PythonBasicCourse-es008

October 4, 2022

1 Curso básico de Python

1.1 Apuntes

Curso básico de Python. Apuntes por Marcelo Horacio Fortino. Versión 2.3. Octubre 2022.

Versión	Autor/es	Fecha	Observaciones
1.0	Marcelo Horacio Fortino	2021/Marzo	Curso Python
1.1	Marcelo Horacio Fortino	2021/Junio	Convertido a markdown - ipynb
1.2	Marcelo Horacio Fortino	2021/Agosto	Actualizados contenidos
1.3	Marcelo Horacio Fortino	2021/Octubre	Agregado Flask microframework
1.4	Marcelo Horacio Fortino	2021/Noviembre	Agregado Pandas - datascience
1.5	Marcelo Horacio Fortino	2021/Diciembre	Agregado Devops - Ansible
2.0	Marcelo Horacio Fortino	2022/Abril	Nueva estructura: core / module
2.1	Marcelo Horacio Fortino	2022/Junio	Módulo apuntes intermedio
2.2	Marcelo Horacio Fortino	2022/Agosto	Actualizado temario y ejercicios
2.3	Marcelo Horacio Fortino	2022/Octubre	Actualizado despliegue a render.com

Esta obra se distribuye con la esperanza de que sea útil, pero SIN NINGUNA GARANTÍA, incluso sin la garantía MERCANTIL implícita o sin garantizar la CONVENIENCIA PARA UN PROPÓSITO PARTICULAR. El autor no asume ninguna responsabilidad si el lector hace un mal uso de la misma.

Estos apuntes se basan en: - La documentación oficial de Python, https://docs.python.org/es/3/tutorial/index.html, - La bibliografía presentada al final de este documento, y - Documentación propia recogida a lo largo de los años de diversas fuentes.

1.2 Objetivo del curso

- Objetivo general del curso:
- Aprender a usar Python para crear scripts y programas simples plenamente funcionales, desarrollar aplicaciones web y realizar análisis de datos.

- Objetivos específicos:

- Reconocer las características principales de Python y su utilidad práctica.
- Identificar tipos de datos (simples y compuestos) y operadores.
- Aplicar variables y estructuras de control de flujo.
- Construir funciones y clases (POO).
- Clasificar los módulos y paquetes por sus funcionalidades y objetivos.
- Comparar el lenguaje con otros similares de scripts, procedimentales y orientados a objetos.
- Probar bibliotecas para conexiones REST a aplicaciones web y bases de datos.
- Resolver problemas y errores en el código fuente proponiendo soluciones alternativas (refactoring).
- Utilizar la biblioteca pandas junto con matplotlib y numpy para realizar análisis estadísticos de datos y gráficos.

- Como resultado práctico al final del curso cada estudiante habrá creado una aplicación web utilizando el microframework de Python Flask.

1.3 Temario

- Introducción, instalación y compilación
- Datos, expresiones y sentencias
- Variables y funciones, control de flujo
- Clases y objetos, herencia, polimorfismo
- Entradas y salidas con Python
- Gestión de módulos, paquetes y bibliotecas
- Servicios y programas en red, REST API
- Desarrollo de aplicaciones web con Flask
- Análisis de datos con pandas, matplotlib y numpy

- Módulos opcionales
  - SQL ejemplos con pandas
  - Plotting con Python
  - Machine Learning con Python
  - DevOps con Ansible
  - SDK de GCP para Python
  - Kubernetes con Python y Docker

1.4 Bibliografía

- Downey, A., Elkner, J., Meyers, C. Aprenda a Pensar Como un Programador con Python. (2015).
  Recuperado de https://argentinaenpython.com/quiero-aprender-python/aprenda-a-pensar-como-un-programador-con-python.pdf
- Kent D. Lee. Python, Programming Fundamentals Second Edition. 2014.
- Marzal Varó, A., Gracia Luengo, I., García Sevilla, Pedro. Introducción a la programación con Python 3. (2014).
  Recuperado de http://repositori.uji.es/xmlui/handle/10234/102653
- Miller, B., Ranum, D. Solución de problemas con algoritmos y estructuras de datos usando Python. Traducido por Mauricio Orozco-Alzate, Universidad Nacional de Colombia - Sede Manizales.
  Recuperado de https://runestone.academy/ns/books/published/pythoned/index.html?mode=browsing
- Shaw, Z. A., Learn Python 3 the Hard Way. (2016).
  Recuperado de https://learnpythonthehardway.org/
- Van Rossum, G. and the Python development team. Documentación de Python en español. (2020).
  Recuperado de https://python-docs-es.readthedocs.io/es/3.10/

