Desenvolvimento WEB com Python e Flask

Prof. Igor Avila Pereira
igor.pereira@riogrande.ifrs.edu.br

Instituto Federal de Educação, Ciência e Tecnologia do Rio Grande do Sul (IFRS)
Câmpus Rio Grande
Divisão de Computação

Agenda

Instalação

Windows 64 bits

1. **Instalar o PostgreSQL:** http://www.postgresql.org/download/
2. **Instalar o Python 2.7:** https://www.python.org/downloads/ Obs: Habilite a opção de adicionar ao PATH.
3. Instalar o Flask: Abra o prompt (cmd) do Windows e digite "pip install Flask"
4. **Trabalhar com PostgreSQL com Python:** Instale o psycopg2 (.exe) http://www.stickpeople.com/projects/python/win-psycopg/

Instalação

Ubuntu

1. **Instalar o PostgreSQL:**

   sudo apt-get install postgresql-9.4

   sudo apt-get install pgadmin3

2. **Instalar Python 2.7:**
   http://askubuntu.com/questions/101591/how-do-i-install-python-2-7-2-on-ubuntu

3. **Instalar o Flask:**

   Abra o terminal e digite "pip install Flask"

4. **Trabalhar com PostgreSQL com Python:**

   sudo apt-get install python-psycopg2

Instalação

Ubuntu dos Laboratórios

1. PostgreSQL já está instalado
2. Python já está instalado
3. Instalar o Flask dentro de um diretório específico

   Obs: para cada projeto criado esse passo deverá ser feito novamente.

   1. mkdir application (application foi nome dado ao projeto)
   2. virtualenv application
   3. cd application
   4. source bin/activate
   5. pip install flask
4. Trabalhar com PostgreSQL com Python:

   sudo apt-get install python-psycopg2

Introdução Python - Vapt Vupt

Os arquivos fonte são identificados geralmente pela extensão .py e podem ser executados diretamente pelo interpretador:

```
python apl.py
```

Assim o programa apl.py será executado.

Introdução Python - Vapt Vupt

Python utiliza tipagem dinâmica, o que significa que o tipo de uma variável é inferido pelo interpretador em tempo de execução.

A tipagem do Python é forte, ou seja, o interpretador verifica se as operações são válidas e não faz coerções automáticas entre tipos incompatíveis.

Para realizar a operação entre tipos não compatíveis, é necessário converter explicitamente o tipo da variável ou variáveis antes da operação.

Introdução Python - Vapt Vupt

Compilação e interpretação

O código fonte é traduzido pelo Python para **bytecode**, que é um formato binário com instruções para o interpretador.

O bytecode é multiplataforma e pode ser distribuído e executado sem fonte original.

Introdução Python - Vapt Vupt

Um programa feito em Python é constituído de linhas, que podem continuar nas linhas seguintes, pelo uso do caractere de **barra invertida** ao final da linha ou parênteses, colchetes ou chaves, em expressões que utilizam tais caracteres.

Comentário

O caractere # marca o início de comentário.

