Introduction to Software Systems Lab

Python, Flask

02/07/2021

Flask

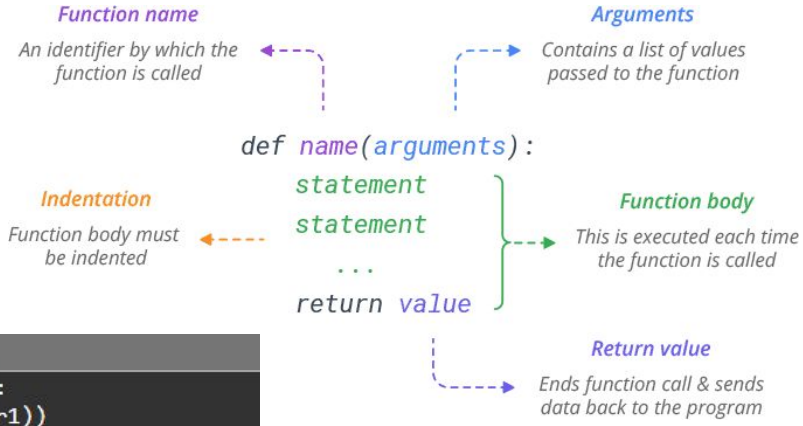
- Classes, functions, exceptions.
- HTTP Requests
- Flask setup.
- Demo: Building a simple calculator using GET and POST requests in Flask.

Python Packages

- A python package is a set of python code libraries which can be used in python project
- Packages can be downloaded using Python package manager `PIP` through command line interface
- Refer to Installation — pip 20.2.3 documentation for installation and detailed instructions
- Refer to Installing Packages for package installation using `PIP`

Functions

- Declaration/Syntax
- Passing as arguments
- Keyword arguments, variable arguments.
- Inbuilt functions, decorators



main.py

```
1  def example(par1, par2, par3):
2      print("par1 is " + str(par1))
3      print("par2 is " + str(par2))
4      print("par3 is " + str(par3))
5
6  args = ("Hello!",1,True)
7  kwargs = {"par1": "Bye!", "par2": 2, "par3" : Fal
8  example(*args)
9  example(**kwargs)
```

```
par1 is Hello!
par2 is 1
par3 is True
par1 is Bye!
par2 is 2
```

Classes, Objects and Methods

- Classes - A structure for holding data together
- Objects - Instance of a class
- Methods - functions called inside objects

```
class MyFirstClass:
    def __init__(self, name):
        self.name = name

    def greet(self):
        print('Hello {}!'.format(self.name))

my_instance = MyFirstClass('John Doe')
my_instance.greet()
```