2 Flask

- Crear la aplicación, Agregar páginas, Agregar plantillas, Conexión a Github, Despliegue a render.com, Recursos.
- Flask es un framework minimalista escrito en Python que permite crear aplicaciones web rápidamente y con un mínimo número de líneas de código.
- Está basado en:
  - La especificación WSGI de Werkzeug https://wsgi.readthedocs.io/en/latest/what.html.
  - El motor de templates Jinja2 https://jinja.palletsprojects.com/en/3.1.x/.
- Fuente: https://flask.palletsprojects.com/en/2.1.x/.
- En este workshop iremos paso a paso desarrollando una aplicación en Flask para luego hacer el despliegue en la plataforma de aplicaciones cloud de Render https://www.render.com/.
- Como requisito se deberá crear una cuenta gratuita en esa plataforma.
- Esto nos permitirá al final del mismo tener una aplicación web en Python disponible en Internet.
- Iniciamos nuestra tarea creando el ambiente virtual:

[ ]:
```
# Comandos ejecutados en GNU/Linux
mkdir app
python3 -m venv app/venv
cd app/
```

```
source venv/bin/activate
```

[ ]:
```
# Comandos ejecutados en Windows
python3 -m venv app/venv
cd app/venv/Scripts
./activate
```

- Instalamos el microframework Flask y el módulo gunicorn:

[ ]:
```
python3 -m pip install flask gunicorn
```

- El fichero *requirements.txt* sirve para instalar módulos de la aplicación si ésta es cambiada de entorno (Sistema operativo, Servidor, etc.).
- Se puede crear manualmente e actualizarlo cuando sea necesario escribiendo los nombres de los módulos a medida que se instalan.

- Como ejemplo, en GNU/Linux creamos el fichero y agregamos *Flask* en el mismo:

[ ]:
```
touch requirements.txt
echo "flask" > requirements.txt
```

- Posteriormente para instalar todos los módulos en el nuevo entorno se ejecutará:

[ ]:
```
python3 -m pip install -r requirements.txt
```

- La siguiente opción (alternativa a la anterior y preferible) crea el fichero *requirements.txt* automáticamente.
- Se suele ejecutar una vez completo el desarrollo de la aplicación y previo a la migración al nuevo entorno.

[ ]:
```
pip freeze > requirements.txt
```

2.1 Crear la aplicación

- El primer paso es crear el script de ejecución de la aplicación.
- Abrimos un fichero que por convención se lo denomina *app.py* y escribimos el siguiente código modificando el autor y agregando la fecha:

[ ]:
```
"""
[Aplicación básica del microframework Flask de Python]

Author: Fortinux
Date: []
"""
from flask import Flask

app = Flask(__name__)
```

```
@app.route("/")
def index():
    return "<H1>Hola Mundo!</H1>"
```

- Para exportar la variable de ambiente *FLASK_APP* ejecutamos:

[ ]:
```
export FLASK_APP=app # Bash de Linux
set FLASK_APP=app # Terminal de Linux
$env:FLASK_APP = "app" # Microsoft Powershell
```

- La variable de ambiente para ambientes de producción predeterminada es *production*:

[ ]:
```
export FLASK_ENV=production # Bash de Linux
```

- Si queremos habilitar el ambiente de desarrollo para no tener que reiniciar la aplicación cada vez que hacemos cambios en ella, exportamos la variable de ambiente:

[ ]:
```
export FLASK_ENV=development # Bash de Linux
set FLASK_ENV=development # Terminal de Linux
$env:FLASK_ENV = "development" # Microsoft Powershell
```

- Para ejecutar la aplicación:

[ ]:
```
flask run
```

- Se agrega el argumento *–host=0.0.0.0* para publicarlo en la red local.
- Se accede a la aplicación en nuestra máquina desde la URL *127.0.0.1:5000* en el navegador.
- Para acceder desde la red local se utiliza la IP de la máquina donde está alojada la aplicación.