Introdução Python - Vapt Vupt

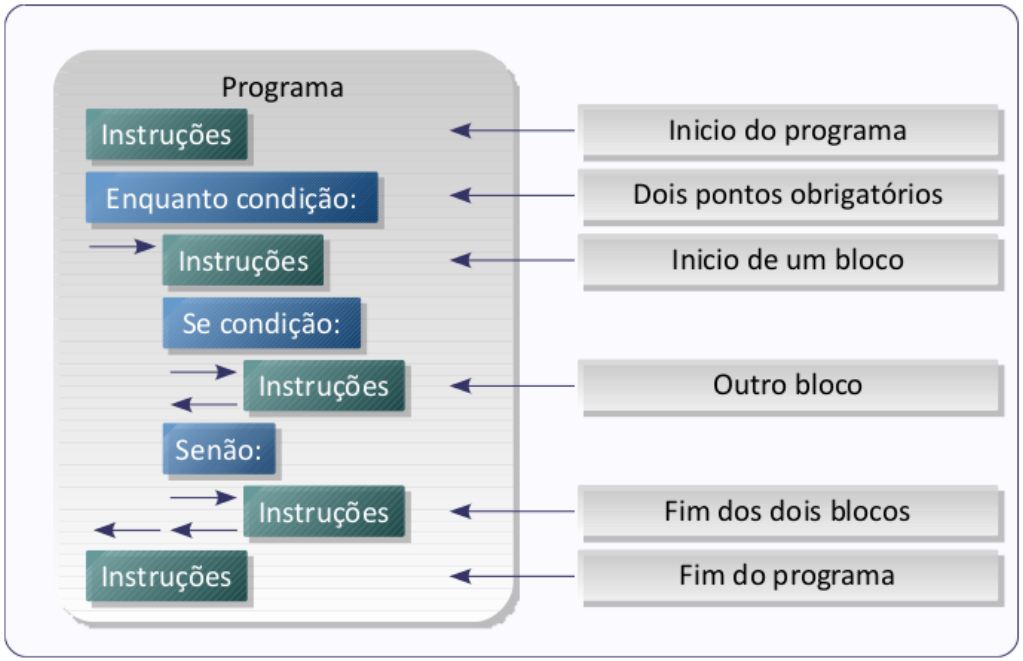
```
# -*- coding: latin1 -*-
# Uma linha quebrada por contra-barra
a = 7 * 3 + \
5 / 2

# Uma lista (quebrada por vírgula)
b = ['a', 'b', 'c',
'd', 'e']

# Uma chamada de função (quebrada por vírgula)
c = range(1,
11)

# imprime todos na tela
print a, b, c
```

Saída:

```
23 ['a', 'b', 'c', 'd', 'e'] [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
```

Figure: Quebra Linha

Introdução Python - Vapt Vupt



Exemplo:

```
# Para i na lista 234, 654, 378, 798:
for i in [234, 654, 378, 798]:

    # Se o resto dividindo por 3 for igual a zero:
    if i % 3 == 0:

        # Imprime...
        print i, '/ 3 =', i / 3
```

Saída:

```
234 / 3 = 78
654 / 3 = 218
378 / 3 = 126
798 / 3 = 266
```

Figure: Blocos

Introdução Python - Vapt Vupt

Controle de fluxo

```
temp = int(raw_input('Entre com a temperatura: '))

if temp < 0:
    print 'Congelando...'
elif 0 <= temp <= 20:
    print 'Frio'
elif 21 <= temp <= 25:
    print 'Normal'
elif 26 <= temp <= 35:
    print 'Quente'
else:
    print 'Muito quente!'
```

Figure: if e else

Introdução Python - Vapt Vupt

Laços

```
# Soma de 0 a 99
s = 0
for x in range(1, 100):
    s = s + x
print s
```

```
# Soma de 0 a 99
s = 0
x = 1

while x < 100:
    s = s + x
    x = x + 1
print s
```

Figure: For While

Tipos: Strings

As strings no Python são buitins para armazenar texto. Como são imutáveis, não é possível adicionar, remover ou mesmo modificar algum caractere de uma string. Para realizar essas operações, o Python precisa criar um nova string.

- *Strings* padrão: s = 'Led Zeppelin'

A inicialização de strings pode ser:

- Com aspas simples ou duplas.
- Em várias linhas consecutivas, desde que seja entre três aspas simples ou duplas.

Introdução Python - Vapt Vupt

Tipos: Strings

```
s = 'Camel'

# Concatenação
print 'The ' + s + ' run away!'

# Interpolação
print 'tamanho de %s => %d' % (s, len(s))

# String tratada como sequência
for ch in s: print ch

# Strings são objetos
if s.startswith('C'): print s.upper()

# o que acontecerá?
print 3 * s
# 3 * s é consistente com s + s + s
```

Figure: Strings

Introdução Python - Vapt Vupt

Tipos: Listas

Listas são coleções heterogêneas de objetos, que podem ser de qualquer tipo, inclusive outras listas.

As listas no Python são mutáveis, podendo ser alteradas a qualquer momento. Listas podem ser fatiadas da mesma forma que as strings, mas como as listas são mutáveis, é possível fazer atribuições a itens da lista.

