PythonBasicCourse-es006

November 9, 2021

1 Curso básico de Python

1.1 Apuntes

Curso básico de Python. Apuntes por Marcelo Horacio Fortino. Versión 1.3. Octubre 2021.

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

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 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
- González Duque, Raúl. Python para todos. (2008).
  Recuperado de: http://mundogeek.net/tutorial-python/

- 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/runestone/static/pythoned/index.html#

- 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/_/downloads/es/pdf/pdf/

- Objetivos del curso:
- Alcanzar un nivel que permita conocer las funciones y métodos que proporciona Python para comenzar a programar de forma eficiente y eficaz.

1.3 Temario

- Módulos en Python
- Módulos estándar
- La declaración *import*
- Herramientas de línea de comandos
- *Argparse*
- Módulo *platform*
- Python webserver

2 Módulos en Python

- Cuando se escribe un programa se suele dividir el mismo en varios ficheros para poder luego gestionarlo más fácilmente.
- Es común también escribir funciones que serán reutilizadas en diversas partes del programa.
- Python permite crear ficheros llamados módulos que se podrán utilizar en scripts o en las instancias interactivas del intérprete del lenguaje.

- Un módulo puede ser escrito en Python,
- Escrito en lenguaje C y cargado dinámicamente en tiempo de ejecución como por ejemplo *re (regular expression)*, o
- Ser un módulo integrado (*built-in*) que está intrínsecamente contenido en el intérprete, como el módulo *itertools*.
- En los tres casos es accedido de la misma manera utilizando la declaración *import*.

```
[ ]: import NombreDelModulo
```

- Fuente: https://docs.python.org/3/tutorial/modules.html

- Los módulos contienen definiciones y declaraciones de Python que son programadas en un fichero con extensión *.py*.
- Cuando el intérprete de Python lee el fichero fuente de un módulo primero establece algunas variables especiales como por ej. `__name__` y luego ejecuta el código fuente del mismo.
- Dentro del módulo, el nombre del mismo está disponible como el valor de la variable global `__name__`.

[ ]:
```
>>>__name__
'__main__'
```

- La variable local `__name__` en el módulo principal del programa es `__main__`.
- Cuando creamos un módulo, `__name__` toma el nombre del mismo.

[ ]:
```
# Ejemplo variables __name y __main__
# Fichero foo.py
print("Antes import")
import math

print("Antes funcion A")
def funcionA():
    print("Funcion A")

print("Antes funcion B")
def funcionB():
    print("Funcion B: {}".format(math.sqrt(100)))

print("Antes del control __name__")
if __name__ == '__main__':
    funcionA()
    funcionB()
print("Después del control __name__")
```

[ ]:
```
# Ejemplo variables __name__ y __main__
# Fichero foo_import.py
import foo
```

- Ejemplo traducido y adaptado de la pregunta *What does if* ***name*** *==* ***"main"****: do?*
- https://stackoverflow.com/questions/419163/what-does-if-name-main-do?page=1&tab=votes#tab-top

- Cuando un módulo es importado el intérprete primero busca por los módulos integrados de Python con ese nombre.
- Si no lo encuentra, busca por un fichero con ese nombre y extensión *.py* en una lista de directorios proporcionada por la variable *sys.path*.

- *Sys.path* es inicializada desde:
  - El directorio donde reside el script ejecutable (o el directorio actual si no se especifica un fichero),

  - PYTHONPATH (una lista de nombres de directorios, con la misma sintaxis que la variable de la shell PATH),
  - El valor predeterminado que depende de la instalación.
- Fuente: https://docs.python.org/3/tutorial/modules.html

[ ]:
```
# Muestra la ruta
import sys
sys.path
```

- Una vez que el módulo se ha importado, se puede determinar la ubicación donde fue encontrado mediante el atributo `__file__`:

[ ]:
```
import os
>>> os.__file__
```

- Un listado de los paquetes, módulos y bibliotecas más populares es ofrecido por la PSF:
- https://wiki.python.org/moin/UsefulModules
- Los *Top 10* módulos del *Python Package Wiki*:
- https://package.wiki/

2.1 Módulos estándar

- Python provee una biblioteca de módulos estándar denominada "Python Library Reference".
- Algunos módulos están integrados en el intérprete para proporcionar acceso a operaciones que no forman parte del núcleo del lenguaje.
- Estos módulos dependen de la plataforma subyacente, por ejemplo, el módulo *winreg* solo se proporciona en sistemas Windows.

