Universidade Federal do Espírito Santo

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

Aula Prática 4

O objetivo dessa aula prática é usar algumas funções integradas, algumas funções do modulo matemático e criar funções simples.

Funções integradas

O interpretador do Python possui várias funções que estão sempre disponíveis para uso. Essas funções são chamadas de funções integradas (*built-in functions*).

No site https://docs.python.org/3.8/library/functions.html você encontra uma lista de funções predefinidas disponíveis pelo interpretador do Python.

Exercício: Abra o Python Shell (abra o Terminou ou PowerShell e digite `python`), digite cada um dos comandos abaixo e tente identificar o objetivo de cada função integrada:

```
abs(-1)
abs(1)
round(2.5)
round(3.5)
round(2.6)
round(-2.7)
round(3.7587, 3)
round(3.7587, 2)
round(3.7587)
isinstance(10, int) # ou type(10) == int
isinstance(10.0, int) # ou type(10.0) == int
```

Funções matemáticas

Por padrão o Python disponibiliza poucos recursos, mas pode-se criar ou utilizar recursos criados por outras pessoas. Para se utilizar recursos externos é necessário importar um módulo (subprogramas externos). Um dos módulos mais usados é o módulo matemático, que oferece a maioria das funções matemáticas comuns.

No site https://docs.python.org/3.8/library/math.html você encontra uma lista de funções disponíveis no módulo matemático.

Antes de usar as funções de um módulo, é preciso importá-lo com uma instrução de importação. A forma mais comum é fazer o seguinte:

```python
import math
```

Após o `import`, podemos usar as funções do módulo digitando o nome do módulo, ponto (.) e o nome da função:

```python
raiz2 = math.sqrt(2)
pi = math.pi
piso = math.floor(2.5)
teto = math.ceil(2.5)
fatorial10 = math.factorial(10)
```

Também podemos dar um novo nome ao módulo importado:

```python
import math as m

raiz2 = m.sqrt(2)
pi = m.pi
piso = m.floor(2.5)
teto = m.ceil(2.5)
fatorial10 = m.factorial(10)
```

Se precisarmos de apenas algumas funções específica, podemos importar apenas essas funções. Nesse caso, qualquer função ou constante que não seja explicitamente importada, não será reconhecida

```python
from math import sqrt, exp

raiz2 = sqrt(2)
e = exp(1)
piso = floor(2.5) #Erro
```

Perguntas: Considerando a função `sqrt`:

a) Ela precisa de algum valor para funcionar, ou seja, precisamos passar um valor como argumento para ela? Se sim, quantos valores ela precisa? Podemos passar um valor de qualquer tipo?

b) Ela retorna alguma informação para quem a chamou? Se sim, o que essa informação representa? O que podemos/devemos fazer com essa informação?

c) Analise o código abaixo. Ele possui algum problema? Se sim, qual?

```python
import math as m

m.sqrt(3)
```

Exercício: (`ex1.py`) Faça um programa que leia um valor real (x), calcule e imprima o valor de $f(x)$:

$$f(x) = \begin{cases} \ln|x|, & \text{se } x \leq 1 \\ \log_{10} x + \sqrt{x}, & \text{se } 1 < x \leq 2 \\ x^2 + e^x, & \text{se } 2 < x \leq 5 \\ x^{x/2} + \log_2 x, & \text{se } x > 5 \end{cases}$$

Criando novas funções

Como visto em sala de aula, a sintaxe básica para a criação de uma nova função é a seguinte:

```
def <nome da funcao>(<lista de parametros>):
  """docstring"""
  <comando1>
  <comando2>
  ...
  <comandoN>
  return <expressao1>, <expressao2>, ..., <expressaoM>
```

Algumas funções precisam de uma lista de parâmetros enquanto outras não. O mesmo acontece em relação ao retorno, algumas funções retornam um ou mais valores enquanto outras não possuem a instrução `return`.

Veja que na sintaxe anterior aparece algo que não foi apresentado em sala de aula: `"""docstring"""`. As `docstrings` são strings que inseridas dentro de um código Python com o intuito de fornecer uma explicação sobre o funcionamento do mesmo. Essas strings, delimitados por `"""` e `"""`, devem ser colocadas logo abaixo da definição de um método ou função. O texto representado por tal string será apresentado quando for executado o comando `help()` utilizando como entrada a função onde a `docstring` está inserida. Veja um exemplo:

```
def mensagem():
  """Esta função apenas imprime uma mensagem"""
  print("Olá mundo")
```

Exercício: Crie um *script* (`ex2.py`) com o código abaixo e execute-o.

```
#Define, mas nao executa, a funcao mensagem
def mensagem():
  print("Minha função feita em Python.")
  print("Esse é um exemplo de função sem parâmetro e sem retorno.")

#Define, mas nao executa, a funcao mensagemRepetida
def mensagemRepetida():
  mensagem()
  mensagem()

#Chama (executa) a funcao mensagemRepetida
mensagemRepetida()
```

Exercício: Mova a última linha deste programa para o topo, para que a chamada de função apareça antes das definições. Execute o programa e veja qual é a mensagem de erro que aparece.

Exercício: Mova a chamada de função de volta para baixo e mova a definição de `mensagem` para depois da definição de `mensagemRepetida`. O que acontece quando este programa é executado?

Como esperado, é preciso criar uma função antes de executá-la. Em outras palavras, a definição de função deve ser feita antes da chamada da função.

A Função `main`

Para manter o código bem organizado, podemos separar todo o programa em funções. Neste caso, a ultima linha do código contém uma chamada para a função principal (por convenção chamada de `main`). Além disso, separando todo o código em funções, evitamos o uso de variáveis globais[1] (visto com mais detalhes nas próximas aulas).

```
1  def main():
2      print("Função main")
3
4  main()
5  # Função main
```

Como a chamada da função `main` fica no final do código, não precisamos nos preocupar com a ordem em que as outras funções são definidas.

```
1  def main():
2      função1()
3      função2()
4
5  def função2():
6      print("Função 2")
7
8  def função1():
9      print("Função 1")
10
11 main()
12 # Função 1
13 # Função 2
```

Pergunta: O que aconteceria com esse código, ao ser executado, se a linha 11 fosse comentada?

Exercício: (`ex3.py`) Faça uma função que recebe um valor em segundos e imprime este valor em horas, minutos e segundos. Em seguida, crie uma função `main()` que utilize a função de conversão. Por exemplo:

```
1  def converte(segundos):
2      #implemente a função
3
4  def main():
5      s = int(input("Digite a quantidade de segundos: "))
6      converte(s)
7
8  main()
```

Exercício: (`ex4.py`) Implemente uma função para calcular e imprimir o número de combinações possíveis de n elementos em grupos de p elementos ($p \leq n$), dado pela fórmula de combinação abaixo. Em seguida, crie uma função `main()` que utilize a função.

$$C(n,p) = \frac{n!}{(n-p)!p!}$$

[1] Assim como `for` e `while`, variáveis globais também são proibidos no Paradigma Funcional.