Universidade Federal do Espírito Santo

Departamento de Computação e Eletrônica - CEUNES
Programação I / Programação Funcional
Prof. Oberlan Romão

Aula Prática 5

Uma das grandes vantagens de se usar funções é encapsular a implementação da função, ou seja, não precisamos nos preocupar como a função foi codificada, mas sim com o que ela faz e como usá-la. A maioria das funções que iremos criar são funções que recebem um ou mais parâmetros e retornam algum valor para quem a chamou. Vamos analisar dois exemplos simples:

```
1  def soma1():
2      x = float(input("Digite um número: "))
3      y = float(input("Digite outro: "))
4      return x+y
5
6  def soma2(x, y):
7      return x+y
```

Note que ambas as funções retornam o valor da soma de duas variáveis, mas existe uma diferença crucial entre elas: a função `soma1` não recebe parâmetro algum e a função soma `soma2` recebe dois parâmetros. Com essa diferença, a chamada de cada função é relativamente diferente.

A palavra-chave `return` é utilizada para declarar a informação a ser retornada pela função. A mesma funciona também para finalizar a execução do bloco de instrução da função, retornado assim, o valor que estiver a sua frente. Então, a instrução `return` é utilizada tanto para o retorno de valores pelas funções, como também, para finalizar a execução da função. Uma função sem a instrução `return` é finalizada no fim do bloco de instruções.

Exercício 1: (`ex1.py`) Defina e chame uma função `main()` que utilize as duas funções anteriores.

Exercício 2: (`ex2.py`) Analise o código abaixo. Ao ser chamada a função `soma3`, a mensagem `"Chegou aqui!"` é exibida? Porquê?

```
1  def soma3(x, y):
2      return x + y
3      print("Chegou aqui!")
4
5  def main():
6      #Chamada da função soma3
7
8  main()
```

Exercício 3: Crie um *script* (`ex3.py`) que contenha a definição das seguintes funções:

a. Uma função que leia o raio de uma esfera e imprima a área da esfera. Chame-a de `areaEsfera1`.

b. Uma função que receba o raio de uma esfera e imprima a área da esfera. Chame-a de `areaEsfera2`.

c. Uma função que receba o raio de uma esfera e retorne a área da esfera. Chame-a de `areaEsfera3`.

d. Uma função `main()` que utiliza as três funções anteriores.

Lembrando que a área da esfera é dada por $A = 4\pi r^2$. No fim do *script*, chame a função `main()`. Seu *script* deverá ser algo assim:

```
#Definicao da funcao areaEsfera1

#Definicao da funcao areaEsfera2

#Definicao da funcao areaEsfera3

#Definicao da funcao main

main()
```

Exercício 4: Abra o Shell do Python pelo terminal e digite `help(print)`. Veja que existem alguns parâmetros com valores *default*. Ainda no Shell do Python, digite os seguintes comandos:

```
>>> print("ok")

>>> print("ok", end="")

>>> print("ok", end="\n\n")

>>> print("ok", end="---\n")

>>> print("ok", end="\n---\n")

>>> print("1", "2", "3", sep="\n")

>>> print("1", "2", "3", sep="-")
```

Você consegue dizer qual o significado dos parâmetros `end` e `sep`?

Expressões Ternárias

A sintaxe de uma expressão ternária é a seguinte:

```
<expressao1> if <condicao> else <expressao2>
```

Primeiro, a `<condição>` é avaliada (ao invés de `<expressao1>`), se for verdadeira a `<expressao1>` é computada e seu valor é retornado; caso contrário, `<expressao2>` é computada e seu valor retornado. Caso não lembre de todos os detalhes, releia o texto sobre Expressões Ternárias.

Veja um exemplo:

```
def ehPar1(x):
    if x % 2 == 0
        print("Par")
    else:
        print("Impar")

def ehPar2(x):
    if x % 2 == 0
        return "Par"
    else:
        return "Impar"
```

o que pode ser simplificado por:

```
def ehPar1(x):
  print("Par" if x % 2 == 0 else "Impar")

def ehPar2(x):
    return "Par" if x % 2 == 0 else "Impar"
```

Também é permitido o uso de encadeamento:

```
  x = 3 if (y == 1) else 2 if (y == -1) else 1
```

Exercício 5: (`ex5.py`) Reescreva a função `divisivel` usado expressão ternária:

```
def divisivel(x, y):
    if x%y == 0
        y = 0
    else:
        y = 1
    return y
```