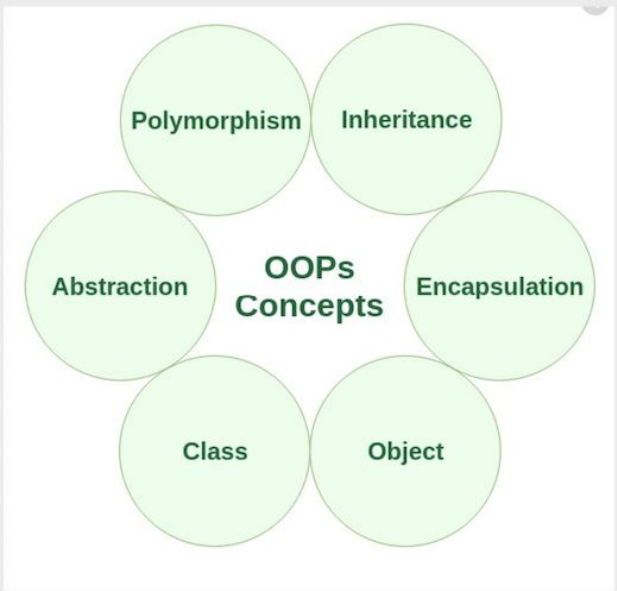
Class

Method

Object

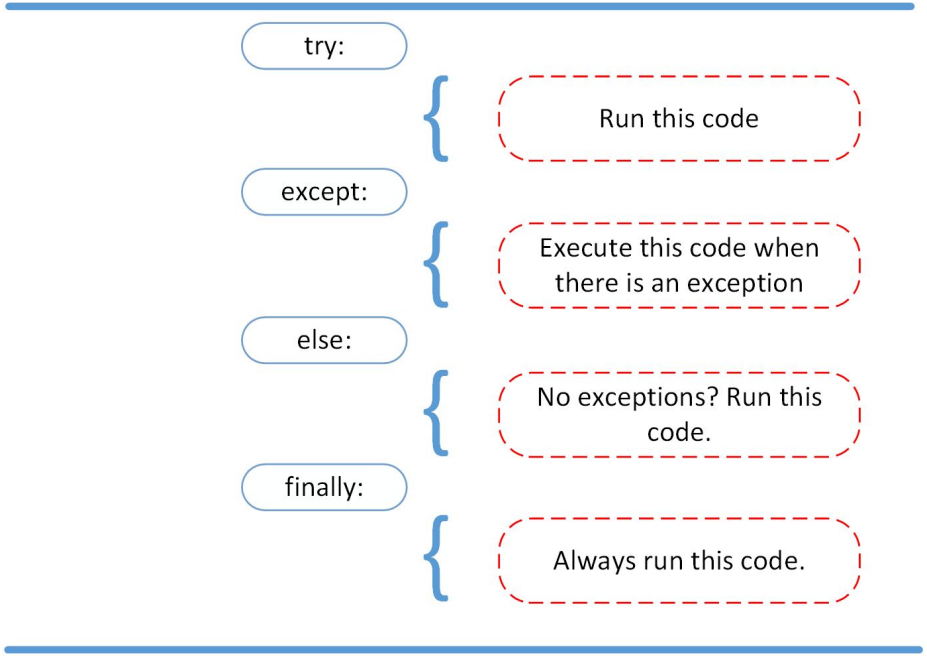
Classes, Objects and Methods

- 4 concepts of OOPS
  - Abstraction - hiding methods.
  - Encapsulation - private, protected variables. Use of getters/setters.
  - Inheritance - inherited classes.
  - Polymorphism - multiple types of methods



Exceptions

- try-except-finally



Flask Setup

- Virtual environment setup (go to virtualenv - PyPI for more details)
  - `sudo apt-get install python3-venv`
  - `virtualenv -p python3 venv`
  - `source venv/bin/activate`

OR

- Flask installation using PIP (Go to Installation — Flask Documentation (1.1.x) for more details)
  - `pip install flask`

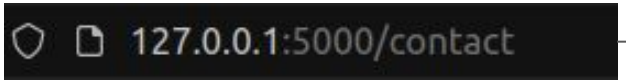
What is Flask?

- Started as an April Fool Joke, now 2nd most used after Django.
- A python micro-web framework, with WSGI env & Jinja2 template engine.
- Does not include database abstraction layers, simple functionality.
- RESTful architecture.
- Open-source, lot of add-ons (we explore some later).
- Lightweight, and easier to pick up.

Basics

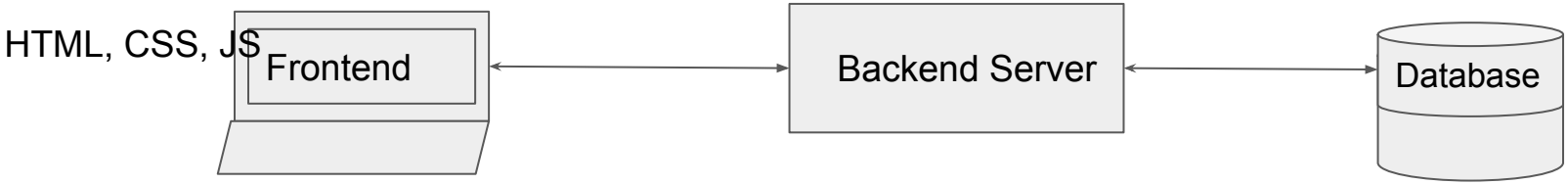
- **Server**: An application created from the Flask application. Ours will start at http://127.0.0.1:5000/
- **Routes**: The endpoints of URLs associated with your app.

/info



/contact

- **Frontend, backend and database**: The HTML files serve on the frontend the flask app as the backend server. In the next labs, we will learn about databases and SQLAlchemy.



HTTP Requests (GET/POST)

GET

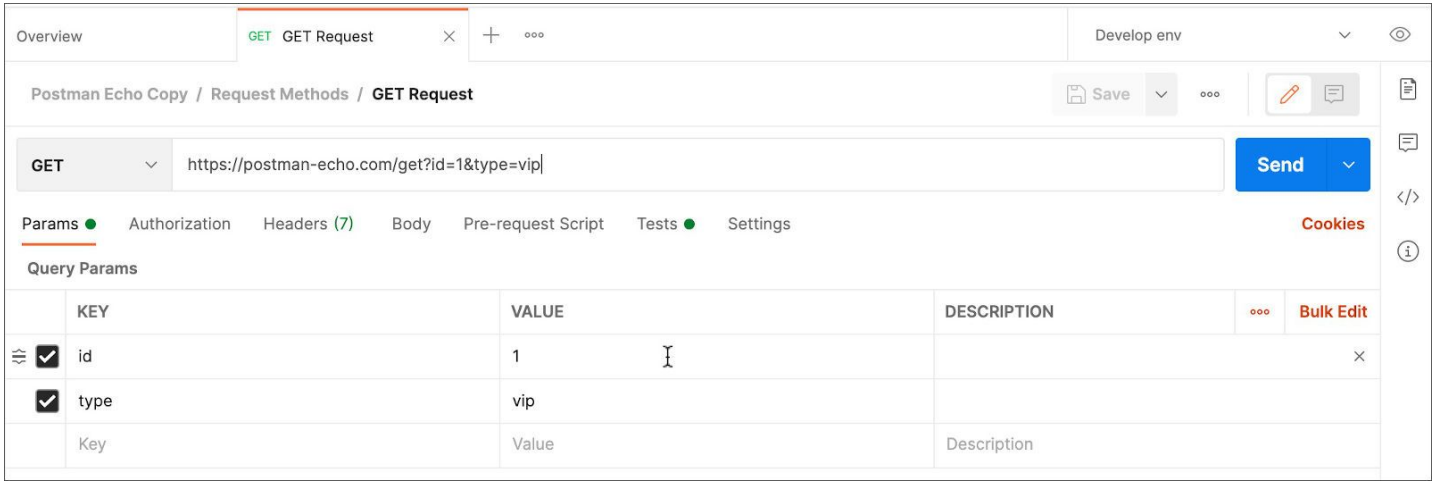
GET method is used to appends form data to the URL in name or value pair.