Introdução Python - Vapt Vupt

Tipos: Listas

```
# Uma nova lista: Brit Progs dos anos 70
progs = ['Yes', 'Genesis', 'Pink Floyd', 'ELP']

# Varrendo a lista inteira
for prog in progs:
    print prog

# Trocando o último elemento
progs[-1] = 'King Crimson'

# Incluindo
progs.append('Camel')

# Removendo
progs.remove('Pink Floyd')

# Ordena a lista
progs.sort()

# Inverte a lista
progs.reverse()

# Imprime numerado
for i, prog in enumerate(progs):
    print i + 1, '=>', prog

# Imprime do segundo item em diante
print progs[1:]
```

Saída:

```
Yes
Genesis
Pink Floyd
ELP
1 => Yes
2 => King Crimson
3 => Genesis
4 => Camel
['King Crimson', 'Genesis', 'Camel']
```

Figure: Listas

Introdução Python - Vapt Vupt

Tipos: Tuplas

Semelhantes as listas, porém são imutáveis: não se pode acrescentar, apagar ou fazer atribuições aos itens.

Sintaxe: tupla = (a, b, ..., z)

Particularidades:

- tupla com apenas um elemento é representada como: t1 = (1,)
- E tuplas podem ser convertidas em listas: lista = list(tupla)

Os elementos de uma tupla podem ser referenciados da mesma forma que os elementos de uma lista: primeiro_elemento = tupla[0]

Introdução Python - Vapt Vupt

Tipos: Dicionários

Um dicionário é uma lista de associações compostas por uma chave única e estruturas correspondentes. Dicionários são mutáveis, tais como as listas. A chave precisa ser de um tipo imutável, geralmente são usadas strings, mas também podem ser tuplas ou tipos numéricos. Já os itens dos dicionários podem ser tanto mutáveis quanto imutáveis. O dicionário do Python não fornece garantia de que as chaves estarão ordenadas.

```
dicionario = {'a': a, 'b': b, ..., 'z': z}
```

Figure: Dicionário

Introdução Python - Vapt Vupt

Tipos: Dicionários

```
# Progs e seus albuns
progs = {'Yes': ['Close To The Edge', 'Fragile'],
    'Genesis': ['Foxtrot', 'The Nursery Crime'],
    'ELP': ['Brain Salad Surgery']}

# Mais progs
progs['King Crimson'] = ['Red', 'Discipline']

# items() retorna uma lista de
# tuplas com a chave e o valor
for prog, albuns in progs.items():
    print prog, '=>', albuns

# Se tiver 'ELP', deleta
if progs.has_key('ELP'):
    del progs['ELP']
```

Figure: Dicionário: Exemplos

Introdução Python - Vapt Vupt

Operadores Booleanos

Os operadores booleanos no Python são: and, or, not, is e in.

- **and**: retorna um valor verdadeiro se e somente se receber duas expressões que forem verdadeiras.
- **or**: retorna um valor falso se e somente se receber duas expressões que forem falsas.
- **not**: retorna falso se receber uma expressão verdadeira e vice-versa.
- **is**: retorna verdadeiro se receber duas referências ao mesmo objeto e falso em caso contrário.
- **in**: retorna verdadeiro se receber um item e uma lista e o item ocorrer uma ou mais vezes na lista e falso em caso contrário

Introdução Python - Vapt Vupt

Módulos

Para o Python, módulos são arquivos fonte que podem importados para um programa. Podem conter qualquer estrutura do Python e são executados quando importados. Eles são compilados quando importados pela primeira vez e armazenados em arquivo (com extensão .pyc ou .pyo), possuem namespace próprio e aceitam Doc Strings.

São objetos Singleton (é carregada somente uma instância em memória, que fica disponível de forma global para o programa).

Introdução Python - Vapt Vupt

Módulos

```
import os
print os.name
```

Também possível importar módulos de forma relativa:

```
from os import name
print name
```

O caractere "*" pode ser usado para importar tudo que está definido no módulo:

```
from os import *
print name
```

Por evitar problemas, como a ofuscação de variáveis, a importação absoluta é considerada uma prática de programação melhor do que a importação relativa.

Introdução Python - Vapt Vupt

```
def fatorial(n):
        n = n if n > 1 else 1
        j = 1
        #for i in range(1, n + 1):
        i = 1
        while  i <= n:
                j = j * i
                i = i + 1
        return j

# Testando...
for i in range(1, 6):
        print i, '->', fatorial(i)
```

Figure: Funções

Introdução Python - Vapt Vupt

```
# Uma lista de instrumentos musicais
instrumentos = ['Baixo', 'Bateria', 'Guitarra']
# Para cada nome na lista de instrumentos
for instrumento in instrumentos:
        # mostre o nome do instrumento musical
        print instrumento
```

Figure: Vetores

Introdução Python - Vapt Vupt

Python é uma linguagem orientada a objeto, sendo assim as estruturas de dados possuem atributos (os dados em si) e métodos (rotinas associadas aos dados). Tanto os atributos quanto os métodos são acessados usando ponto (.).

