UNIP
UNIVERSIDADE PAULISTA

python

IPE – Introdução a Programação Estruturada

Aula 07 – Funções

Profº Ms Gustavo Molina

Funções em Python

- A linguagem Python nos oferece uma sub-rotina para criar programas modulares e estruturados que são as **funções**. Função é um bloco de programa (trechos de código) contendo inicio e fim, identificado por um nome, através da qual será referenciado em qualquer parte do programa principal ou da sub-rotina chamadora.
- Funções são pequenos trechos de códigos reutilizáveis. Elas permitem dar um nome a um bloco de comandos e rodar esse bloco usando o nome a partir de qualquer lugar do programa, quantas vezes você quiser. Isto é conhecido como “chamar” a função. Já usamos algumas funções internas do Python, tais com *len* e *range*.
- As funções são definidas usando a palavra **def.** Esta é seguida pelo nome da função, um par de parênteses que pode envolver algumas variáveis , dois pontos e o bloco de comandos.

Funções em Python

Função_01.py

E: > UNIP > 2020 > 2º Semestre > IPE > Aula 7 > Função_01.py > ...

```
1   #definindo a função
2   def unip():
3       print("Introdução a Programação Estruturada")
4
5   #chamando a função
6
7   unip()
```

Funções em Python

- A função **unip** apresentada não possui nenhum parâmetro. Parâmetros são as variáveis que podem ser incluídas nos parênteses da definição. O bloco de comandos da função pode utilizar essas variáveis para processar a informação e executar algum trabalho útil. Quando a função é chamada, são passados valores a serem atribuídos as variáveis e esses valores são chamados argumentos.

Função_02.py

E: > UNIP > 2020 > 2º Semestre > IPE > Aula 7 > Função_02.py > ...

```
1   #definindo a função
2   def subtracao(a,b): #a e b são os parâmetros da função
3       print("A subtração é: ",a-b)
4
5   #chamando a função subtração
6   subtracao(5,2) #5 e 2 são argumentos da função.
```

Exemplo 1

Exemplo 1:

Deve ser criado um programa que, por intermédio de uma sub-rotina do tipo função, efetue a leitura de dois valores inteiros e apresente como saída uma mensagem informando se os números são iguais ou diferentes.

Exemplo_01.py

E: > UNIP > 2020 > 2º Semestre > IPE > Aula 7 > Exemplo_01.py > ...

```
1  def compara(x,y):
2      if x == y:
3          print("Números Iguais: ")
4      else:
5          print("Números Diferentes: ")
6
7  num1 = input("Digite um valor: ")
8  num2 = input("Digite outro valor: ")
9  compara(num1,num2)
```

Exemplo 2

Exemplo 2:

Faça um programa para ler um número inteiro e chamar uma função que retorna uma mensagem, dizendo se o número lido é: Par, Ímpar ou Zero. Mostre o retorno da função.

Exemplo_02.py

E: > UNIP > 2020 > 2º Semestre > IPE > Aula 7 > Exemplo_02.py > ...

```
1   def compara(x):
2       if x == 0:
3           print("Zero")
4       elif x % 2 == 0:
5           print("Par")
6       else:
7           print("Impar")
8
9   n1 = int(input("Digite um valor: "))
10  compara(n1)
```

Variáveis Locais

Quando você cria uma variável dentro de uma função, ela é totalmente independente de qualquer outra variável que tenha o mesmo nome fora da função, isto é, os nomes das variáveis são locais para a função. Isto é chamado de escopo da variável. Todas as variáveis têm o escopo do bloco de comandos em que foram criadas.

Variavel_local.py

E: > UNIP > 2020 > 2º Semestre > IPE > Aula 7 > Variavel_local.py > ...

```
1   def func (x):
2       print("x é: ",x)
3       x=2
4       print ("Valor local alterado de x para: ",x)
5   x=50
6   func(x)
7   print("x ainda é: ",x)
```

Variáveis Locais

Na função, quando usamos o valor de *x* pela primeira vez, o Python usa o valor passado como argumento, no caso 50.

Depois, atribuímos o valor 2 a *x*. O nome *x* é local para a nossa função. Assim, quando alteramos o valor de *x* na função, o *x* definido no bloco principal permanece inalterado. A última instrução print confirma isto.

Variavel_local.py

E: > UNIP > 2020 > 2º Semestre > IPE > Aula 7 > Variavel_local.py > ...

```
1  def func (x):
2      print("x é: ",x)
3      x=2
4      print ("Valor local alterado de x para: ",x)
5  x=50
6  func(x)
7  print("x ainda é: ",x)
```

Variáveis Globais

Se você quiser atribuir um valor a uma variável definida fora da função, terá que informar ao Python que a variável não é local, mas sim global. Fazemos isto usando a instrução **global**. É impossível atribuir um valor a uma variável fora da função sem utilizar a instrução global.

Você pode usar o valor das variáveis definidas fora da função (considerando que não existem variáveis com o mesmo nome dentro da função). Porém, isto é desencorajado e deve ser evitado, uma vez que se torna difícil para o leitor saber onde a definição da variável está. Usar a instrução global torna claro que a variável foi definida fora da função.

Variáveis Globais

Variavel_Global.py

E: > UNIP > 2020 > 2º Semestre > IPE > Aula 7 > Variavel_Global.py > ...

```
1  def func ():
2      global X
3      X = 19 # Aqui está sendo alterado o objeto X global
4      Y = X * 2
5      print("X global existe dentro da função: valor = {0}".format(X))
6      print("Y local existe dentro da função: valor = {0}".format(Y))
7
8  print("Inicio do Programa")
9  X = 10
10 print("X global existe fora da função: valor = {0}".format(X))
11 func()
12 print("X global alterado na função: valor = {0}".format(X))
13 print ("Fim do programa")
```

Retornando Valores

Nos exemplos descritos anteriormente, as funções não retornam valor algum, apenas exibem informação. Para retornar um valor, basta usar a expressão return dentro da função.

Return.py

E: > UNIP > 2020 > 2º Semestre > IPE > Aula 7 > Return.py > ...

```
1  def maior(x,y):
2      if x>y:
3          return x
4      else:
5          return y
6
7  print (maior (2,3))
```

Dúvidas???