No payload/body required.

POST

POST method returns a value stored in the data based on the parameters sent.

Payload/Body required.



HTTP Requests (GET/POST)

GET

GET method is used to appends form data to the URL in name or value pair.

No payload/body required.

```
$("button-get").click(function(){
  $.get("/calc", function(data, status){
     value = data.info
  });
});
```

POST

POST method returns a value stored in the data based on the parameters sent.

Payload/Body required.

```
$("button-calc").click(function(){
  $.post("/calc",
  {
          val1: $(“#val1”).val(),
          val2: $(“#val2”).val(),
          operation: $('#dropdown :selected').text();
  },
  function(data, status){
     value = data.info
  });
});
```

Understanding Code Structure

- app.py → The server application
- **static**
  - styles.css
  - scripts.js

  } Contains the static files (CSS/JS/img)
- **templates**
  - base.html
  - home.html
  - calc.html

  } The HTML files

Demo

- Add, subtract or multiply two or more numbers by passing them as
  - Query parameters in URL
    - Example:/calculate?n=2&n=4&n=2&operation=mul → 16
  - JSON payload in POST request
    - Example:
      - Payload:
        - `{ “numbers”=[2,3,4], “operation”: “add” }`
      - Output: 9
- Use of headers for authentication.

Understanding Code - app.py

```
from flask import Flask, render_template
```

1. Importing the class

Understanding Code - app.py

```
from flask import Flask, render_template
```

```
app = Flask(__name__)
```

2. Instantiate the Flask class

Understanding Code - app.py

```
from flask import Flask, render_template
```

```
app = Flask(__name__)
```

3. Decorators

```
@app.route('/')
def home_page():
    return render_template('home.html')
```

4. Returns this file in folder templates/.

Data from URL to server

Giving variable as argument -

1. Passing variable value/type in URL.

   ‘/info/<string:name>’

   ```
   @app.route('/info/<string:name>')
   ```

   ‘/ids/<int:id>’

   ```
   @app.route('/ids/<int:id>')
   ```

2. Passing variable name as well in URL.

   ‘/calc/var1=2&var2=5’

Using Jinja2

Some examples -

- Extends: Used to extend the definition of an HTML file using another.

```
{% extends 'base.html' %}
```

- Blocks: Blocks of code with a label used as a placeholder.

Label of block

```
{% block title %}
        <h1>
                Head Page
        </h1>
{% endblock %}
```

Label

```
    <title>
        {% block title %}

        {% endblock %}
    </title>
```

Data from server to templates

Pass them as parameters to the render_template function -

```
return render_template('calc.html',params=params)
```

Render them in Jinja2

```
<!-- Your rows inside the table HERE: -->
{% for item in items %}
    <tr>
        <td>{{ item.id }}</td>
        <td>{{ item.name }}</td>
        <td>{{ item.barcode }}</td>
        <td>{{ item.price }}$</td>
        <td>
            <button class="btn btn-outline
            <button class="btn btn-outline
        </td>
    </tr>
{% endfor %}
```

Data from templates to server

Using POST requests. We use AJAX in javascript to send our requests.

```
$("button-calc").click(function(){
  $.post("/calc",
  {
          val1: $(“#val1”).val(),
          val2: $(“#val2”).val(),
          operation: $('#dropdown :selected').text();
  },
  function(data, status){
     value = data.info
  });
});
```

Using url_for

It is mainly used in 2 purposes -

1. Rendering Static files

2. Redirecting to another URL along with redirect.

```
@app.route('/login')
def login_page():
    return redirect(url_for('home_page'))
```

Learn and Build Webapps

- Flask course (recommended for beginners)
- Flask Quickstart
- Flask Cheat Sheet
- Jinja Blocks
- The Hitchhiker's Guide to Python
- Python Examples