**Python**

→ Python was created by Guido Van Rossum, released in 1991.

* Uses

- Web development ( Server Side)

- Software development

- Mathematics

- System scripting

* Platform

Windows, Mac, Linux, Raspberry Pi, etc

→ High level, free and open source language

→ Python runs on an interpreter system, meaning that code can be executed as soon as it is written. This means that prototyping can be very quick.

→ procedural way, an object-oriented way or a functional way.

**How python works?**

**Source code >> Interpreter >>Output**

A minimal python program looks something like this:

print(“Hello, World!”);

Hello World with a variable:

msg = “Hello, World!”

print(msg)

Print with user input:

name = input()

print(f"Hello, {name}!")

Print with user input:

name = input()

print(f"Hello, {name}!")

If-elif-else statement

x = 0

if x > 0:

print("x is positive")

elif x<0:

print("x is negative")

else:

print("x is zero")

Make a list

bikes = ["trek", "honda", "yamaha", "giant"]

Get the first item in a list

first\_bike = bikes[0]

Get the last item in a list

last\_bike = bikes[-1]

Looping through the list

for bike in bikes:

print(bike)

Adding Item to list

bikes.append("Royal Enfield")

Sets

s=set()

s.add(1)

s.add(3)

s.add(5)

s.add(3)

>>> print(s)

Output: {1, 3, 5}

Dictionary:

ages = {"Alice":22, "Bob":27}

ages["Charlie"]=30

ages["Alice"] +=1

>>> print(ages)

{'Alice': 23, 'Bob': 27, 'Charlie': 30}

Functions

def square(x):

return x\*x

for i in range(10):

print("{} square is {}".format(i, square(i)))

Use function from other files

save above function in functions.py file, then

from functions import square

print(square(5))

The above program when executed also runs each lines in functions.py. So, in order to eliminate unwanted code execution we can use main as function.

def square(x):

return x\*x

def main():

for i in range(10):

print("{} square is {}".format(i, square(i)))

if \_\_name\_\_ == '\_\_main\_\_':

main()

Python Classes

Python is object-oriented programming language. A class is like an object constructor, or a “blue print”

for creating objects.

To create a class, use the keyword class

class Point:

def \_\_init\_\_(self, x, y):

Class definition

self.x=x

self.y=y

p=Point(3,5)

print(p.x)

Object Definition

print(p.y)

* How to make 'python' command in terminal run python3 instead of python2

modify your ~/.bashrc file, add a new line: alias python=python3.x

* #is used to write comments in Python

**Flask**

## Dependencies

These distributions will be installed automatically when installing Flask.

* [Werkzeug](https://palletsprojects.com/p/werkzeug/) implements WSGI, the standard Python interface between applications and servers.
* [Jinja](https://palletsprojects.com/p/jinja/) is a template language that renders the pages your application serves.
* [MarkupSafe](https://palletsprojects.com/p/markupsafe/) comes with Jinja. It escapes untrusted input when rendering templates to avoid injection attacks.
* [ItsDangerous](https://palletsprojects.com/p/itsdangerous/) securely signs data to ensure its integrity. This is used to protect Flask’s session cookie.
* [Click](https://palletsprojects.com/p/click/) is a framework for writing command line applications. It provides the flask command and allows adding custom management commands.

**Optional Dependencies**

* [Blinker](https://pythonhosted.org/blinker/) provides support for [Signals](https://flask.palletsprojects.com/en/1.1.x/signals/" \l "signals).
* [SimpleJSON](https://simplejson.readthedocs.io/) is a fast JSON implementation that is compatible with Python’s json module. It is preferred for JSON operations if it is installed.
* [python-dotenv](https://github.com/theskumar/python-dotenv" \l "readme) enables support for [Environment Variables From dotenv](https://flask.palletsprojects.com/en/1.1.x/cli/" \l "dotenv) when running flask commands.
* [Watchdog](https://pythonhosted.org/watchdog/) provides a faster, more efficient reloader for the development server.

1. Create an environment {Virtual Environment}

$ mkdir myproject

$ cd myproject

$ python3 -m venv venv

On Windows:

$ py -3 -m venv venv

1. Activate the environment

$ source venv/bin/activate

On Windows:

$ env\Scripts\activate

1. Install Flask

$ pip install Flask

## A Minimal Application

A minimal Flask application looks something like this:

from flask import Flask

app = Flask(\_\_name\_\_)

@app.route('/')

def hello\_world():

return 'Hello, World!'

just save it as hello.py or something similar. To run use command:

$ flask run

\*Debug mode: off

\*Running on http://127.0.0.1:5000/

To enable Debug mode:

$ export FLASK\_ENV=development

On Windows:

$ set FLASK\_ENV=development

→ This does the following things:

1. It activates the debugger
2. Activates the automatic realoder
3. It enables the debug mode on the Flask application.

Note: Debug Mode should not be used on production machines.

**Routing**

→ Use the route() decorator to bind a function to a URL.

@app.route("/")

def index():

return "Hello, World!"

@app.route("/nepal")

def natabar():

return "Hello, Nepal!"

**To tell terminal which application to work with use the following command:**

$ export FLASK\_APP = hello.py

For Windows:

$ set FLASK\_APP = hello.py

from flask import Flask

app = Flask(\_\_name\_\_)

@app.route("/")

def index():

return "Hello World!"