Verificando e validando o tipo de uma variável

Em algumas situações, precisamos garantir que a informação passada como argumento para uma função é de determinado tipo. Por exemplo, não faz sentido tentar calcular a raiz quadrada de um `string`. Em Python, como já vimos, podemos verificar se o valor armazenado em uma variável é de determinado tipo usando a função integrada `type` ou a função `isinstance`. Veja alguns exemplos:

```
>>> type(3) == int
True
>>> type(3) == float
False
>>> type(3.0) == float
True
>>> type(3.0) == int
False
```

```
>>> isinstance(3, int)
True
>>> isinstance(3.0, float)
True
```

Dessa forma, podemos utilizar essas funções integradas para validar se o valor armazenado em uma variável é do tipo esperado/desejado:

```
a = 10
if type(a) == int: #ou isinstance(a, int)
    print(f"{a} é inteiro")
else:
    print(f"{a} não é inteiro")
a = 10.0
if isinstance(a, int): #ou type(a) == int
    print(f"{a} é inteiro")
else:
    print(f"{a} não é inteiro")
```

Exercício 6: (ex6.py) Crie uma função, chamada `ehNumero`, que receba um valor e retorne `True` se o valor passado como argumento for um número (inteiro ou real) e `False`, caso contrário. Defina uma função `main()` que exemplifica a utilização da função `ehNumero`.

Funções com `return None`

Como já sabemos a palavra-chave `return` é utilizada para declarar a informação/valor a ser retornada pela função. A mesma funciona também para finalizar a execução do bloco de instrução da função, retornado assim, o valor que estiver a sua frente. Quando a instrução `return` é utilizada sem a definição de um valor a ser retornado, por padrão, o valor que será retornado é `None`. Lembre-se que, em Python, uma função sempre retorna algum valor e quando esse valor não é definido explicitamente, é retornado `None`. Veja um exemplo:

```
def ehPar(x):
    if type(x) == int:
        return True if x%2 == 0 else False
    else:
        return None
```

A função `ehPar` retorna `True` ou `False` se o valor passado para a função for do tipo inteiro. Caso contrário, é retornado `None`. Essa função pode ser simplificada da seguinte forma:

```
def ehPar(x):
    if type(x) == int:
        return True if x%2 == 0 else False
```

Nesse caso, quando `type(x) != int`, a função não executa nenhum comando e se encera, o que é equivalente a retornar `None`. Veja alguns exemplos de utilização da função `ehPar`:

```python
def main():
    print(ehPar(10))
    print(ehPar(13))
    print(ehPar(13.5))
    x = int(input("Digite um número: "))
    if ehPar(x): # equivalente a ehPar(x) == True
        print(f"{x} é par")
    else:
        print(f"{x} é impar")
    x = 10.0
    if ehPar(x) == True:
        print(f"{x} é par")
    elif ehPar(x) == False:
        print(f"{x} é impar")
    else:
        print(f"{x} não é um número inteiro")

main()
```

Exercício 7: (ex7.py) Defina uma função (`volumeCilindro`) que receba o raio e a altura de um cilindro e retorne o volume. Sua função deve, primeiramente, verificar se o raio e a altura são números. Caso não sejam, sua função deve imprimir uma mensagem de erro e retornar `None`. Em seguida, defina e chame uma função `main()` que utiliza a função `volumeCilindro`.

Formatação em Python

Existem várias maneiras de se formatar `string` em Python. Iremos apresentar alguns exemplos. Para saber mais, de forma completa e detalhada, consulte a documentação oficial[1].

Método `format`

Recebe um valor e, opcionalmente, uma especificação de formatação como parâmetro. Retorna um `string`. Abra o Python Shell e teste cada um dos comandos abaixo. Para cada exemplo, modifique os valores para perceber e ficar mais claro como o `format` pode ser usado.

```
>>> format(3)        # apenas o valor
'3'
>>> format(3, "d")   # valor e indicação de inteiro
'3'
>>> format(3, ".2f") # valor e indicação de float com duas casas decimais
'3.00'
>>> format(math.pi, ".4f") # valor e indicação de float com quatro casas decimais
'3.1416'
>>> format(math.pi, ".5f") # valor e indicação de float com cinco casas decimais
'3.14159'
```

[1] https://docs.python.org/3.8/tutorial/

```
>>> format(3, "5d")   # valor e indicação de inteiro em um campo de tamanho cinco
'    3'
>>> format(123, "10d")   # valor e indicação de inteiro em um campo de tamanho dez
'       123'
>>> format(3, "03d")   # campo de tamanho três e preenchido com zero a esquerda
'003'
>>> format(123, "03d")   # campo de tamanho três e preenchido com zero a esquerda
'123'
>>> format(123, "05d")   # campo de tamanho cinco e preenchido com zero a esquerda
'00123'
```

