Programação Script

Funções (parte 1)

Aula 05

Prof. Felipe A. Louza

Roteiro

Roteiro

Funções

Uma função corresponde a um bloco de comandos desenvolvido para uma tarefa específica.

- Em **Python** existem algumas funções prontas:

```
1 input()
```

```
1 print()
```

```
1 x = input("Qual o seu nome?")
2
3 print(x)
```

Funções

Outro exemplo:

- Função que calcula x^y:

```
1 pow(x,y)
```

```
1 import math
2
3 x = int(input())
4 math.pow(x,2)
```

- Em geral, uma função possui:
  1. parâmetros de entrada, e
  2. um valor de retorno.

Funções

Mais funções do módulo `math`:

`math.cos(x)`	retorna o cosene de x
`math.sin(x)`	retorna o seno de x
`math.tan(x)`	retorna o tangente de x
`math.sqrt(x)`	retorna a raiz quadrada de x
`math.pow(x, y)`	retorna the value de x to the power de y
`math.fabs(x)`	retorna o valor absoluto de x
`math.factorial(x)`	retorna o fatorial de x
`math.isfinite(x)`	verifica se x é um número finito
`math.log(x, b)`	retorna o logaritmo de x na base b
`math.log10(x)`	retorna o logaritmo de x na base 10
`math.log2(x)`	retorna o logaritmo de x na base 2
`math.gcd(x,y)`	retorna o máximo divisor comum de x e y

Definindo uma função

Podemos criar as nossas próprias funções em **Python**:

- Uma função é definida da seguinte forma:

```
1 def nome(parâmetro1, parâmetro2,..., parâmetroN):
2     comando1
3     comando2
4     ...
5     comandoM
6     return valor
```

- Cada parâmetro corresponde à uma variável na função, que é inicializada com valor indicado durante a chamada da função.

```
1 nome(x, 'x',..., x+2):
```

- O comando `return` devolve o resultado em valor.

Definindo uma função

Exemplo:

- A função abaixo recebe dois valores inteiros `a` e `b` e retorna `a+b`.

```
1 def soma(a, b):
2   c = a + b
3   return c
```

  - Importante: sempre que o comando `return` é executado, a função para de executar e retorna o valor indicado para quem fez a chamada da função.

Definindo uma função

Podemos chamar a função `soma()` quantas vezes for necessário.

```
1 def soma(a, b):
2     c = a + b
3     return c
```

- Para isso, basta passar os parâmetros que serão associados com as variáveis `a` e `b`, respectivamente.

```
1 r = soma(12, 10)
```

  - Podemos passar valores (cópias) de variáveis como parâmetros:

```
1 a = 10
2 b = 5
3 r = soma(a, b+2)
```

Definindo uma função

Podemos definir funções ao longo de um programa:

- A execução do programa inicia após as definições de funções.

```
1 def soma(a, b):
2   c = a + b
3   return c
4
5 x = int(input())
6 y = int(input())
7 r = soma(x, y)
8 print("r = ", r)
```

- Ao encontrar a chamada para a função, o fluxo de execução muda para o inicio da função, e continua até encontrar um `return`.
- Depois, o fluxo de execução volta para o ponto onde a chamada da função ocorreu.

Fluxo de execução

Nada após o `return` será executado.

```
1 def soma(a, b):
2   c = a + b
3   return c
4   print("Bla bla bla!")
5
6 x = int(input())
7 y = int(input())
8 r = soma(x, y)
9 print("r = ", r)
```

- A expressão contida dentro do comando `return` é chamado de valor de retorno (é a resposta da função).

Fluxo de execução

Uma função pode ter mais de um comando `return`.

```
1  def diff(a, b):
2    c = a - b
3    if(c<0):
4      return (-1)*c;
5    else:
6      return c
7
8  x = int(input())
9  y = int(input())
10 r = diff(x, y)
11 print("r = ", r)
```

- Apenas um comando `return` será executado.

