centro universitário FEI

# CC8210 – NCA210 Programação Avançada I

Prof. Reinaldo A. C. Bianchi
Prof. Isaac Jesus da Silva
Prof. Danilo H. Perico

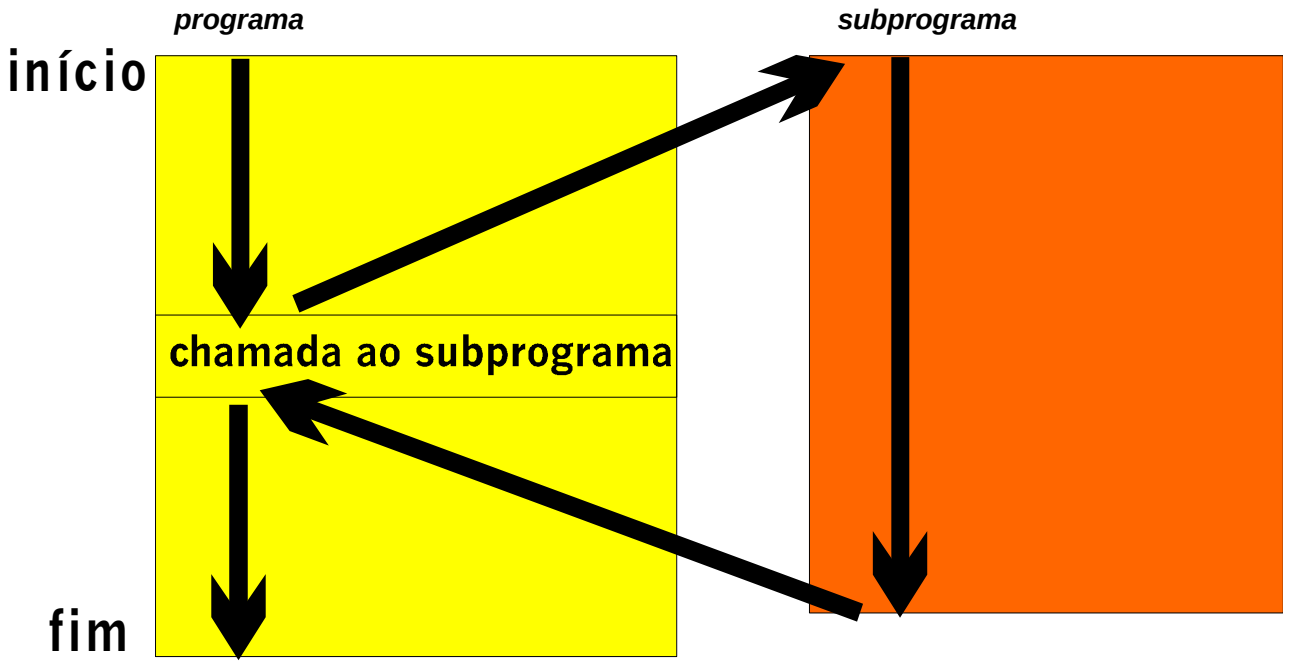
# Modularização: Funções

# Funções

- Funções **são blocos de código** que realizam **determinadas tarefas** que normalmente precisam ser executadas diversas vezes dentro da mesma aplicação

- Assim, **tarefas muito utilizadas costumam ser agrupadas em funções**, que, depois de definidas, podem ser utilizadas / chamadas em qualquer parte do código somente pelo seu nome

# Subprogramas: características fundamentais



# Definindo funções em Python - **def**

- Podemos criar / definir nossas próprias funções no Python utilizando a palavra-chave def seguido do nome da função, parêntesis ( ) e “:”
- Sintaxe:

```
def <nome da função>():
    # tarefas que serão realizadas dentro da função
```

- Exemplo:

```
def imprimeOla():
    print("Olá")
```

# Funções **com** e **sem** parâmetros

- As funções podem ou não ter **parâmetros**, que **são valores enviados às funções** dentro dos parêntesis no momento em que elas são chamadas
- Exemplos:

**Sem parâmetros:**

```
def soma():
    a = 9
    b = 8
    print(a+b)

soma()
```

```
17
```

Chamada da função

**Com parâmetros:**

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

soma(3,4)
```

```
7
```

Chamada da função

# Funções - *return*

- Além dos parâmetros, as funções podem ou não ter um **valor de retorno**
- O retorno é definido pela palavra-chave ***return***
- Exemplos:

**Sem parâmetros:**

```
def soma():
    a = 9
    b = 8
    return(a+b)

print(soma())
```

```
17
```

**Com parâmetros:**

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

print(soma(3,4))
```

```
7
```

# Funções

- Exemplo: Fazer uma função que retorne ***True*** ou ***False*** para a verificação de números pares.
  - Precisa de parâmetros? Sim ou Não?
  - É melhor usar ou não o ***return***?

