# Fundamentos da Programação

## Funções

Definição de funções. Aplicação de funções. Abstração procedimental. Exemplos simples.

## Aula 5

José Monteiro

(slides adaptados do Prof. Alberto Abad)

## Na semana passada 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 envolve a **definição** da função (nome, argumentos e algoritmo) e **aplicação de função** (execução do algoritmo sobre valores passados como argumentos).
- Exemplo de 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 de 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):
    res = a + b
    return res
```

- Aplicação antes e após definição.
- Aplicação: `soma(2)` , `soma(2,5)` , `soma(7,5)` , `soma(3*2, 6+4)` , `soma`
- ```
  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 e Aplicação de Funções

## Exemplo 2:

```
def soma_progressao_aritmetica(n):
    soma = 0
    it = 1
    while it <= n:
        soma = soma + it
        it = it + 1
    return soma
```

- Aplicação:

  ```
  soma = 10
  print(soma_progressao_aritmetica(100), soma)
  ```

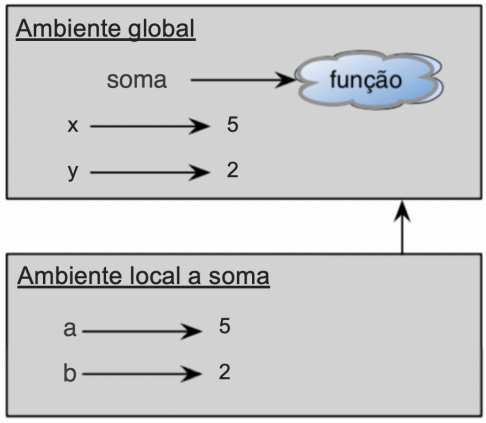
- Que acontece com a variável *soma*!?!?
- Conseguem pensar numa solução não iterativa?

In [ ]:
  ```

# 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 só existem até a função terminar) e o valor de *return* é retornado ao ambiente global

# Ambientes e Quadros (*frames*)




- Querem 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
  ```

In [ ]:

# Abstração Procedimental

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

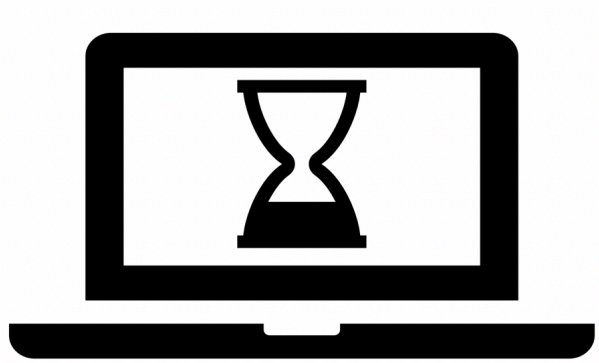
### Exemplo 2

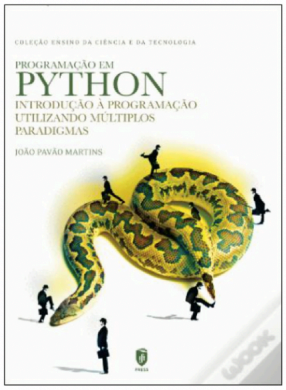
```
def quadrado(x):
    return x * x
def quadrado(x):
    return x ** 2
def quadrado(x):
    return pow(x, 2)
def quadrado(x):
    res, i = 0, 0
    while i < x:
        res = res + x
        i += 1
    return res
```

In [ ]:

# Funções - Tarefas próxima aula

- Trabalhar matéria apresentada hoje
- Estudar Cap 3 do livro
- Tentar fazer/pensar nos exercícios/programas da próxima aula






## Para 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

### Exemplo 2: Potência de dois números inteiros

### Exemplo 3: Fatorial

## Funções

### 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 [4]:

```
def fahr_para_cent(fahr):
    return (5*(fahr-32))/9

def tabela(min_f, max_f):
    if not (type(min_f) == int and isinstance(max_f, int) and min_f < max_
        raise ValueError('argumentos invalidos')

    while min_f <= max_f:
        print("fahrheneit=", min_f, "e celsius=", fahr_para_cent(min_f))
        min_f=min_f+1

# min_f = eval(input('Introduzir temp mínima'))
# max_f = eval(input('Introduzir temp máxima'))
tabela(40, 41)
```

```
fahrheneit= 40 e celsius= 4.444444444444445
fahrheneit= 41 e celsius= 5.0
```

# Funções

## Exemplo 2, Potência de dois números inteiros

In [ ]:

```
def potencia_aux(x, n):
    if not ((isinstance(x,float) or isinstance(x,int)) and isinstance(n, i
        raise ValueError('argumentos invalidos')
    pot = 1
    while n != 0:
        pot = pot*x
        n = n - 1

    return pot


def potencia(x, n):
    if not ((isinstance(x,float) or isinstance(x,int)) and isinstance(n, i
        raise ValueError('argumentos invalidos')

    if n >= 0:
        return potencia_aux(x,n)
    else:
        return 1/potencia_aux(x,-n)


def potencia_v2(x, n):
    if not ((isinstance(x,float) or isinstance(x,int)) and isinstance(n, i
        raise ValueError('argumentos invalidos')

    pot = 1
    while n != 0:
        if n > 0:
            pot = pot*x
            n = n - 1
        else:
            pot = pot/x
            n = n + 1

    return pot


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


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

- Solução iterativa para `k` positivos
- E para `k` negativos?

# Funções

## Exemplo 3, Fatorial

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

In [ ]:

```
def factorial(x):
    if not (isinstance(x, int) and x >= 0):
        raise ValueError('invalido')

    fat = 1

    while x >= 1:
        fat = fat * x
        x = x - 1

    return fat

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