IO Lab: Python Web Frameworks Flask

November 18, 2013

Info 290TA (Information Organization Lab)
Kate Rushton & Raymon Sutedjo-The
Python
Python

Is an interactive, object-oriented, extensible programming language.

---

**I learned it last night! Everything is so simple!**

**Hello world is just print "Hello, world!"**

**I dunno... dynamic typing? Whitespace?**

**Come join us! Programming is fun again! It's a whole new world up here!**

**But how are you flying?**

**I just typed import antigravity**

**That's it?**

**... I also sampled everything in the medicine cabinet for comparison.**

**But I think this is the Python.**
Syntax

• Python has semantic white space.

//JavaScript
if (foo > bar)
    { foo = 1;
      bar = 2;  }

# python
if foo > bar:
    foo = 1
    bar = 2
Indentation Rules

• Relative distance
• Indent each level of nesting

if foo > bar:
    for item in list:
        print item

• Required over multiple lines; single line uses semi-colons

if a > b: print "foo"; print "bar";
Variables

• Must begin with a letter or underscore, can contain numbers, letters, or underscores

```python
foo = 0
_bar = 20
another1 = "test"
```

• Best practice – separate multiple words with underscores, do not use camel case

```python
api_key = "abcd12345" #not apiKey
```
Comments

• Single-line comments are marked with a #

# this is a single line comment

• Multi-line comments are delineated by three "

" " " this is a comment
        that spans more than one line
" " " 
## Data Types

<table>
<thead>
<tr>
<th>Type</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>int</td>
<td>14</td>
</tr>
<tr>
<td>float</td>
<td>1.125</td>
</tr>
<tr>
<td>bool</td>
<td>True/False</td>
</tr>
<tr>
<td>str</td>
<td>“hello”</td>
</tr>
<tr>
<td>list</td>
<td>[“a”,”b”,”c”]</td>
</tr>
<tr>
<td>tuple</td>
<td>(“Oregon”, 1, False)</td>
</tr>
<tr>
<td>dict</td>
<td>{ “name”: “fred”, “age”, 29 }</td>
</tr>
</tbody>
</table>
Strings

• Defined with single or double quotes

```python
fruit = "pear"
name = 'George'
```

• Are immutable

```python
fruit[0] = "b"  # error!
```
Strings

"hello" + "world"  -->  "helloworld"

"hello" * 3       -->  "hellohellohello"

"hello"[0]        -->  "h"

"hello"[-1]       -->  "o"

"hello"[1:4]      -->  "ell"

"hello" < "jello"  -->  True
More String Operations

len("hello") --> 5

"g" in "hello" --> False

"hello".upper() --> "HELLO"

"hello".split('e') --> ["h", "llo"]

" hello ".strip() --> "hello"

"hello" < "jello" --> True
Lists

- Usually defined with brackets
  
  \[
  \text{fruit} = ["apple", "cherry", "kiwi"]
  \]

- Can contain different data types
  
  \[
  \text{stuff} = [3, "blind", "mice"]
  \]

- Are mutable
  
  \[
  \text{stuff}[0] = "three"
  \]
Lists

fruit = [“apple”, “cherry”, “kiwi”]

fruit[0] --> “apple”

fruit[1] = “pear” --> [“apple”, “pear”, “kiwi”]

fruit.append(“grape”) --> [“apple”, “cherry”, “kiwi”, “grape”]

fruit.insert(1, “grape”) --> [“apple”, “grape”, “cherry”, “kiwi”]

fruit.index( “cherry”) --> 1

fruit.index( “orange”) --> ValueError
Looping Through Lists

• With `for..in`

```python
for letter in ["a", "b", "c"]:
    print letter
```

• With `range`

```python
for i in range(len(list)):
    print list[i]
```
Looping Through Lists

• Use `enumerate` to get both index and item

```python
for i, letter in enumerate(['a', 'b', 'c']):
    print(letter * (i+1))
```
List Comprehension

- When performing an action on every item in a list, can do something like

```python
doubled = []
for x in [2, 3, 4, 5]:
    doubled.append(x*2)
```

- **List comprehensions** make this simpler

```python
doubled = [x*2 for x in [2, 3, 4, 5]]:
```
Dictionaries

- Defined with braces {} or `dict()` constructor
- Key-value pairs – keys must be immutable (most often strings/numbers)

```python
pet = { "name": "fido",
       "type": "dog" }
```

Standard dictionaries are unordered.
Dictionaries

employee = {  "id": 133, "name": "Bob", "location": "Chicago" }

len(employee)  -->  3
employee["id"]  -->  133
employee.get("id")  -->  133
employee.values()  -->  [133, "Bob", "Chicago"]
del employee["location"]  -->  { "id" : 133, "name": "Bob"}
"Bob" in d  -->  False
Looping through Dictionaries

