  
Palavra: Oi
```

por sua vez, as **listas** e os **dicionários** compartilham as ações executadas dentro da função.

```
def leia_lista_dict(N, L):
    D = {}
    N = int(input('Numero de elementos: '))
    for i in range(N):
        ra = input('RA:')
        nome = input('Nome:')
        L.append([ra,nome])
        D[ra] = nome
    return D

def incluir_aluno (L, D):
    ra = input('RA:')
    nome = input('Nome:')
    L.append([ra,nome])
    D[ra] = nome
    return
```

```
''' principal '''
# listas e dicionários e as funções

lista = []
N =0
Dict= leia_lista_dict(N, lista)
print('Valor de N após a chamada: ',N)
print('\nLista:', lista)
print('Dicionário:',Dict)
print('>> INCLUIR DADOS DE ALUNO \n')
incluir_aluno(lista, Dict)

print('Lista:',lista)
print('Dicionario: ', Dict)
```

aqui: o **mesmo** comportamento da **lista** é o do **dicionário**, em relação aos argumentos e no **return**

no exemplo, na **1ª. função**: a **lista** está como argumento e o **dicionário** no return.

no exemplo, na **2ª. função**: ambas a estruturas de dados estão como argumento.

**obs.:** a leitura do valor para N na função apenas para exemplificar que seu valor não modifica

```
Numero de elementos: 2
RA:1234
Nome:Ana Maria
RA:2345
Nome:Rosa Maria
Valor de N após a chamada: 0


Lista: [['1234', 'Ana Maria'], ['2345', 'Rosa Maria']]
Dicionário: {'1234': 'Ana Maria', '2345': 'Rosa Maria'}
>> INCLUIR DADOS DE ALUNO

RA:4567
Nome:Luiz Antonio
Lista: [['1234', 'Ana Maria'], ['2345', 'Rosa Maria'],
['4567', 'Luiz Antonio']]
Dicionario: {'1234': 'Ana Maria', '2345': 'Rosa Maria',
, '4567': 'Luiz Antonio'}
```

# Estruturas (string, lista e dict) e as funções

Outra possibilidade: mais de uma variável no retorno

```
def leia_lista_dict(L):
    D = {}
    N = int(input('Numero de elementos: '))
    for i in range(N):
        ra = input('RA:')
        nome = input('Nome:')
        L.append([ra,nome])
        D[ra] = nome
    return D,N
```



```
''' principal '''
# listas e dicionários e as funções

lista = []

Dict,N= leia_lista_dict(lista)
```

## Exemplos com funções

# EXEMPLO 1

Exemplo de função com dicionário na lista de parâmetros

obs.: o mesmo comportamento se aplica às listas

```
File Edit Format Run Options Window Help
1 ''' funções com estruturas como Dicionário, '''
2
3 ''' popular um dicionário com um conjunto de N livros'''
4 def popular(D:dict):
5     N = int(input('Numero de livros: '))
6     for i in range(N):
7         codlivro = int(input('Codigo do Livro:'))
8         while valida(D,codlivro):
9             print(">>> Código já existe!")
10            codlivro = int(input('Codigo do Livro:'))
11            titulo = input('Título:')
12            preco = float(input('Preço:'))
13            D[codlivro] = [titulo,preco]
14
15 '''verifica se código existe ou não no dicionário
16 se existir return True e, False em caso contrário'''
17 def valida(D:dict, cod:int):
18     if cod in D.keys(): return True
19     else: return False
20
21 ''' imprime livros dentro de uma faixa de preço'''
22 def imprimeLivros(D:dict, pI:float, pF:float):
23
24     for cod, lista in D.items():
25         if pI<=lista[1]<=pF:
26             print(f'Livro: {lista[0]:<20} Código: {cod:4d}\
27 Preço: R${lista[1]:.2f}')
28
29 '''----- principal----- '''
30 # dicionários e as funções
31
32 Livros={}
33 print(' <<< ENTRADA DE DADOS >>>\n')
34
35 popular (Livros)
36
37 print('\n <<< ENTRADA DO INTERVALO DE BUSCA >>>\n')
38
39 precoI = float(input('Preço Inicial para busca:'))
40 precoF = float(input('Preço Final para busca:'))
41
42 print('\n <<< RELATÓRIO DE LIVROS >>>\n')
43
44 imprimeLivros(Livros, precoI, precoF)
45
46 print('\n\n >>> FIM DE PROGRAMA...')
47
```

```
===== RESTART: C:/Users/profa/OneDrive/
<<< ENTRADA DE DADOS >>>

Numero de livros: 3
Codigo do Livro:1234
Titulo:Programação Python
Preço:98
Codigo do Livro:1234
>>> Código já existe!
Codigo do Livro:3456
Titulo:Programação C/C++
Preço:95.55
Codigo do Livro:5432
Titulo:Programação Java
Preço:45

<<< ENTRADA DO INTERVALO DE BUSCA >>>

Preço Inicial para busca:50
Preço Final para busca:100

<<< RELATÓRIO DE LIVROS >>>

Livro: Programação Python      Código: 1234 Preço: R$98.00
Livro: Programação C/C++      Código: 3456 Preço: R$95.55
>>> FIM DE PROGRAMA...
<<<
```

EXEMPLO: uso do argumento com **valor padrão**

#exemplo: considerar apenas uma parte de algum programa de cadastro de aluno

```
...
print('<< DADOS DE UM ALUNO >>')
```

```
while True:
    try:
        RA = int(input(" RA: "))
        break
    except ValueError:
        print('Digite número int')
```