2.2 Crear páginas

- Para crear las páginas servicios, contacto y admin de la aplicación:

[ ]:
```
from flask import Flask


app = Flask(__name__)


@app.route("/")
def home():
    return "<H1>Esta es la página principal</H1>"


@app.route("/servicios")
def servicios():
    return "<H1>Esta es la página de servicios</H1>"
```

```
@app.route("/contacto")
def contacto():
    return "<H1>Esta es la página de contacto</H1>"


@app.route("/admin")
def admin():
    return "<H1>Esta es la página de admin</H1>"
```

2.3 Agregar plantillas

- Para agregar plantillas de diseño web se utilizará *Bootstrap*.
- Bootstrap Front-end open source toolkit

- https://getbootstrap.com/.

- Creamos el fichero *index.html* en un nuevo directorio llamado *templates*.
- Esta página HTML será la página principal de la aplicación.

[ ]:
```
vim templates/index.html
```

[ ]:
```
<!doctype html>
<html>
<head>
  <title>Página principal</title>
    <meta name="description" content="Página principal">
    <meta name="keywords" content="HTML template - plantilla">
</head>

<body>
    {% extends "base.html" %}
    {% block title %}Página principal{% endblock %}
    {% block content %}
    <h2>Esta es la página principal</h2>
    <p>Un párrafo de bienvenida.</p>
    {% endblock %}
</body>
</html>
```

- El siguiente paso es crear una plantilla base para toda la aplicación.
- Tendrá de nombre *base.html* y deberá ser alojada en el directorio *templates* como el resto de las plantillas que se irán agregando a la aplicación.

[ ]:
```
vim templates/base.html
```

[ ]:
```
<!doctype html>
<html>
<head>
```

```
    <title>{% block title %}{% endblock %}</title>
    <meta name="description" content="Página principal">
    <meta name="keywords" content="html template">
<link href="https://cdn.jsdelivr.net/npm/bootstrap@5.1.0/dist/css/bootstrap.min.
 ↪css" rel="stylesheet"
integrity="sha384-KyZXEAg3QhqLMpG8r+8fhAXLRk2vvoC2f3B09zVXn8CA5QIVfZOJ3BCsw2P0p/
 ↪We" crossorigin="anonymous">
</head>

<body>
<nav class="navbar navbar-expand-lg navbar-light bg-light">
  <div class="container-fluid">
    <a class="navbar-brand" href="https://fortinux.com/">fortinux</a>
    <button class="navbar-toggler" type="button" data-bs-toggle="collapse"
    data-bs-target="#navbarSupportedContent"␣
 ↪aria-controls="navbarSupportedContent"
    aria-expanded="false" aria-label="Toggle navigation">
      <span class="navbar-toggler-icon"></span>
    </button>
    <div class="collapse navbar-collapse" id="navbarSupportedContent">
      <ul class="navbar-nav me-auto mb-2 mb-lg-0">
        <li class="nav-item">
          <a class="nav-link active" aria-current="page" href="/">Página␣
 ↪Principal</a>
        </li>
        <li class="nav-item">
          <a class="nav-link" href="servicios">Servicios</a>
        </li>
        <li class="nav-item">
          <a class="nav-link" href="contacto">Contacto</a>
        </li>
        <li class="nav-item">
          <a class="nav-link" href="admin">Admin</a>
        </li>
        <li class="nav-item dropdown">
          <a class="nav-link dropdown-toggle" href="#" id="navbarDropdown"␣
 ↪role="button"
            data-bs-toggle="dropdown" aria-expanded="false">Menú desplegable</a>
          <ul class="dropdown-menu" aria-labelledby="navbarDropdown">
            <li><a class="dropdown-item" href="#">Submenú</a></li>
            <li><a class="dropdown-item" href="#">Otro submenú</a></li>
            <li><hr class="dropdown-divider"></li>
            <li><a class="dropdown-item" href="#">Otro enlace</a></li>
          </ul>
        </li>
        <li class="nav-item">
```

```
          <a class="nav-link disabled" href="#" tabindex="-1"␣
 ↪aria-disabled="true">Deshabilitado</a>
        </li>
      </ul>
      <form class="d-flex">
        <input class="form-control me-2" type="search" placeholder="Search"␣
 ↪aria-label="Search">
        <button class="btn btn-outline-success" type="submit">Buscar</button>
      </form>
    </div>
  </div>
</nav>
        <h1>Nombre de la organización</h1>

        {% block content %}{% endblock %}

<script src="https://cdn.jsdelivr.net/npm/bootstrap@5.1.0/dist/js/bootstrap.
 ↪bundle.min.js"
integrity="sha384-U1DAWAznBHeqEIlVSCgzq+c9gqGAJn5c/
 ↪t99JyeKa9xxaYpSvHU5awsuZVVFIhvj"
crossorigin="anonymous"></script>
</body>
</html>
```