• With `for..in`

```python
for key in mydict:
    print mydict[key]
```

• With `iteritems/items`

```python
for key, val in mydict.items():
    print key + "::" + val
```
Functions

- Defined with the keyword `def` followed by the function name and then arguments in parens

- Return values with the `return` keyword

```python
def add_two_things(arg1, arg2):
    return arg1 + arg2

sum = add_two_things(1,2)  # sum is 3
```
Functions

- Can provide default values for arguments in the function definition

```python
def add_two_things(arg1=2, arg2=2):
    return arg1 + arg2;

sum = add_two_things(1,2)  # sum is 3
sum = add_two_things()     # sum is 4
```
Classes and Objects

• Define a class with the `class` keyword
  ```
  class Car:
      #instance data
      #and function definitions
  ```

• Define instance data with `self`
  ```
  self.make = "Geo"
  self.model = "Prism"
  self.year = 1997
  self.max_speed = 60
  ```
Classes and Objects

• All functions in a class have the parameter `self`
  ```python
def stop(self):
    self.speed = 0
def go(self, speed):
    self.speed = speed
  ```

• Implement the `__init__` constructor function
  ```python
def __init__(self, model, year):
    self.model = model
    self.year = year
    # other setup tasks
  ```
class Employee:
    'Represents one employee'
    def __init__(self, first, last, salary):
        self.first_name = first
        self.last_name = last
        self.salary = salary

    def give_raise(self, amount=100):
        self.salary += amount

    def get_name(self):
        return first + " " + last
Classes and Objects

```python
new_hire = Employee("Bob", "Jones", 30000)

new_hire.give_raise(1000)
# salary is now 31K

new_hire.full_name()
# outputs Bob Jones
```
Web Frameworks
File-Folder Web

http://www.mysite.com/dir/subdir/subdir/page.html
Modern Web

https://twitter.com/barackobama/status/401427096276201474
Web Frameworks

CakePHP, Symfony, CodeIgniter

Django, web2py, Bottle, Flask

Ruby on Rails, Sinatra

Spring, Struts
Web Framework Features

- Routing
- Templates
- Database integration / ORM
- Unit testing
- Client or server-side validation
- Localization
- Scaffolding
- Asset management
# Web Framework Features

## Routing

<table>
<thead>
<tr>
<th>Route</th>
<th>Method</th>
<th>Parameters</th>
<th>Result (JSON)</th>
</tr>
</thead>
<tbody>
<tr>
<td>/</td>
<td>GET</td>
<td></td>
<td>Returns 'hello world'</td>
</tr>
<tr>
<td>/messages</td>
<td>GET</td>
<td></td>
<td>Returns all messages</td>
</tr>
<tr>
<td>/messages</td>
<td>POST</td>
<td>name=foo(comment=bar)</td>
<td>Creates a new message with the posted values</td>
</tr>
<tr>
<td>/search</td>
<td>GET</td>
<td>name=foo</td>
<td>Returns all messages where name = 'foo'</td>
</tr>
<tr>
<td>/messages/1</td>
<td>GET</td>
<td></td>
<td>Gets the message with id=1</td>
</tr>
<tr>
<td>/messages/1</td>
<td>POST</td>
<td>name=foo(comment=bar)</td>
<td>Update the message with id=1</td>
</tr>
<tr>
<td>/messages/1</td>
<td>DELETE</td>
<td></td>
<td>Delete the message with id=1</td>
</tr>
</tbody>
</table>
Web Framework Features

Templates

```erb
<h1 style="color: #CC0000">Please Enter New Post... </h1>

<%= error_messages_for :post %>

<% form_for @post do |f| %>
    <p>
        <b>Post Title</b><br />
        <%= f.text_field :title, :class => "textareas", :size
    </p>
    
    <p>
        <b>Text of Post...</b><br />
        <%= f.text_area :body, :cols => "60", :rows => "10", :
    </p>
    
    <p>
        <%= f.datetime_select :created_at %>
    </p>
    
    <p>
        <%= f.submit "Create", :style => "width: 548px" %></p>
    <p>
        <%= link_to 'Back', posts_path %>
    </p>
<% end %>
```
Web Framework Features

Database Integration/ORM
Web Framework Features

Unit Testing

```python
import unittest
import quotes

class MyTests (unittest.TestCase):

def test_add_get_quote (self):
    quotes.add("Confucius", "A journey of a thousand miles ...
    q = quotes.get("Confucius", contains ="step")
    self.assertEqual(q,[ "A journey of a thousand miles ...

def test_add_get_quote_no_contains (self):
    quotes.add("Confucius", "A journey of a thousand miles ...
    q = quotes.get("Confucius")
    self.assertEqual(q,[ "A journey of a thousand miles ...

if __name__ == "__main__":
    unittest.main()
```
Web Framework Features