Definindo uma função

Podemos utilizar o resultado de uma função diretamente em uma expressão:

```
1 def soma(a, b):
2   c = a + b
3   return c
```

```
1 print("r =", soma(a, b))
```

Definindo uma função

O que acontece se tentarmos chamar uma função com o número errado de parâmetros?

```
1 r = soma(12)
2 r = soma(-9, 11, 1)
```

- Ocorre um erro ao executarmos o programa!

Porque utilizar funções?

Porque utilizar funções?

- Separar o programa em partes que possam ser compreendidas de forma isolada $\leftarrow$ modularização
- Permitir o reaproveitamento de código já construído (por você ou por outros programadores).
- Evitar que um trecho de código seja repetido várias vezes dentro de um mesmo programa, minimizando erros e facilitando alterações.

Roteiro

Definindo funções depois do seu uso

O que está errado no programa abaixo? Como corrigir?

```
1 x1 = int(input())
2 x2 = int(input())
3 res = soma(x1, x2
4 print("Soma é: ", res)
5
6 def soma(a, b):
7   c = a + b
8   return c
```

- Ocorre um erro ao executarmos o programa!

```
1 Traceback (most recent call last):
2   File "soma.py", line 2, in <module>
3     x1 = leNumero()
4 NameError: name 'leNumero' is not defined
```

Definindo funções depois do seu uso

Até o momento, aprendemos a definir as funções antes do seu uso.

```
1 def soma(a, b):
2   c = a + b
3   return c
4
5 x1 = int(input())
6 x2 = int(input())
7 res = soma(x1, x2)
8
9 print("Soma é: ", res)
```

Definindo funções depois do seu uso

É comum criarmos um função `main()` como ponto de partida de um programa.

```
1  import bibliotecas
2
3  def main():
4    Comandos Iniciais
5
6  def fun1(Parâmetros):
7    Comandos
8
9  def fun2(Parâmetros):
10   Comandos
11
12 ...
13 ...
14 main()
```

- O seu programa conterá então várias funções (incluindo a `main()`) e um único comando no final do arquivo que é a chamada da função `main()`.

Definindo funções depois do seu uso

Exemplo:

```
1  def main():
2    x1 = int(input())
3    x2 = int(input())
4    res = soma(x1, x2)
5    print("Soma é: ", res)
6
7  def soma(a, b):
8    c = a + b
9    return c
10
11 main()
```

- Agora a execução do programa ocorre sem problemas.

Definindo funções depois do seu uso

Outro exemplo:

```
1  def main():
2    a = int(input())
3    b = int(input())
4    c = int(input())
5
6    print("O menor valor é:", menor(a,b,c))
7
8  def menor(a, b, c):
9    min = a
10   if (b < min): min = b
11   if (c < min): min = c
12   return min
13
14 main()
```

Roteiro

Funções Podem Invocar Funções

Qualquer função pode chamar outra função:

```
1  def funcao1():
2    #cmd C
3    funcao2();
4    #cmd K
5
6    ...
7
8  def funcao2():
9    #cmd D
10   funcao3();
11   #cmd J
```

```
1  def funcao3():
2    #cmd E
3    funcao4()
4    #cmd I
5
6    ...
7
8  def funcao4():
9    #cmd F
10   #cmd G
11   #cmd H
```

```
1  def main():
2    # cmd A
3    # cmd B
4    funcao1()
5
6  main()
```

Funções Podem Invocar Funções

No exemplo a seguir `fun1()` chama `fun2()`.

```
1  def main():
2    c = 5
3    c = fun1(c)
4    print("c =", c)
5
6  def fun1(a):
7    a = a + 1
8    a = fun2(a)
9    return a
10
11 def fun2(b):
12   b = 2*b
13   return b
14
15 main()
```

- O que será impresso?