Lembre-se que se você estiver no modo *script* e quiser que o valor seja exibido na tela, você deve usar a função `print`. Por exemplo:

```
print(format(123, "05d"))
```

Método `str.format()`

A string `str` é um template com pontos de substituição delimitados por pares `{}`. Estes pontos serão preenchidos com os valores fornecidos ao método `format`, respeitando-se a posição ou o nome do parâmetro. Veja os exemplos:

Parâmetros por posição

```
>>> print("a = {}".format(100)) # Apenas um parâmetro
'a = 100'
>>> print("a = {}, b = {}".format(100, 200)) # Dois parâmetros, preenchidos na ordem
                                              natural
a = 100, b = 200
>>> print("a = {0}, b = {1}".format(100, 200)) # Dois parâmetros, com indicação da
                                              ordem natural
a = 100, b = 200
>>> print("a = {1}, b = {0}".format(200, 100)) # Dois parâmetros, indicando ordem
                                              inversa
a = 100, b = 200
>>> print("a = {0}, b = {1}, c = {1}".format(100, 200)) #Dois parâmetros, com duas
                                              ocorrências do segundo
a = 100, b = 200, c = 200
```

Parâmetros por nome

```
>>> print("a = {x}, b = {x}".format(x = 100)) # Um parâmetro, duas ocorrências
a = 100, b = 100
>>> print("a = {x}, b = {y}".format(x = 100, y = 200)) # Dois parâmetros, dois nomes
a = 100, b = 200
>>> print("b = {y}, a = {x}".format(x = 100, y = 200)) # Dois parâmetros, dois nomes
                                                      utilizados em ordem inversa
b = 200, a = 100
```

Formatação por tipo

Será feita no template, precedida por `:`. Pode ser colocada isoladamente `:<desc_tipo>` ou após a indicação posição `<pos>:<desc_tipo>` ou do nome `<nome>:<desc_tipo>`. Veja os exemplos com o controle de casas decimais de um `float` e o preenchimento de zero a esquerda de um número `int`.

```
>>> print("a = {:.2f}".format(100)) # Um parâmetro, um float com duas casas decimais
a = 100.00
>>> print("a = {0:.2f}, b = {0:.3f}".format(100)) # Um parâmetro, duas e três casas
                                                  decimais
a = 100.00, b = 100.000
>>> print("b = {y:.1f}, a = {x:.2f}".format(x = 100.1234, y = 100.567)) # Dois
                                                  parâmetros, ordem inversa
b = 100.6, a = 100.12
>>> print("a = {0:02d}, b = {0:03d}".format(5)) #Adicionando zero a esq. do número
a = 05, b = 005
>>> print("a = {0:05d}, b = {1:05d}".format(5, 123))
a = 00005, b = 00123
>>> print("Tempo do corredor: {0:02d}h {1:02d}min".format(3, 5))
Tempo do corredor: 03h 05min
```

Método `f-string`

A formatação usando `f-string` segue o mesmo padrão da formatação por tipo do `format`. Veja alguns exemplos:

```
>>> a = 100
>>> print(f"a = {a:.2f}") # Imprime o valor de 'a' com duas casas decimais
100.00
>>> print(f"a = {a:.3f}") # Imprime o valor de 'a' com três casas decimais
100.000
>>> pi = 3.141592653589793
>>> print(f"a = {a}\npi = {pi:.2f}")
a = 100
pi = 3.14
>>> print(f"a = {a:.5f}\npi = {pi:.5f}")
a = 100.00000
pi = 3.14159
```

```
>>> b = 1
>>> print(f"a = {a}\nb = {b:03d}") #Adicionando zero a esq. do número
a = 100
b = 001
>>> print(f"a = {a:05d}\nb = {b:05d}")
a = 00100
b = 00001
>>> print(f"a = {a:5d}\nb = {b:05d}") #Note a diferença do ex. anterior
a =   100
b = 00001
>>> print(f"a = {a:5d}\nb = {b:5d}")
a =   100
b =     1
>>> print(f"Tempo do corredor: {3:02d}h {33:02d}min")
Tempo do corredor: 03h 33min
```

O último exemplo será usado no miniEP4 :)

Obs.: Na disciplina, usaremos preferencialmente o `f-string`.