@app.route("/<string:name>")

def hello\_world(name):

name = name.capitalize()

return f"Hello, {name}!"

**Use render\_template module:**

from flask import Flask, render\_template

app = Flask(\_\_name\_\_)

@app.route("/")

def index():

return render\_template("index.html")

Flask will look for templates in the templates folder.

So, if your application is a module, this folder is next to that module, if its a package it is actually inside your package:

Case 2: (A package)

/application

/\_\_init\_\_.py

/templates

/index.html

Case 1:

/application

/templates

/index.html

from flask import Flask, render\_template

app = Flask(\_\_name\_\_)

@app.route("/")

def index():

heading = "Hello, World!"

return render\_template("index.html", headline=heading)

In index.html file:

<h1>{{ headline }}</h1>

from flask import Flask, render\_template

import datetime

app=Flask(\_\_name\_\_)

@app.route("/")

def index():

headline = "main route"

now = datetime.datetime.now()

new\_year = now.month ==1 and now.day==1

return render\_template("index.html",headline=headline, new\_year=new\_year)

In index.html file:

<h1>{{ headline }}</h1>

{% if new\_year %}

<h1>Yes! Happy New Year!</h1>

{% else %}

<h1>No! It's Not a New Year.</h1>

{% endif %}

from flask import Flask, render\_template

app = Flask(\_\_name\_\_)

@app.route("/")

def index():

names = ['Alice', 'Bob', 'Charlie']

return render\_template("index.html", names=names)

@app.route ("/more")

def more():

return render\_template("more.html")

In index.html file:

<body>

<h1>Names</h1>

<ul>

{% for name in names %}

<li>{{ name }}</li>

{% endfor %}

</ul>

<a href="{{ url\_for('more') }}">See more..</a>

</body>

In more.html file:

<body>

lPython is a popular programming language created by Guido Van Rossum, released in 1991.

Python is high level, free and open source language.

<a href="{{ url\_for('index') }}">See previus</a>

</body>

**Template Inheritance**

from flask import Flask, render\_template

app= Flask(\_\_name\_\_)

@app.route('/')

def index():

return render\_template('index.html')

@app.route('/more')

def more():

return render\_template('more.html')

In index.html

{% extends "layout.html" %}

{% block heading %}

First Page

{% endblock %}

{% block body %}

<p>

ython is object-oriented programming language. A class is like an object constructor, or a “blue print”

for creating objects.</p>

<a href="{{ url\_for('more') }}">See more...</a>

{% endblock %}

In more.html

{% extends "layout.html" %}

{% block heading %}

Second Page

{% endblock %}

{% block body %}

<p>

Python is object-oriented programming language. A class is like an object constructor, or a “blue print”

for creating objects. Python is object-oriented programming language. A class is like an object constructor, or a “blue print”

for creating objects.</p>

<a href="{{ url\_for('index') }}">See previous...</a>

{% endblock %}

In layout.html

<!DOCTYPE html>

<html lang="en" dir="ltr">

<head>

<meta charset="utf-8">

<title>My Website!</title>

</head>

<body>

<<h1>{% block heading %} {% endblock %}</h1>

{% block body %}

{% endblock %}

</body>

</html>

**Forms**

from flask import Flask, render\_template, request

app = Flask(\_\_name\_\_)

@app.route("/")

def index():

return render\_template("index.html")

@app.route("/hello", methods=["GET", "POST"])

def hello():

if request.method == "GET":

return "Please submit the form instead."

else:

name = request.form.get("name")

return render\_template('hello.html', name=name)

In index.html

{% extends "layout.html" %}

{% block heading %}

First Page

{% endblock %}

{% block body %}

<form action="{{url\_for('hello') }}" method="post">

<input type="text" name="name" placeholder="Enter Your Name">

<button>Submit</button>

</form>

{% endblock %}

In hello.html

{% extends "layout.html" %}

{% block heading %}

Hello!

{% endblock %}

{% block body %}

<h2>{{ name }}</h2>

{% endblock %}

In layout.html

<!DOCTYPE html>

<html lang="en" dir="ltr">

<head>

<meta charset="utf-8">

<title>My Website!</title>

</head>

<body>

<h1>{% block heading %} {% endblock %}</h1>

{% block body %}

{% endblock %}

</body>

</html>

**SESSION**

from flask import Flask, render\_template, request, session

from flask\_session import Session

app = Flask(\_\_name\_\_)

app.config["SESSION\_PERMANENT"] = False

app.config["SESSION\_TYPE"] = "filesystem"

Session(app)

@app.route("/", methods=["GET", "POST"])

def index():

if session.get("notes") is None:

session['notes'] = []

if request.method == "POST":

note = request.form.get("note")

session['notes'].append(note)

return render\_template("index.html", notes=session['notes'])

In index.html

<body>

{% block body %}

<ul>

{% for note in notes %}

<li>{{ note }}</li>

{% endfor %}

</ul>

<form action="{{ url\_for('index') }}" method="post">

<input type="text" name="note" placeholder="Enter your note">

<button>Add Note</button>

</form>

{% endblock %}

</body>