Introdução a Python e Flask

Por: Juan Funez e Samuel Sampaio

<juan.funez@gmail.com> <samukasmk@gmail.com>

@juanfunez @samukasmk

Agenda

1. Uma breve introdução ao Python;
2. Conceitos bem básicos de HTML;
3. Uma breve introdução ao Flask;

1. Uma breve introdução ao Python

Instalando o Python

Instruções para instalar o ambiente necessário:

bit.ly/InstalandoOPython

Referência: https://docs.python.org/3/using/windows.html

2. Introdução ao Python

O que é o Python

- Linguagem de programação de alto nível;
- Interpretada (através de scripts), porém compilados se necessário;
- Orientada a objetos;
- Tipagem dinâmica e forte (duck typing);
- Criada por Guido Van Rossum em 1991;
- Desenvolvida para ser uma linguagem de leitura fácil, com um visual agradável e alta produtividade;
- Os blocos de código são definidos com indentação, usando espaços ou tabs;

Referência: https://pt.wikipedia.org/wiki/Python

Como executar um programa em Python ?

O python é uma linguagem interpretada, portanto ele necessita de um arquivo (chamado de script), com as instruções definidas.

Crie um arquivo chamado minha_app.py (com o conteúdo abaixo):

```
print('Essa foi a execução da minha primeira aplicação em python')
```

Para executar um programa python:

1. Abra o terminal de seu sistema operacional (seja Bash, MS-DOS, etc)
2. Digite o comando **python**
3. Pressione a tecla **espaço**
4. Logo em seguida digite o **caminho do seu script**
5. Pressione a tecla **enter**

```
$ python minha_app.py
Essa foi a execução da minha primeira aplicação em python
```

Referência: https://www.pythoncentral.io/execute-python-script-file-shell/

O interpretador interativo de comandos

O python possui uma ferramenta poderosa para o fácil aprendizado, o interpretador interativo!

Para executar interpretador interativo basta:

1. **Abra o terminal de seu sistema operacional (seja Bash, MS-DOS, etc);**
2. **Digite o comando python**
3. **Pressione a tecla enter**
4. **Digitar instruções em python**

```
$ python
Python 3.6.5 (default, Apr  1 2018, 05:46:30)
[GCC 7.3.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

Pronto! agora você será capaz de entender o significado de cada instrução do python, de forma bem dinâmica.

Referência: https://docs.python.org/3/tutorial/interpreter.html

Usando o python como uma calculadora

Na linguagem python tudo é objeto, cada objeto deriva de um tipo (**classe**).

Segue alguns tipos de números:

1. **Números inteiros**, representados pela classe (**int**), exemplo: **105**
2. **Pontos flutuantes**, representados pela classe (**float**), exemplo: **105.28**

Em seu terminal que já está aberto, vamos aprender a calcular:

```
>>> 40 + 2
42
>>> 40 - 2
38
>>> 40 * 2
80
>>> 40 / 2
20.0
>>> (20 * 2) + 2
42
```

Referência: https://docs.python.org/3/tutorial/introduction.html#using-python-as-a-calculator

Manipulando textos (str)

Além de números, o Python também pode manipular textos, com o tipo string **(str)**, tão conhecido na informática como cadeia de caracteres.

Existem várias maneiras de se declarar strings, porém o meio mais simples são:

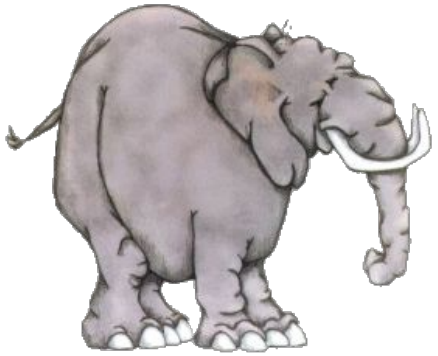
- entre aspas simples ('...')
- ou aspas duplas ("...")

```
>>> 'esse é um texto em python'
'esse é um texto em python'
>>> "esse é um texto em python"
'esse é um texto em python'
>>>
```

Ambas maneiras de se declarar têm o mesmo resultado.

Referência: https://docs.python.org/3/tutorial/introduction.html#strings

Manipulando textos (str)



Como o Python é uma linguagem de tipagem dinamicamente forte, é possível usar operadores numéricos em strings **(str)**.

Como a soma:

```
>>> 'exemplo de concatenação ' + 'de 2 pedaços de texto'
'exemplo de concatenação de 2 pedaços de texto'
```

Ou a multiplicação:

```
>>> '10 elefantes' + (' incomodam' * 10) + ' muito mais'
'10 elefantes incomodam incomodam incomodam incomodam incomodam incomodam
incomodam incomodam incomodam incomodam muito mais'
```

Referência: https://docs.python.org/3/tutorial/introduction.html#strings

Listas de objetos

Lista é uma estrutura de dados, criada para agrupar um conjunto sequencial de objetos por **“índices”**, armazenados em uma única variável como uma **“gaveta”**.

Seguindo nossa analogia ao gaveteiro, vamos pensar em uma lista, como as pastas internas de uma gaveta, que serve para segmentar cada conteúdo.

O **conteúdo de uma lista é obtido através de índices**, que são números inteiros da posição de cada item, assim como as **“etiquetas das pastas internas da gaveta”**.

Referência: https://docs.python.org/3/tutorial/datastructures.html

Listas de objetos

As listas são declaradas por colchetes **[]** onde cada item dentro do colchete deve ser separado por vírgulas.

Segue o exemplo da **declaração** de uma lista de strings, com nomes de frutas:

```
>>> lista_de_frutas = ['pera', 'uva', 'maçã']
>>> lista_de_frutas
['pera', 'uva', 'maçã']
```

Para **obter** um objeto de dentro da lista, deve-se passar o número inteiro de seu índice entre colchetes, **lembrando que primeiro índice é 0**:

```
>>> lista_de_frutas[0]
'pera'
>>> lista_de_frutas[2]
maçã
>>>
```

Referência: https://docs.python.org/3/tutorial/datastructures.html

Inserindo e removendo itens de uma Lista

Uma vez que a lista foi criada, para **inserirmos** novos itens à lista usamos o método **.append()**:

```
>>> lista_de_frutas
['pera', 'uva', 'maçã']
>>> lista_de_frutas.append('MANGA')
>>> lista_de_frutas
['pera', 'uva', 'maçã', 'MANGA']
>>>
```

Para **removermos** um item de uma lista pelo seu índice usamos o método **.pop()**:

```
>>> lista_de_frutas.pop(2)
'maçã'
>>>
>>> lista_de_frutas
['pera', 'uva', 'MANGA']
```

Referência: https://docs.python.org/3/tutorial/datastructures.html

Dicionários de objetos

Dicionário é uma estrutura de dados, criada para agrupar um conjunto de objetos por **“textos chaves”**, armazenados em uma única variável como uma **“gaveta”**.

Podemos pensar em um dicionário também como as pastas internas de uma gaveta, que servem para segmentar cada conteúdo.

O **conteúdo de um dicionário é obtido através de suas chaves**, que são textos de referência, assim como as **“etiquetas das pastas internas da gaveta”**.

Referência: https://docs.python.org/3/tutorial/datastructures.html

Dicionário de objetos

Elas são declaradas por chaves **{}** onde deve se declarar um **“texto chave”** para acessar o **valor**, a separação também é feita por vírgulas.

Segue o exemplo de um dicionário de strings:

```
>>> agenda_telefonica = {'samuel': '11 9999-0000', 'juan': '1234-5678'}
>>> agenda_telefonica
{'samuel': '11 9999-0000', 'juan': '1234-5678'}
```

Para obter um objeto de um dicionário, deve-se passar o **“texto chave”** do objeto entre colchetes, para obter o valor correspondente:

```
>>> agenda_telefonica['samuel']
'11 9999 0000'
>>> agenda_telefonica['juan']
'1234-5678'
```

Referência: https://docs.python.org/3/tutorial/datastructures.html#dictionaries

Inserindo e removendo itens de um dicionário

Uma vez que o dicionário foi criado, para **inserirmos** novos itens à lista devemos passar entre colchetes um **“texto chave”**, com o símbolo de igual **=**, para atribuir o valor a chave:

```
>>> agenda_telefonica['bruno'] = '11 8765 4321'
>>> agenda_telefonica
{'samuel': '11 9999-0000', 'juan': '1234-5678', 'bruno': '11 8765 4321'}
```

Para **removermos** um item de um dicionário usamos o método **.pop()**:

```
>>> agenda_telefonica.pop('bruno')
'11 8765 4321'
>>> agenda_telefonica
{'samuel': '11 9999-0000', 'juan': '1234-5678'}
```

Referência: https://docs.python.org/3/tutorial/datastructures.html#dictionaries

O que é uma função ?

Função é objeto, com instruções encapsuladas em um bloco de código, como uma caixa preta, provendo abstração de código.
Para se executar uma função, deve-se passar o nome dela, prosseguido de seus argumentos (dentro dos parênteses).

Exemplo de chamada da função nativa (**print**) que imprime textos na tela:

```
>>> print('esse texto foi impresso na tela com a função print')
esse texto foi impresso na tela com a função print

>>> print(42)
42
```

Referência: https://docs.python.org/3/library/functions.html#print

Como definir uma função ?

Para definir uma função, utilize a palavra reservada **def**, seguida do nome e opcionalmente entre parênteses uma lista de parâmetros para sua nova função.

Exemplo de declaração de uma função, que imprime textos na tela:

```
def oi(nome):
    print('Olá!', nome)
    print('Tudo bem?')
```

Execute a função definida acima, da mesma forma que executamos a função print

```
>>> oi('Juan')
Olá! Juan
Tudo bem?
```

Referência: https://docs.python.org/3/library/functions.html#print

O que são decorators ?

Um decorator é uma forma prática e reusável de adicionarmos funcionalidades às nossas funções/métodos/classes, sem precisarmos alterar o código delas.

```
def meu_decorador(funcao_decorada):
    def wrapper():
        print('Agora sua função tem super poderes!')
        return funcao_decorada()

    return wrapper
```

```
@meu_decorador
def funcao_simples():
    print('Eu sou uma função simples #SQN')
```

```
>>> funcao_simples()
Agora sua função tem super poderes!
Eu sou uma função simples #SQN
```

Referência: https://pythonhelp.wordpress.com/2013/06/09/entendendo-os-decorators
https://klauslaube.com.br/2011/08/02/decorators-em-python.html

O que são módulos e como importar ?

- **Módulos em Python** são os próprios **arquivos de texto puro** do seu computador, com a extensão **(.py)**.
- Um **pacote para o python**, nada mais é, do que **uma pasta com vários arquivos (.py)**, desde que possua um arquivo **__init__.py** (geralmente sem conteúdo).

OBS: Ao criar a pasta, de um novo pacote, não esqueça de criar um arquivo __init__.py!

O Python possui diversos módulos nativos, segue um exemplo abaixo para importar o módulo **time** e usar a função **sleep** como referência de dentro do módulo:

```
>>> import time
>>> time.sleep(10)
```

Neste outro exemplo, importamos apenas a função **sleep** do módulo **time** e a chamamos direto:

```
>>> from time import sleep
>>> sleep(15)
```

Referência: https://docs.python.org/3/tutorial/modules.html

Virtualenv

O que é um virtualenv?

Virtualenvs são ambientes “virtuais” isolados, contendo em um pasta, todo o python e as dependências instaladas separadas da instalação global. Permitindo que o desenvolvedor trabalhe com tranquilidade sem alterar dependências do sistema operacional ou de outros projetos. Isso é possível manipulando o valor da variável de ambiente **PATH**.

Como criamos um virtualenv?

Para criar um virtualenv usamos o módulo: **venv** que já vem com python. Assim:

```
$ python -m venv meu_venv
```

Como ativamos um virtualenv?

Após criar o nosso venv, cada vez que vamos trabalhar com ele temos que ativá-lo:

No linux e OSX:

```
$ source meu_venv/bin/activate
```

No Windows:

```
$ meu_venv\Scripts\activate.bat
```

Como desativamos um virtualenv?

```
$ deactivate
```

Referência: https://docs.python.org/3.6/library/venv.html

Instalando módulos com o pip

Além dos módulos nativos como o (time), o que torna o Python uma linguagem tão especial atualmente é engajamento da comunidade, perante a quantidade de módulos externos disponíveis (open source) de qualidade, mantido por pessoas e distribuídos pela plataforma oficial do Python o: https://pypi.org/

O **pip** é uma ferramenta de linha de comando responsável por baixar os módulos externos do site pypi.org e armazená-los no ambiente python de sua máquina.

- Para instalar o pacote do flask:

```
python -m pip install flask
```

- Para procurar pacotes para instalar:

```
python -m pip search flask
```

- Para listar pacotes instalados:

```
python -m pip list
```

Referência: https://docs.python.org/3/installing/index.html

2. Conceitos bem básicos de HTML

HTML 101: Conceitos bem básicos de HTML

O que é HTML?

É uma linguagem de marcação de hipertexto. Com ela definimos elementos que compõem de um website, com titulos, paragrafos, links, imagens e etc.

O que são tags ?

O HTML é uma linguagem baseada em marcação. Nós ***marcamos*** elementos para mostrar quais informações a página exibe, por exemplo, um título importante:

```
<h1>Aqui vai o texto do título</h1>
```

Referência: https://tableless.com.br/o-que-html-basico/

HTML 101: Estrutura básica de uma página

Uma página HTML possui basicamente 2 segmentos:

- **head**: Cabeçalho da página, que não é visível, além do título;
- **body**: Corpo da página, onde todos os elementos visíveis são declarados;