Para mostrar um atributo: **print** objeto.atributo

Para executar um método: objeto.metodo(argumentos)

Mesmo um método sem argumentos precisa de parênteses: objeto.metodo()

Figure: Objetos

Introdução Python - Vapt Vupt

```
class User(object):
    """Uma classe bem simples.
    """
    def __init__(self, name):
        """Inicializa a classe, atribuindo um nome
        """
        self.name = name
```

Figure: Classes

Introdução Python - Vapt Vupt

```
class Pendrive(object):

    def __init__(self, tamanho, interface='2.0'):

        self.tamanho = tamanho
        self.interface = interface


class MP3Player(Pendrive):

    def __init__(self, tamanho, interface='2.0', turner=False):

        self.turner = turner
        Pendrive.__init__(self, tamanho, interface)


mp3 = MP3Player(1024)

print '%s\n%s\n%s' % (mp3.tamanho, mp3.interface, mp3.turner)
```

A classe *MP3Player* é derivada da classe *Pendrive*.

Figure: Herança

Python suporta herança múltipla

Flask

Relembrando...

Criação - Projeto Flask Python no Ubuntu do Laboratório

1. mkdir **application**
2. virtualenv **application**
3. cd **application**
4. source bin/activate
5. pip install flask
6. touch **index.py**
7. gedit **index.py**

Usuários Ubuntu..instale antes:

```
$ sudo apt-get install python-virtualenv
```

HelloWorld

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

@app.route('/')
def hello_world():
    return 'Hello World!'

if __name__ == '__main__':
    app.run()
```

Figure: hello.py

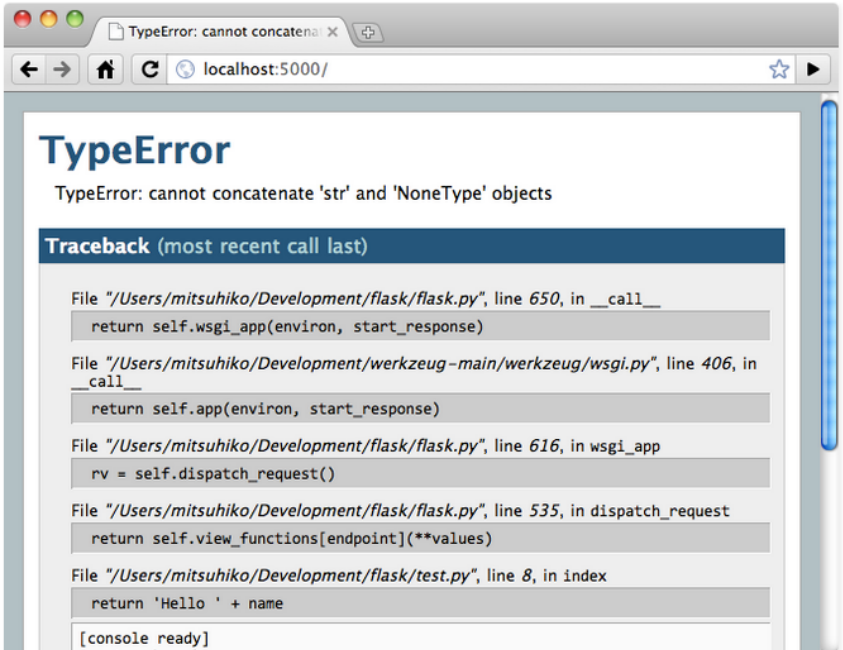
```
$ python hello.py
 * Running on http://127.0.0.1:5000/
```

Figure: Executar HelloWorld Flask

Debug

```
app.debug = True
app.run()
```

Figure: Habilitar Debug



Rotas

```python
@app.route('/')
def index():
    return 'Index Page'

@app.route('/hello')
def hello():
    return 'Hello World'
```

Figure: Definindo Rotas

Rotas

```python
@app.route('/user/<username>')
def show_user_profile(username):
    # show the user profile for that user
    return 'User %s' % username