# Funções

- Exemplo: Fazer uma função que retorne ***True*** ou ***False*** para a verificação de números pares.

```
def par(num):
    return(num % 2 == 0)

print(par(3))
print(par(4))
print(par(67))
```

```
False
True
False
```

# Funções

- Exemplo:
- Se precisarmos de uma função que retorne a string "*par*" ou "*ímpar*", podemos reutilizar a função par e criar uma função nova:

```
def par(num):
    return(num % 2 == 0)

def parOuImpar(x):
    if par(x) == True:
        return "par!"
    else:
        return "ímpar!"

print(parOuImpar(4))
print(parOuImpar(3))
print(parOuImpar(56))
```

```
par!
ímpar!
par!
```

# Funções

- A grande diferença de Python para outras linguagens de programação:
- Não é necessário definir os tipos dos parâmetros nem do que a função retorna!
- Exemplo em C:

```
int quadrado (int x)
{
    return (x * x);
}
```

# PythonTutor.com - passagem por valor

```
string = "Geeks"
def test(string):
    string = "GeeksforGeeks"
    print("Inside Function:", string)
# Driver's code
test(string)
print("Outside Function:", string)
```

# PythonTutor.com - passagem por referência

```
def add_more(list):
    list.append(50)
    print("Inside Function", list)
# Driver's code
mylist = [10,20,30,40]
add_more(mylist)
print("Outside Function:", mylist)
```

# Funções - escopo das variáveis: **locais** *vs.* **globais**

- Quando usamos funções, trabalhamos com **variáveis internas**, que **pertencem somente à função**.
- Estas variáveis internas são chamadas ***variáveis locais***
- **Não** podemos acessar os valores das variáveis locais fora da função a que elas pertencem
- É por isso que passamos parâmetros e retornamos valores das funções. Os **parâmetros** e o ***return*** possibilitam a troca de dados no programa

# Funções - escopo das variáveis: locais *vs.* globais

- Por sua vez, as ***variáveis globais*** são **definidas fora das funções** e podem ser vistas e acessadas por todas as funções e pelo "*código principal*" (que não está dentro de uma função específica)

- Exemplo:

```
# variável global:
a = 5

def alteraValor():
    # variável local da função alteraValor():
    a = 7
    print("Dentro da função 'a' vale: ", a)

print("'a' antes da chamada da função: ", a)
alteraValor()
print("'a' depois da chamada da função", a)
```

```
'a' antes da chamada da função:  5
Dentro da função 'a' vale:  7
'a' depois da chamada da função 5
```

# Escopo das variáveis: locais *vs.* globais

- Se quisermos modificar a variável global dentro da função, devemos utilizar a palavra-chave ***global***

```
# variável global:
a = 5

def alteraValor():
    # dizemos para a função que a variável 'a' é global:
    global a
    a = 7
    print("Dentro da função 'a' vale: ", a)

print("'a' antes da chamada da função: ", a)
alteraValor()
print("'a' depois da chamada da função", a)
```

```
'a' antes da chamada da função:  5
Dentro da função 'a' vale:  7
'a' depois da chamada da função 7
```

# Execute no PythonTutor.com os dois exemplos

```
# variável global:
a = 5

def alteraValor():
    # variável local da função alteraValor():
    a = 7
    print("Dentro da função 'a' vale: ", a)

print("'a' antes da chamada da função: ", a)
alteraValor()
print("'a' depois da chamada da função", a)
```

```
'a' antes da chamada da função:  5
Dentro da função 'a' vale:  7
'a' depois da chamada da função 5
```

```
# variável global:
a = 5

def alteraValor():
    # dizemos para a função que a variável 'a' é global:
    global a
    a = 7
    print("Dentro da função 'a' vale: ", a)

print("'a' antes da chamada da função: ", a)
alteraValor()
print("'a' depois da chamada da função", a)
```

```
'a' antes da chamada da função:  5
Dentro da função 'a' vale:  7
'a' depois da chamada da função 7
```

# Funções - parâmetros opcionais

- Podemos ainda criar funções que podem ou não receber argumentos.
- Exemplo:

```
def soma(a=1, b=1):
    print(a+b)

soma()
soma(2,3)
```

```
2
5
```

