# Fundamentos da Programação

## Aula 8

### Funções

Definição de funções. Aplicação de funções. Abstração procedimental.

Alberto Abad, IST, 2024-25

## Até agora aprendemos...

- O que é e quais são as fases da atividade de programação?
- A descrever sintaxe de uma linguagem de programação --> **BNF**
- Sobre o Python e como interagir com ele: interpretador vs. script
- Alguns elementos básicos de programação:
    - Tipos, nomes, expressões, condições, entrada/saída, funções embebidas, etc.
- Sobre execução sequencial e sobre instruções para alterar o fluxo de execução:
    - Seleção --> **IF**
    - Repetição --> **WHILE**

# Funções

- Conjunto pares ordenado entrada (domínio) e saída (contra-domínio)
    - Definição por extensão ou por abstração
    - Como utilizar funções? Definição e aplicação:

```
f(x, y) = x + y   Definições equivalentes?
f(3,5) = 8        Aplicação!
```

- Igual que na Matemática, a utilização de funções em programação compreende a **definição** da função (nome, argumentos e algoritmo) e **aplicação de função** (execução do algoritmo sobre valores passados como argumentos).
- Exemplo funções Python já conhecidas: `print(...)`, `input(...)`, `eval(...)`

# Definição de Funções (BNF)

```
<definição de função> ::=
    def <nome> (<parâmetros formais>): NEWLINE
    INDENT <corpo> DEDENT

<parâmetros formais> ::= <nada> | <nomes>

<nomes> ::= <nome> | <nome>, <nomes>

<nada> ::=

<corpo> ::= <definição de função>* <instruções em função>

<instruções em função> ::=
    <instrução em função> NEWLINE |
    <instrução em função> NEWLINE <instruções em função>

<instrução em função> ::= <instrução> | <expressão> | <instrução return>

<instrução return> ::= return | return <expressão>
```

# Aplicação Funções (BNF)

```
<aplicação de função> ::= <nome>(<parâmetros concretos>)
```

```
    <parâmetros concretos> ::= <nada> | <expressões>
```

```
    <expressões> ::= <expressão> |
                <expressão>, <expressões>
```

# Definição e Aplicação de Funções, Exemplo 1:

```
def soma(a,b):
    return a + b
```

Exemplos:

- Aplicação antes e após definição.
- Aplicação: `soma(2)`, `soma(2,5`, `soma(7,5)`, `soma(3*2, 6+4)`, `soma`
- O que acontece se:

```
x, y = 2, 5
print("x =",x, "y =",y, "soma(x,y) =",soma(x,y))
```

- O que acontece se:

```
a, b = 2, 5
print("soma(a,b) =", soma(a,b))
print("soma(b,a) = ", soma(b,a))
print("a =",a,"b =",b)
```

```
In [ ]:
```

# Definição de Aplicação Funções, Exemplo 2:

```
def soma_progressao_aritmetica(n):
    soma = 0
    pass # complete
```

- Aplicação:

```
soma = 4 + 6
print(soma_progressao_aritmetica(100), soma)
```

- Que acontece com a variável *soma*!?!?
- Conseguem pensar numa solução não iterativa?

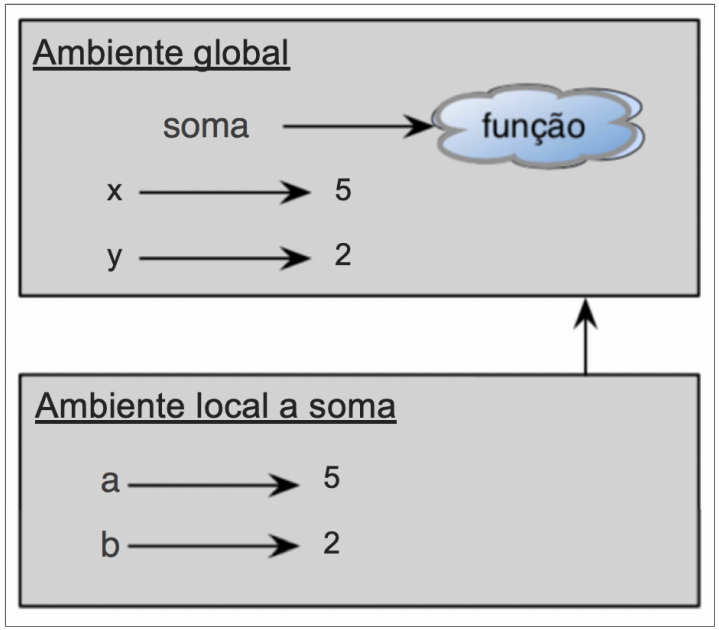
```
In [27]: n = 100
         def soma_progressao_aritmetica(n):
    soma = 0
    pass


res = soma_progressao_aritmetica(n)
print(res)
```

```
55
```

# Ambientes e quadros (*frames*)

- Ambientes: Global vs. Local
- Passos seguidos pelo Python quando uma função é invocada:
    - Os parâmetros concretos são avaliados (ordem arbitrária)
    - Os parâmetros formais da função são associados com os valores concretos no ambiente local (em ordem)
    - O corpo da função é executado no ambiente local (os ambientes locais existem só até a função terminar) e o valor de *return* é retornado ao ambiente global

# Ambientes e quadros (*frames*)




- Gostavam ver?

http://pythontutor.com/visualize.html

## Abstração procedimental

- As funções permitem aos programadores pensar no **que** (faz a função) e não no **como** (a função é implementada).

## Exemplo #1

```
def soma_prog_arit(n):
    iter = 1
    soma = 0
    while iter <= n:
        soma = soma + iter
        iter = iter + 1
    return soma
```

```
def soma_prog_arit(n):
    if n < 1:
        return 0
    else:
        return n*(n+1)//2
```

## Abstração procedimental

- As funções permitem aos programadores pensar no **que** (faz a função) e não no **como** (a função é implementada).

## Exemplo #2

```
def quadrado(x):
    return x * x
```

```
def quadrado(n):
    pass # complete
```

In [ ]:
```
def quadrado_v1(x):
    return x * x

def quadrado_v2(x): # **
    pass

def quadrado_v3(x): # pow
    pass

def quadrado_v4(x): # iter
    pass


print(quadrado_v1(9), quadrado_v2(9), quadrado_v3(9), quadrado_v4(9))
```

A treinar!!!!

# Exemplo 1, Tabela conversão temperaturas

- Escreva uma função que converta temperaturas de Fahrenheit para Celsius $C = 5 * (F - 32)/9$
- Escreva uma função que recebe uma temperatura mínima e máxima (inteiros) em Fahrenheit e imprime a tabela de conversão para Celsius

In [ ]:
```
def fahr_para_cent(fahr):
    pass

def tabela(min_f, max_f):
    pass

tabela(40, 50)
```

# Exemplo 2, Potência de dois números inteiros

- Solução iterativa para $k$ positivos
- E para $k$ negativos?
- E para qualquer $k$ ?

In [ ]:
```
def potencia(x, n):
    pass

# Power of two numbers inteiros
x = eval(input("Escreva a base da potencia: "))
n = eval(input("Escreva a potencia: "))


print(potencia(x, n))
```

## Exemplo 3, Factorial

- Para números inteiros não negativos ( `fact(0) = 1` ) iterativo

In [ ]:
```
def factorial(x):
    pass


x = eval(input('Inteiro: '))
f = factorial(x)
print(f)
```

# Funções - Tarefas próxima aula

- Trabalhar matéria apresentada hoje
- Fazer todos os exercícios/programas