Funções Podem Invocar Funções

No próximo exemplo, `fun1()` chama `fun2()`, que chama `fun3()`.

```
1  def main():
2    c = 5
3    c = fun1(c)
4    print("c =", c)
5
6  def fun1(a):
7    a = a + 1
8    a = fun2(a)
9    return a
10
11 def fun2(b):
12   b = 2*b
13   b = fun3(b)
14   return b
15
16 def fun3(c):
17   c = c**2
18   return c
19
20 main()
```

- O que será impresso?

Funções Podem Invocar Funções

Funções também podem chamar elas mesmas.

```
1  def main():
2    c = 5
3    c = fun1(c)
4    print("c =", c)
5
6  def fun1(a):
7    a = a + 1
8    a = fun1(a)
9    return a
10
11 main()
```

- Cuidado!!
- É preciso definir um critério de parada

Funções Podem Invocar Funções

Agora `fun1()` invoca `fun1()` apenas enquanto `a < 10`:

```
1  def main():
2    c = 5
3    c = fun1(c)
4    print("c =", c)
5
6  def fun1(a):
7    a = a + 1
8    if(a<10):
9      a = fun1(a)
10   return a
11
12 main()
```

- O que será impresso?
- Veremos melhor o conceito de recursão mais tarde no curso.

Roteiro

Definindo uma função

A lista de parâmetros de uma função pode ser vazia:

```
1 def leNumeroInt():
2   c = input("Digite um número inteiro: ")
3   return int(c)
4
5 r = leNumeroInt()
6 print("Número digitado: ", r)
```

Mais sobre funções

Podemos definir valores *default* para cada parâmetro da função:

```
1 def soma(a=1, b=3):
2   c = a + b
3   return c
4
5 soma(5, 2)
6 soma(5)
7 soma()
```

- Quando nenhum valor é passado ao parâmetro, a variável correspondente recebe o valor *default*.

Mais sobre funções

Outro exemplo:

```
1  def fun(a=1, b=2, c=3):
2      return a * b * c
3
4  print(fun())
5
6  print(fun(2))
7
8  print(fun(2, 3))
9
10 print(fun(2, 3, 4))
```

Mais sobre funções

Podemos estabelecer qual valor será passado para cada parâmetro da função:

```
1 def soma(a, b):
2   c = a + b
3   return c
4
5 soma(b=5, a=2)
```

- Podemos passar os parâmetros fora de ordem dessa forma.

Mais sobre funções

Outro exemplo:

```
1 def my_function(child3, child2, child1):
2   print("The youngest child is " + child3)
3
4 my_function(child1 = "Emil", child2 = "Tobias", child3 = "Linus")
```

Mais sobre funções

Agora podemos chamar a função com apenas o valor $b = 5$:

```
1 def soma(a=1, b=3):
2   c = a + b
3   return c
4
5 soma(b=5)
```

Mais sobre funções

Funções podem não retornar nada.

```
1 def imprime(num):
2   print("Número: ", num)
3   return None
```

- `None` é um objeto em **Python** que representa o "nada".
- Outra opção é não escrever o comando `return`.

```
1 def imprime(num):
2   print("Número: ", num)
```

Mais sobre funções

No **Python** o bloco de comandos de uma função não pode ser vazio:

```
1 def myfunction():
2
3 a = 1
4 b = a + 1
5 ...
```

- Se você quiser criar uma função vazia (por alguma razão), deve usar o comando `pass`

```
1 def myfunction():
2   pass
```

Roteiro

Exemplo 1

Escreva uma função que calcule x^2:

```
>>> quadrado(8)
64
```

Exemplo 1

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

```
1 x = int(input())
2 print(quadrado(x))
```

- Como podemos validar `quadrado(x)`??