Chamada sem argumento: Neste caso, *soma* assume valores 1 para ***a*** e ***b***

Chamada com argumentos: Neste caso, *soma* assume valores 2 para ***a*** e 3 para ***b***

# Funções Lambda

- No Python, podemos criar funções simples em somente uma linha
- Estas funções, ou expressões, são chamadas de lambda
- Exemplo: função lambda que recebe um parâmetro x e retorna o quadrado deste número.
  - lambda cria uma função a

```
a = lambda x: x**2
print(a(3))
```

```
9
```

# Funções Lambda

- Exemplo:
- Função lambda que calcula o aumento, dado o valor inicial e a porcentagem de aumento:

```
aumento = lambda a,b : a*b/100
aumento(100,5)
```

```
5.0
```

# Funções Recursivas

- Uma função recursiva é uma função que se refere a si própria.
  - A ideia consiste em utilizar a própria função que estamos a definir na sua definição.
- A execução de uma função recursiva consiste em ir resolvendo subproblemas sucessivamente mais simples até se atingir o caso mais simples de todos, cujo resultado é imediato.

http://www.dca.fee.unicamp.br/courses/EA072/lisp9596/node17.html

# Funções Recursivas

- Em todas as funções recursivas existe:
  - Um passo básico (ou mais) cujo resultado é imediatamente conhecido.
  - Um passo recursivo em que se tenta resolver um sub-problema do problema inicial.
- Geralmente, uma função recursiva só funciona se tiver uma expressão condicional.

http://www.dca.fee.unicamp.br/courses/EA072/lisp9596/node17.html

# Funções Recursivas

- Desta forma, o padrão mais comum para escrever uma função recursiva é:
  - Começar por testar os casos mais simples.
  - Fazer chamada (ou chamadas) recursivas com subproblemas cada vez mais próximos dos casos mais simples.
- Como exemplo clássico, podemos citar a Função Fatorial, que pode ser declarada como a seguir:
  - x! = x * (x-1) * (x-2) * ... * (3) * (2) * (1)

http://www.dca.fee.unicamp.br/courses/EA072/lisp9596/node17.html

# Fatorial Recursivo em Python

```
def fatorial(n):
    if n == 0 or n == 1:
        return 1
    else:
        return n * fatorial(n-1)

print(fatorial (5))
```

# Modularização:
# Funções - Exercícios

# Exercício 01

a) Escreva uma função com parâmetros que receba a base e a altura de um triângulo e retorne sua área (A = base * altura / 2).

b) Escreva uma função lambda que receba a base e a altura de um triângulo e retorne sua área (A = base * altura / 2).

# Exercício 02

a) Escreva uma função chamada par que receba um número e retorne True se o número é par ou False caso contrário.

b) Escreva uma função lambda chamada par que receba um número e retorne True se o número é par ou False caso contrário.

# Exercício 03

Escreva uma função que receba quatro parametros referente a notas de atividades do aluno e retorne a media dessa 4 notas, caso algumas notas não sejam atribuidas na chamada da função, atribuir como padrão o valor zero para essa notas.

# Exercício 04

Escreva uma função que receba uma lista de números e retorne o maior e o menor número dessa lista.

# Exercício 05

Escreva uma função que receba um número inteiro e retorne o valor desse número em binário e em hexadecimal.

# Exercício 06

Escreva uma função que receba uma lista e remova todos os valores duplicados e retorne a lista sem elementos duplicados. Porém a função não deve alterar a lista que recebeu como parâmetro.

# Exercício 07

Escreva uma função que receba uma lista de números inteiros e retorne duas listas, uma com os números pares e outra com os números impares.

# Exercício 08

Escreva uma função que receba duas listas de números inteiros e retorne uma lista seja a intersecção dessas duas listas.
Obs.: Intersecção é quando um valor estiver nas duas lista. Garanta que não haja elementos duplicados em cada uma dessas duas listas (use a função do exercício 06 para eliminar os valores duplicados).