@app.route('/post/<int:post_id>')
def show_post(post_id):
    # show the post with the given id, the id is an integer
    return 'Post %d' % post_id
```

Figure: Definindo Rotas

A variável **post_id** já é automaticamente convertida para **int**

Rotas

```python
@app.route('/login', methods=['GET', 'POST'])
def login():
    if request.method == 'POST':
        do_the_login()
    else:
        show_the_login_form()
```

Figure: Definindo Rotas

Observações:

default é GET

É preciso importar **request**: from flask import request

Acessando dados vindos por POST

```python
@app.route('/login', methods=['POST', 'GET'])
def login():
    error = None
    if request.method == 'POST':
        if valid_login(request.form['username'],
                       request.form['password']):
            return log_the_user_in(request.form['username'])
        else:
            error = 'Invalid username/password'
    # the code below is executed if the request method
    # was GET or the credentials were invalid
    return render_template('login.html', error=error)
```

Figure: Acessando dados vindos por POST

Sessão

```
from flask import Flask, session, redirect, url_for, escape, request

app = Flask(__name__)

@app.route('/')
def index():
    if 'username' in session:
        return 'Logged in as %s' % escape(session['username'])
    return 'You are not logged in'

@app.route('/login', methods=['GET', 'POST'])
def login():
    if request.method == 'POST':
        session['username'] = request.form['username']
        return redirect(url_for('index'))
    return '''
        <form action="" method="post">
            <p><input type=text name=username>
            <p><input type=submit value=Login>
        </form>
    '''

@app.route('/logout')
def logout():
    # remove the username from the session if it's there
    session.pop('username', None)
    return redirect(url_for('index'))

# set the secret key.  keep this really secret:
app.secret_key = 'A0Zr98j/3yX R~XHH!jmN]LWX/,?RT'
```

Redirecionamento

```python
from flask import abort, redirect, url_for

@app.route('/')
def index():
    return redirect(url_for('login'))

@app.route('/login')
def login():
    abort(401)
    this_is_never_executed()
```

Figure: Exemplo de Redirecionamento

Templates

```python
from flask import render_template

@app.route('/hello/')
@app.route('/hello/<name>')
def hello(name=None):
    return render_template('hello.html', name=name)
```

Figure: Exemplo - Templates

Templates

Case 1: a module:

```
/application.py
/templates
    /hello.html
```

Case 2: a package:

```
/application
    /__init__.py
    /templates
        /hello.html
```

Figure: Organização de Diretórios necessária para trabalhar com Template

Templates

```python
@app.route('/')
def show_entries():
    cur = g.db.execute('select title, text from entries order by id desc')
    entries = [dict(title=row[0], text=row[1]) for row in cur.fetchall()]
    return render_template('show_entries.html', entries=entries)
```

Figure: Exemplo - Listagem

```
<ul class=entries>
{% for entry in entries %}
  <li><h2>{{ entry.title }}</h2>{{ entry.text|safe }}
{% else %}
  <li><em>Unbelievable.  No entries here so far</em>
{% endfor %}
</ul>
```

Figure: Exemplo - Listagem HTML

Templates

```python
from flask import Flask
from flask import render_template

app = Flask(__name__)

@app.route('/')
@app.route('/<name>')
def hello(name=None):
    vetor = ['igor', 'marcio']
    teste = "teste"
    return render_template('hello.html', name=name, vetor=vetor, teste=teste)

if __name__ == "__main__":
    app.debug = True
    app.run()
```

Figure: Exemplo - Template

Templates

```
1 <!doctype html>
2 <title>Hello from Flask</title>
3
4 {{teste}}
5
6 {% if name %}
7   <h1>Hello {{ name }}!</h1>
8 {% else %}
9   <h1>Hello World!</h1>
10 {% endif %}
11
12 <table>
13   {% for v in vetor %}
14         <tr> <td> {{v}}  </td> </tr>
15  {% endfor %}
16 </table>
```

Figure: Exemplo - Template

Desenvolvimento WEB com Python e Flask

Prof. Igor Avila Pereira
igor.pereira@riogrande.ifrs.edu.br

Instituto Federal de Educação, Ciência e Tecnologia do Rio Grande do Sul (IFRS)
Câmpus Rio Grande
Divisão de Computação