Exemplo 1

Podemos verificar `quadrado(x)` para alguns valores de $x = 1, 2, \ldots, 100$.

```
1  def quadrado(x):
2    return x*x
3
4  def verifica():
5    i = 1
6    while(i <= 100):
7      if(quadrado(i)!=(i**2)):
8        print("Erro!")
9      i = i + 1
10
11 verifica()
```

Exemplo 1

O comando `assert`:

```
1  def quadrado(x):
2      if(x == 50): return x
3      return x*x
4
5  def verifica():
6      i = 0
7      while(i <= 100):
8          assert quadrado(i) == (i**2), "Erro quando i = {}!".format(i)
9          i = i+1
10
11 verifica()
```

Exemplo 2

Escreva uma função que calcule x^y:

```
>>> potencia(2, 6)
64
```

Exemplo 2

```
1 def potencia(x, y):
2   res = x
3   i = 1
4   while(i < y):
5     res = res*x
6     i = i+1
7   return res
```

```
1 x = int(input())
2 y = int(input())
3
4 print(potencia(x,y))
```

- Como podemos validar `potencia(x,y)`??

Exemplo 2

Podemos verificar `potencia(x,y)` para $x = 2$ e $y \in [0, 10]$.

```
1  import math
2
3  def verifica():
4    x = 2
5    y = 0
6    while(y <= 0):
7      assert potencia(x,y) == math.pow(x,y), "Erro!"
8      y = y+1
9
10 def potencia(x, y):
11   res = x
12   i = 1
13   while(i < y):
14     res = res*x
15     i = i+1
16   return res
17
18 verifica()
```

- Qual é o erro?

Exemplo 2

Vamos melhorar a saida do comando `assert`.

```
1  import math
2
3  def verifica():
4    x = 2
5    y = 0
6    while(y <= 0):
7      assert potencia(x,y) == math.pow(x,y), \
8                          "Erro a validar x = {} e y = {}".format(x,y)
9      y = y+1
10
11 def potencia(x, y):
12   res = x
13   i = 1
14   while(i < y):
15     res = res*x
16     i = i+1
17   return res
18
19 verifica()
```

- Qual é o resultado de `potencia(2,0)` ??

Exemplo 2

Corrigindo a função potencia para quando $y = 0$, retornar 1:

```
1 def potencia(x, y):
2   if(y==0):
3     return 1
4   res = x
5   i = 1
6   while(i < y):
7     res = res*x
8     i = i+1
9   return res
```

Exemplo 3

Escreva uma função que calcule $\sum_{i=1}^{n} i = 1 + 2 + \cdots + n$:

```
>>> somatorio(5)
15
```

Exemplo 3

```
1 def somatorio(n):
2   soma = 0
3   i = 1
4   while(i <= n):
5     soma = soma+i
6     i = i+1
7   return soma
```

```
1 n = int(input())
2 print(somatorio(n))
```

- Como podemos validar `somatorio(n)`??

Exemplo 3

Podemos verificar `somatorio(n)` para alguns valores de $x = 1, 2, \ldots, 100$.

$$\sum_{i=1}^{n} i = 1 + 2 + \cdots + n$$

$$1 + 2 + 3 + 4 + 5 = 15$$

$$\sum_{i=1}^{n} i = \frac{(n+1) * n}{2}$$

Exemplo 3

Verificar se `somatorio(i)` é igual à $((n+1)*n)/2$:

```
1  def somatorio(n):
2    soma = 0
3    i = 1
4    while(i <= n):
5      soma = soma+i
6      i = i+1
7    return soma
8
9  def verifica():
10   i = 0
11   while( i <= 100):
12     assert somatorio(i) == ((i+1)*i)/2, "Erro!"
13     i = i+1
14
15 verifica()
```

Fim

Dúvidas?

Leitura complementar

Leitura complementar:

1. https://panda.ime.usp.br/pensepy/static/pensepy/05-Funcoes/funcoes.html
2. https://www.w3schools.com/python/python_functions.asp

Roteiro

Referências

1. Materiais adaptados dos slides do Prof. Eduardo C. Xavier, da Universidade Estadual de Campinas.