# Exercício 09

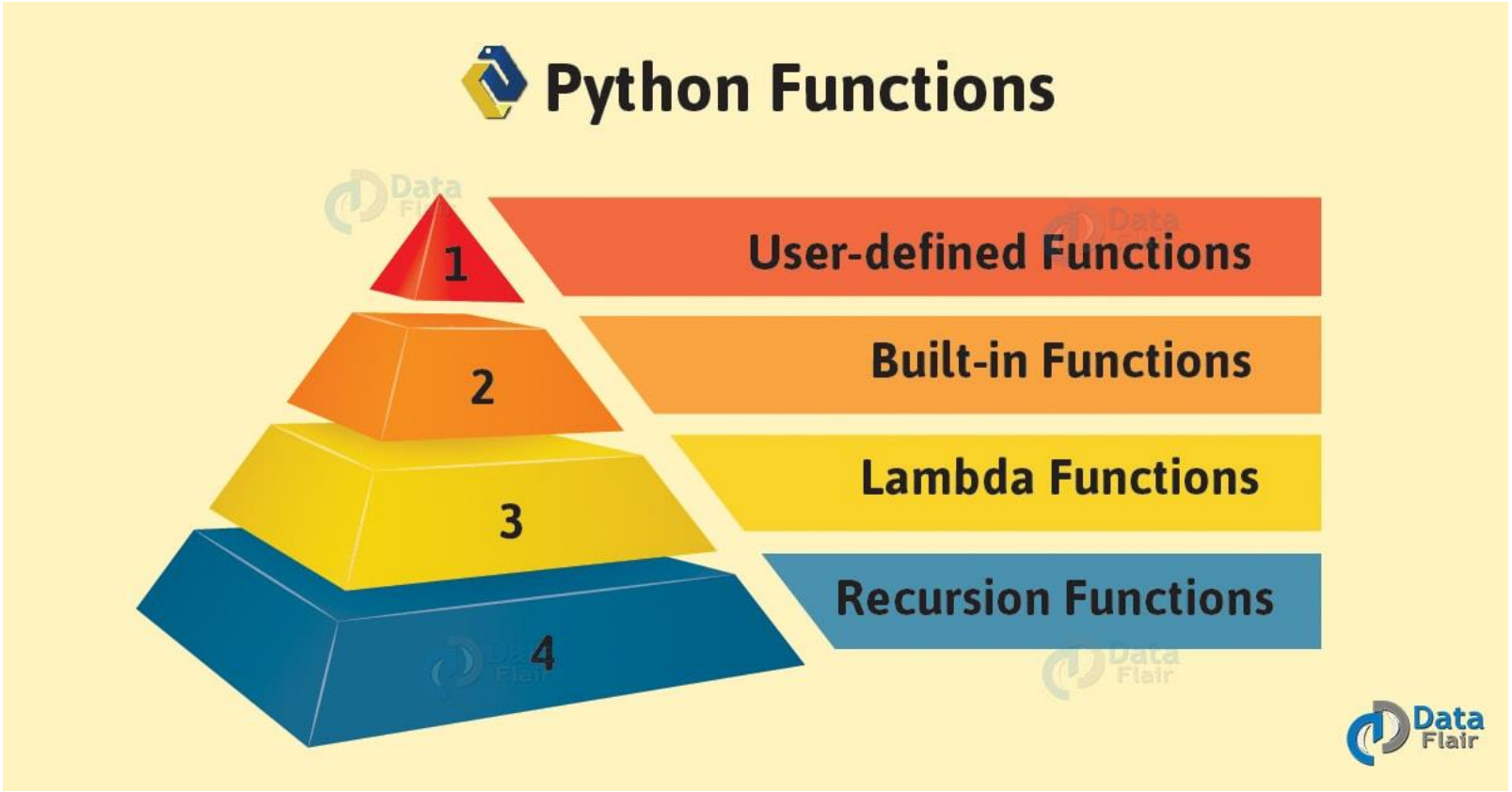
Newton descobriu um método para aproximar os valores das raízes de uma equação numérica. Esse método é bastante simples e é conhecido como método de Newton.

Escreva uma função que implemente o método de Newton para calcular e exibir a raiz quadrada de um número x digitado pelo usuário.

O algoritmo (pseudocódigo) para o método de Newton é o seguinte:

**Receba** *x*
**Inicialize** a variável *palpite* para ser igual a x/2
**Inicialize** a variável *erro* para ser igual a |$palpite^2 - x$ |
**Enquanto** *erro* > $10^{-12}$ faça
    **Atualize** *palpite* para ser igual a média entre *palpite* e *x/palpite*
    **Atualize** *erro* para ser igual a diferença entre $palpite^2$ e *x*
**Retorne** *palpite*

# Fim


Python Functions
1
User-defined Functions
2
Built-in Functions
3
Lambda Functions
4
Recursion Functions
Data Flair