```
while True:
    try:
        IDADE = int(input(" IDADE:"))
        break
    except ValueError:
        print('Digite número int')
```

```
while True:
    try:
        NOTA = float(input("NOTA: "))
        break
    except ValueError:
        print('Digite número float')
```

```
NOME = input(" Nome: ")

print()
print('-'*30)
print(" << DADOS DO ALUNO >>")
print(f" >> RA: {RA}")
print(f" >> NOME: {NOME}")
print(f" >> IDADE: {IDADE}")
print(f" >> NOTA: {NOTA}")
print('-'*30)
```

**observar as validações – exceções**  
são muito parecidas – observe as diferenças:

- o texto indicativo para o dado de leitura (*prompt*)

- o conversor do tipo *int()* ou *float()* de acordo com o conteúdo a ser lido

```
while True:
    try:
        var = tipo(input(mensagem_imprimir))
        break
    except ValueError:
        print(mensagem_erro)
```

a **título de exemplo**: a ideia é construir uma função para receber como parâmetro esses 3 elementos de leitura e validação, reduzindo o trecho de programa das validações

File Edit Format Run Options Window Help

```
# exemplo do uso do argumento com valor padrão

def leia ( prompt, msgerro= ' Erro: Digite número ', tipo = 'int'):
    while True:
        try:
            var = int(input(prompt)) if tipo == 'int' else float(input(prompt))
            return var
        except ValueError:
            print(msgerro + tipo)

#exemplo: considerar apenas uma parte de algum programa de cadastro de aluno
# ler e VALIDAR informações de apenas um aluno:
# RA (inteiro), IDADE(inteiro), NOTA (float) e NOME(string)

print('<< DADOS DE UM ALUNO >>')

RA = leia(" RA: ")

IDADE = leia(" IDADE:")

NOTA = leia(" NOTA: ", tipo = 'float')

NOME = input(" Nome: ")

print()
print('-'*30)
print(" << DADOS DO ALUNO >>")
print(f" >> RA: {RA}")
print(f" >> NOME: {NOME}")
print(f" >> IDADE: {IDADE}")
print(f" >> NOTA: {NOTA}")
print('-'*30)
```

```
<< DADOS DE UM ALUNO >>
RA: a
Erro: Digite número int
RA: 1234
IDADE:d
Erro: Digite número int
IDADE:20
NOTA: d
Erro: Digite número float
NOTA: 3,5
Erro: Digite número float
NOTA: 3.5
Nome: Ana Maria

-----

<< DADOS DO ALUNO >>
>> RA: 1234
>> NOME: Ana Maria
>> IDADE: 20
>> NOTA: 3.5
-----
```

Argumento nomeados: *tipo*

Obs.: caso não nomeássemos o argumento *tipo* → `NOTA = leia(" NOTA: ", 'float')`

seria interpretado como o 2º, parâmetro →  
como sendo a mensagem do erro  
*msgerro* = 'float'

```
<< DADOS DE UM ALUNO >>
RA: 1234
IDADE:18
NOTA: a
floatint
```

## Funções recursivas:

**Recursividade** é uma técnica de desenvolvimento de algoritmos – procedimentos e funções - que fazem chamadas de si mesmos.

A chamada recursiva é também conhecida de chamada interna.

Uma definição recursiva de um conjunto ou processo **deve** ter:

- uma redefinição explícita para pelo menos um parâmetro;
- pelo menos uma chamada dele mesmo;
- um critério de decisão para parada.

**Exemplo clássico:** cálculo do fatorial de um número

Definição:  $N! = N * (N-1)!$  e sabe-se que  $0! = 1! = 1$



## Exemplo – fatorial:

```
File Edit Format Run Options Window Help
''' função recursiva - fatorial '''

def fat (n):
    if n <=1: return 1
    return n* fat(n-1)

#principal

N = int (input (' Fatorial de: '))

print(f'Fatorial de {N}: {fat(N)}')
```

Uma definição recursiva de um conjunto ou processo **deve** ter:

- uma redefinição explícita para pelo menos um parâmetro;
- pelo menos uma chamada dele mesmo;
- um critério de decisão para parada.

```
Py
Fatorial de: 5
Fatorial de 5: 120
/// |
```

**Apenas** para observar os passos das chamadas – incluídos **prints** na função:

```
File Edit Format Run Options Window Help
''' função recursiva - fatorial '''

def fat (n):
    print(f' n = {n}')
    if n <=1: return 1
    print(f' return {n} * fat ({n-1})')
    return n* fat(n-1)

#principal

N = int (input (' Fatorial de: '))

print(f'Fatorial de {N}: {fat(N)}')
```

```
py
Fatorial de: 5
n = 5
return 5 * fat (4)
n = 4
return 4 * fat (3)
n = 3
return 3 * fat (2)
n = 2
return 2 * fat (1)
n = 1
Fatorial de 5: 120
>>> |
```

## VAMOS PRATICAR?

1. Elabore um programa que **leia dois números inteiros** e uma função que some todos os valores inteiros contidos no intervalo entre esses dois números. **Retorne** o resultado da soma pelo **return**.
2. Elabore um programa que leia um número inteiro e construa **duas funções**: uma que some os dígitos desse número inteiro e outra que determine o maior dígito desse número.

**Exemplo: número = 1063, então a soma =  $1 + 0 + 6 + 3 = 10$  e o maior dígito é 6**