- Actualizamos el script ejecutable *app.py*:

[ ]:
```
from flask import Flask, render_template


app = Flask(__name__)


@app.route("/")
def home():
    return render_template("index.html")


@app.route("/servicios")
def servicios():
    return render_template("base.html")


@app.route("/contacto")
def contacto():
    return render_template("base.html")
```

```
@app.route("/admin")
def admin():
    return render_template("base.html")
```

2.4 Conexión a Github

- Una vez que tenemos nuestra aplicación en Flask, ejecutamos *git init* para inicializar nuestro repositorio.

[ ]:
```
git init
```

- Creamos el fichero *.gitignore* para que no suba al servidor git ficheros sensibles.

[ ]:
```
venv/

*.pyc
__pycache__/

instance/

.pytest_cache/
.coverage
htmlcov/

dist/
build/
*.egg-info/
```

- También podemos utilizar https://www.toptal.com/developers/gitignore como base.
- Un ejemplo:

[ ]:
```
# Created by https://www.toptal.com/developers/gitignore/api/flask
# Edit at https://www.toptal.com/developers/gitignore?templates=flask
### Flask ###
instance/*
!instance/.gitignore
.webassets-cache
.env
...
# Environments
.venv
env/
venv/
ENV/
env.bak/
venv.bak/
```

- Luego se deberá crear un repositorio vacío en Github https://github.com/.

- Una vez creado el repositorio, generar un *token* de acceso al repositorio:
- Ir a Settings / Developer settings / Personal access tokens
  - Generate new token
- En el directorio local subiremos los cambios a Github.com.
- Creamos el commit con todos los cambios hechos hasta ahora en la aplicación:

[ ]:
```
git add .
git commit -m "Primer commit"
git push -u origin main
```

2.5 Despliegue a Render

- Ahora se puede hacer un deploy a la plataforma de aplicaciones cloud de Render.
- https://www.render.com.
- Esta plataforma ofrece un ambiente cloud en el cual podremos desplegar nuestra aplicación apenas creada.
- Basta crear una cuenta gratuita en https://dashboard.render.com/register.
- En https://www.render.com clicar en el botón *New*.
  - *Web Service*
- Conectarse al repositorio creado en Github previamente.
- Escribir un nombre para el subdominio.
- Los parámetros de configuración son los siguientes:
  - Environment Python
  - Build Command pip install -r requirements.txt
  - Start Command gunicorn app:app
- Al guardar los cambios iniciará el despliegue.
- Si el *Web Service* creado se ha guardado con el nombre *nuevo*, la aplicación se podrá ver en la dirección web:
- https://nuevo.onrender.com/

2.6 Recursos

- Configurar un ambiente virtual
  https://mothergeo-py.readthedocs.io/en/latest/development/how-to/venv-win.html
- Flask Quickstart
  https://flask.palletsprojects.com/en/latest/quickstart/
- Bootstrap Front-end open source toolkit
  https://getbootstrap.com/
- Crear un fichero .gitignore para Git
  https://www.toptal.com/developers/gitignore
- Documentación Render para el microframework Flask
  https://render.com/docs/deploy-flask