```
<!DOCTYPE html>

<html>
    <head>
        <title>Título da página</title>
    </head>

    <body>
        <h1>Aqui vai todo o código HTML que faz seu site...</h1>
    </body>
</html>
```

Referência: https://tableless.com.br/o-que-html-basico/

3. Uma breve introdução ao

Flask

web development,
one drop at a time

Introdução ao Flask

O que é Flask ?

Flask é framework web, simplista, desenvolvido em Python.

Propósitos:

- **Simples:** O Flask nos permite criar aplicações com poucas linhas de código, abstraindo muitas complexidades e ganhando em produtividade.
- **Design patterns “aberto”:** O Flask não toma muitas decisões por você como o banco de dados a ser usado. E as poucas decisões que ele toma, como o mecanismo de templates, são fáceis de alterar.
- **Extensível:** Você encontrará uma variedade de extensões prontas na comunidade, para integrar a seu projeto. Exemplo: flask-admin, flask-login, flask-sqlalchemy, flask-celery e etc encontre mais no site, na seção de extensões.

Referência: http://flask.pocoo.org/docs/1.0/foreword/

Rodando sua primeira aplicação

1. **Crie um arquivo chamado hello.py**

2. **Adicione o conteúdo abaixo:**

```
from flask import Flask
app = Flask(__name__)

@app.route("/")
def hello():
   return "Olá Mundo!"
```

3. **Execute sua aplicação com o comando** (no windows trocar export por set):

```
$ export FLASK_APP=hello.py
$ flask run
```

4. **Acesse seu aplicativo pela url:** http://127.0.0.1:5000/

Referência: http://flask.pocoo.org/

Exemplos de Rotas

1. Altere o arquivo hello.py adicionando a nova view abaixo:

```
@app.route("/rota1")
def view_da_rota1():
   return "Essa é uma nova view da /rota1"
```

2. Salve o arquivo e execute novamente sua aplicação na linha de comando;

3. Acesse seu aplicativo pela url: http://127.0.0.1:5000/rota1

Referência: http://flask.pocoo.org/

Obtendo parâmetros de uma URL por Rotas

1. **Altere o arquivo hello.py adicionando mais uma nova view abaixo:**

```
@app.route("/rota2/<nome>")
def view_da_rota2(nome):
   return "Bem vindo, " + nome + "! Essa é a view da /rota2!"
```

2. **Salve o arquivo e execute novamente sua aplicação na linha de comando;**
3. **Acesse seu aplicativo pela url: http://127.0.0.1:5000/rota2/seunome**

Referência: http://flask.pocoo.org/

Tipos de respostas: String simples

Após uma view ser processada, você deve obrigatoriamente retornar o conteúdo do site, pela returno da função da view, com um objeto de resposta. Segue alguns tipos mais comuns:

1. String simples (como no exemplo anterior)

```
@app.route("/ola")
def view_com_str():
   return "Esse é o conteúdo do meu site, por string simples"
```

Acesse pela URL: http://127.0.0.1:5000/ola

Referência: http://flask.pocoo.org/

Tipos de respostas: Templates Jinja2

2. Templates Jinja2

Templates Jinja são utilizados para compor páginas html onde passamos valores dinâmicos, como o exemplo abaixo (com a variável nome)

```
from flask import render_template
```

```
@app.route("/ola/<nome>")
def view_com_template(nome):
    return render_template("index.html", nome=nome)
```

Crie uma pasta chamada **templates** no mesmo nivel do hello.py

Crie nessa pasta, o arquivo index.html com o conteúdo:

```
Olá de novo, {{ nome }}. Agora no template!
```

Acesse pela URL: http://127.0.0.1:5000/api/seunome

Referência: http://flask.pocoo.org/docs/1.0/api/#flask.render_template

Tipos de respostas: JSON

3. Objetos JSON

Para facilitar o desenvolvimento de APIs REST, onde o retorno em geral é no formato JSON, o Flask disponibiliza nativamente o método **jsonify**, que converte seu objeto Python (exemplo: dicionários ou listas) em JSON.

```
from flask import jsonify
```

```
@app.route("/api")
def view_com_json():
    return jsonify({"mensagem": "Bem vindo a sua primeira API REST!"})
```

Acesse pela URL: http://127.0.0.1:5000/api

Referência: http://flask.pocoo.org/docs/1.0/api/#flask.json.jsonify

FIM

PERGUNTAS ?

Por: Juan Funez e Samuel Sampaio
<juan.funez@gmail.com> <samukasmk@gmail.com>
@juanfunez @samukasmk