Validation

```csharp
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.ComponentModel.DataAnnotations;

namespace DataAnnotationSample.Models
{
    public class Employee
    {
        [Display(Name = "Name: ")]
        [Required(ValidationMessage = "Name is Required")]
        public string EmployeeName { get; set; }

        [Display(Name = "Designation: ")]
        [Required(ValidationMessage = "Designation is Required")]
        public string Designation { get; set; }

        [Display(Name = "Age: ")]
        [Required(ValidationMessage = "Age is Required")]
        public int? Age { get; set; }

        [Display(Name = "Place: ")]
        [Required(ValidationMessage = "Place is Required")]
        public string Place { get; set; }

        [Display(Name = "Contact: ")]
        [Required(ValidationMessage = "Contact is Required")]
        public string Contact { get; set; }
    }
}
Web Framework Features

Localization
Web Framework Features

Scaffolding

Create Read Update Destroy

- Create/Add resource
- Edit/Update resource
- View/Show resource
- List/All resources
- Delete resource
Web Framework Features

Asset management

HTML 5

JavaScript

CSS 3
Model-View-Controller

Application architecture that emphasizes separation of business logic and user interaction

- Model – the data and rules that apply to it
- View – a representation of that data in a UI
- Controller – handles requests and mediates between model and view
MVC Grocery Store

Model FoodItem
• Attributes: id, name, brand, price, units, whether on sale, number in stock, etc.
• Functions: getPrice(), updateInventory(), etc.

ID: 1093
Name: Apples Breaburn
Brand: Dole
Price: 1.10
Units: each
Number In Stock: 1200
MVC Grocery Store

View Item Details
- HTML template for a food item detail view
- Model may change in response to user action
MVC Grocery Store

Controller

• Look for incoming requests, eg a GET request to http://mvcgrocery.com/item/1093
• Retrieve item #1093 from db and use it to populate an item details template page
Django

- Routing ✔
- Templates ✔
- Database integration / ORM ✔
- Unit testing ✔
- Client or server-side validation ✔
- Localization ✔
- Scaffolding ✔
- Asset management ✔
Flask – A Microframework

- Routing ✓
- Templates ✓
- Database integration / ORM ✗
- Unit testing ✗
- Client or server-side validation ✗
- Localization ✗
- Scaffolding ✗
- Asset management ✗
from flask import Flask

app = Flask(__name__)

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

if __name__ == '__main__':
    app.run()
@app.route('/search')
def search():
    q = request.args.get('query')
    #implement search for query,
    #return search results page
Flask – Sample Route 2

```python
@app.route('/items/<id>')</def food_item(id):
    #get details about food item with
    #specified id from db
    #return with details template
```
@app.route('/api/items')
def get_all_items():
    # get all items from db
    # return in JSON array
Flask – Route Method

```python
@app.route('/items', methods=['GET'])
def get_all_items():
    #get all items from db
    #return in item list page template

@app.route('/items', methods=['POST'])
def post_item():
    params = request.form
    #use params to add a new item to db
    #return success message
```
Flask – Response HTML

```python
@app.route('/bad', methods=['GET'])
def bad_html_page():
    message = "Don't do this."
    page = "<html><body><p>" + message + "</p></body></html>"
    return page
```
Flask – Route with Template

@app.route('/items/<id>')
def food_item(id):

    food = get_item_object_from_db()
    return render_template('item.html', item=food)
Templates

- Templates are HTML with special markup for server-side rendering
- There are many template engines, but Flask uses jinja2
- Braces/percent signs separate code and html

```html
<ul>
  {% for book in library %}
    <li>{{ book.title }}</li>
  {% endfor %}
</ul>
```
Flask – Directory Structure

app.py
static (directory)
   --css and js files
   --images
templates (directory)
   --template files
Sample Item Detail Template

<html>
    <head>
        <title>{{item.name}}</title>
    </head>
    <body>
        <div id="item-details">
            <p>Name: {{item.name}}</p>
            <p>Price: {{item.price}}</p>
        </div>
    </body>
</html>
Jinja2 Template Logic

- **Conditionals**
  
  ```
  {% if book.available %}
    <p>{{ book.title }}</p>
  {% endif %}
  ```

- **Loops**
  
  ```
  <ul>
  {% for book in library %}
    <li>{{ book.title }}</li>
  {% endfor %}
  </ul>
  ```
Jinja2 Template Logic

- Math
  <p>Price with sales tax: {{ book.price * .08 }}</p>

- Filters
  {{ name|upper }}
  {{ name|strip }}