INSTITUTO DE INFORMÁTICA
Universidade Federal de Goiás

Introdução à Python

Altino Dantas

PARTE I

INSTITUTO DE INFORMÁTICA
UFG

UFG
UNIVERSIDADE FEDERAL DE GOIÁS

Objetivos

- Compreender as funções em Python
- Entender funções simples e anônimas

Funções

Entendendo as funções

- Função é uma sequência de instruções que realiza uma operação desejada;
- Podemos definir nossas próprias funções para resolver um problema da nossa aplicação;
- Podemos utilizar qualquer nome para a nossa função, porém é recomendado pela PEP8 que os nomes sejam sempre minúsculo e, caso necessário, sejam separados por _ para facilitar a legibilidade.

https://www.python.org/dev/peps/pep-0008

Criando funções

A sintaxe para definir uma função:

```
def nome_da_funcao(parametros):
    comandos
```

Uma função pode ter quantos comandos forem necessários, mas devemos lembrar que devem estar identados.

Definindo uma função

As funções também podem possuir parâmetros opcionais, e podem ser omitidos durante a chamada de função:

```
def nome_funcao(parametro=padrao):

    comandos
```

Atenção: Para que um parâmetro não seja obrigatório, o mesmo deve ter um valor padrão, que será assumido caso o valor não seja passado.

Definindo uma função

- Quando definimos uma função é necessário escrever suas instruções. Porém quando estamos apenas definindo a estrutura da nossa aplicação é comum utilizarmos a declaração **“pass”**

```
def nome_da_funcao(parametros):
    pass
```

Escopo da função

- Uma função pode utilizar variáveis de escopo local, com isso podemos definir a mesma variável em funções diferentes.

```
def sistema():

    nome = 'Linux'

def curso():

    nome = 'Python'
```

Passando parâmetros

- Para passarmos um parâmetro para a função, podemos fazer da seguinte forma:

```
def nome_da_funcao(parametro):

    print (x)

nome_da_funcao('Aqui eu passo o parametro')
```

Múltiplos retornos

- Como existe tupla, em python é possível retornar mais de um valor numa função:

```
def divisao(dividendo, divisor):
    quociente = dividendo // divisor
    resto = dividendo % divisor
    return quociente, resto

q, r = divisao(35, 3)
_, r = divisao(35, 3)
q, _ = divisao(35, 3)
```

Codificando

- Crie uma função que receba um item da lista **users** e retorne **verdadeiro,** caso a idade seja maior do que 30, caso contrário, **falso**;
- Use a função filter para gerar outra lista apenas com os itens cujo campo idade possui valor maior do que 30, assim:

```
list_fil_30 = filter(nome_função, users)
list(list_fil_30)
```

```
users = [
 {"id": 1,
  “nome": "Altino",
  “idade": 29,
  “cidade": "Goiânia“ },
 { "id": 2,
  “nome": "Dantas",
  “idade": 30,
  “cidade": "Curitiba“ },
 { "id": 3,
  “nome": "Basílio",
  “idade": 31,
  “cidade": "Fortaleza"}
]
```

Funções anônimas (lambda)

- Funções anônimas são aquelas que não estão vinculadas a um nome, em Python podem ser chamadas também como “**expressões lambda**”;
- São muito utilizadas no meio acadêmico para resolução de cálculos matemáticos;
- Funções **lambda** podem ser definidas dentro de uma função, sendo, normalmente, atribuídas a uma variável da função principal.

Funções anônimas (lambda)

Sintaxe

Ao definirmos uma função anônima, utilizamos a seguinte sintaxe:

```
lambda argumentos: expressão
```

É importante lembrar que “lambda” é uma palavra reservada em Python, então não podemos definir nenhuma variável com este nome para que não ocorra nenhum erro.

Utilizando funções anônimas

- No exemplo acima, a função espera receber um argumento (valor de x) e depois utiliza este número para fazer a multiplicação.

```
var = lambda x: x*2

print (var(2))
```

Mais de um parâmetro

- Podemos passar mais de um número para fazer o cálculo, como por exemplo:

```
lamb = lambda a,b,c: ((b ** 2) - (4 * a * c))

print (lamb(3,-2,-5))
```

Mais de uma função

Podemos criar mais de uma função:

```
anonimas = [lambda x: x** 2,
            lambda x: x** 3,
            lambda x: x** 4]

for a in anonimas:
    print (a(10))
```

Codificando

- Faça novamente a filtragem dos item cuja idade é maior do que 30 mas desta vez usando um função lambda dentro do filter:
- Algo como:
  `filter(lambda... , users)`

```
users = [
 {"id": 1,
  “nome": "Altino",
  “idade": 29,
  “cidade": "Goiânia“ },
 { "id": 2,
  “nome": "Dantas",
  “idade": 30,
  “cidade": "Curitiba“ },
 { "id": 3,
  “nome": "Basílio",
  “idade": 31,
  “cidade": "Fortaleza"}
]
```

Exercício

Brincando com dados

	sepal.length	sepal.width	petal.length	petal.width	variety
0	5.1	3.5	1.4	0.2	Setosa
1	4.9	3.0	1.4	0.2	Setosa
2	4.7	3.2	1.3	0.2	Setosa
3	4.6	3.1	1.5	0.2	Setosa
4	5.0	3.6	1.4	0.2	Setosa

```
import pandas as pd
import numpy as np

iris = pd.read_csv('caminho\\IRIS.csv')
iris.head()
iris['variety'].value_counts()

dict_data = {}
for line in range(len(iris['variety'])):
    …
```

- Reproduza o código até a sexta linha, e, com o arquivo IRIS.csv fornecido pelo professor veja se consegue o resultado ao lado;
- Conclua a implementação de modo a preencher o dicionário dict_data seja preenchido com os dados do objeto iris com a seguinte estrutura:

dict_data = {0 : { ‘sepal.length’ : 5.1, ‘sepal.width’ : 3.5, ‘petal.length’ : 1.4, ’petalwidth’ : 0.2, ’variety’ : ’Setosa’ } }

Brincando com dados

- Pesquise como funcionam os métodos update() e items() de dicionários;
- Crie uma lista **setosa_list = [ ]** e **percorra** o dicionário procurando pelos valores nos quais o campo **‘variety’** seja igual a **“Setosa”.**
- Cada item da lista deve ter o formato como o exemplo abaixo:

{ ‘sepal.length’ : 5.1, ‘sepal.width’ : 3.5, ‘petal.length’ : 1.4, ’petalwidth’ : 0.2, ’variety’ : ’Setosa’ }

- Você pode usar **setosa_list.append(item)** para adicionar um novo item na lista.

Obrigado

altinobasilio@inf.ufg.br

Dúvidas ou sugestões?