[ ]:
```
# Show a list of all available modules
>>> help('modules')
```

- La función integrada *dir()* es utilizada para descubrir que nombres son definidos por un módulo.
- Ella devuelve una lista ordenada de strings:

[ ]:
```
import sys
dir(sys)
```

- La función *dir()* de todas maneras no devuelve la lista de las funciones y variables integradas.
- Para esto, se utiliza el módulo estándar *builtins*:

[ ]:
```
import builtins
dir(sys)
```

2.2 La declaración import

- Para importar un módulo se puede:
- Utilizar la declaración *import* como se hizo anteriormente: *import sys*

- Luego se puede llamara al objeto utilizando un punto: *sys.path*
- Importar los distintos objetos del módulo individualmente: *from sys import platform.*

[ ]:
```
from sys import platform
platform
```

- Al importar un módulo se le puede agregar un alias:

[ ]:
```
# Importa el módulo Pandas con el alias 'pd'
import pandas as pd
serie = pd.Series([1, 3, 5, 8, 13])
print(serie)
```

2.3 Módulos y paquetes

- Para organizar los módulos de modo jerárquico Python utiliza el concepto de paquetes (packages).
- Una manera de entender este concepto es pensar en paquetes como directorios y módulos como ficheros:

[ ]:
```
paquete/                          Top-level package
    __init__.py                Inicializa el paquete
    subpaquete/                   subpaquete para funcionalidad x
               __init__.py
               script_x1.py
               script_x2.py
               ...

    subpaquete2/                    subpaquete para funcionalidad y
               __init__.py
               script_y1.py
               script_y2.py
               ...
```

- Es importante notar que todos los paquetes son módulos pero no todos los módulos son paquetes.
- Los paquetes son un tipo especial de módulo: específicamente cualquier módulo que contiene un atributo `__path__` lo es.
- Todos los subpaquetes tienen un nombre separado por su paquete padre mediante un punto:

[ ]:
```
paquete.subpaquete
```

- Fuente: https://docs.python.org/es/3/tutorial/modules.html

2.4 Herramientas de línea de comandos

- El soporte que tiene Python para trabajar con los comandos y argumentos de la línea de comando es *sys.argv* incluido en la biblioteca *sys*.

- https://docs.python.org/3/library/sys.html#sys.argv

- El módulo *getopt* a su vez, ayuda a *parsear* opciones de la línea de comando y sus argumentos.
- Los scripts de Python pueden ejecutarse con la orden: *python fichero.py* o agregar argumentos y opciones o párámetros como en: *python fichero.py argumento parámetros*

[ ]:
```
import sys

print("Nombre del script: ", (sys.argv[0]))
print("Argumentos del script: ", (sys.argv[1:]))
print('Cantidad de argumentos:', len(sys.argv), 'argumentos.')
print('Lista de argumentos:', str(sys.argv))
```

2.5 Argparse

- Python utiliza para programar una interfaz para la línea de comando la biblioteca *argparse (argument parse)*.
- Esta biblioteca permite usar argumentos de posición, personalizar caracteres, y utilizar subcomandos entre otras cosas.

- La documentación se encuentra en: https://docs.python.org/es/3/library/argparse.html.
- El tutorial de la misma: https://docs.python.org/es/3/howto/argparse.html#id1.

- En la línea de comando se trabaja con argumentos separados por espacios que contienen, como en el siguiente ejemplo, un comando (*ls*), sus argumentos (*-la*), y parámetros (*/etc*):

[ ]:
```
ls -la /etc
```

[ ]:
```
# Fichero ejemplo_argparse.py
import argparse
import os
import sys

# Crea el parser
mi_parser = argparse.ArgumentParser(prog='fichero_argparse.py',␣
 ↪description='Listado del contenido del directorio',
                                    epilog='Muchas gracias')

# Agrega los argumentos
mi_parser.add_argument('Path',
                       metavar='Ruta',
                       type=str,
                       help='La ruta al directorio')
```

[ ]:
```
# Ejecuta el método parse_args()
args = mi_parser.parse_args()

dir_ruta = args.Path
```

```
if not os.path.isdir(dir_ruta):
    print('Este directorio no existe')
    sys.exit()

print('\n'.join(os.listdir(dir_ruta)))
```

[ ]:
```
python ejemplo_argparse.py directorio
```

2.6 Módulo Platform

- El módulo platform nos permite obtener información sobre el sistema operativo en el cual se está ejecutando python.
- La página de la documentación: https://docs.python.org/3/library/platform.html

[ ]:
```
import platform

print("Sistema operativo: ", platform.system())
print("Versión plataforma:", platform.release())
print("Versión SO: ", platform.version())
print("Identificación del SO: ", platform.release())
print("Arquitectura: ", platform.machine())
print("Procesador: ", platform.processor())
print("Versión del Linux Kernel: ", platform.platform())
print("-------")

if platform.system() == 'Linux':
    print("Linux Rocks")
elif platform.system() == 'Darwin':
    print("Mac")
elif platform.system() == 'Win':
    print("Windows")
print("-------")
```

- Muestra también información sobre Python:

[ ]:
```
print("Versión de Python: ", platform.python_version())
print("Compilación: ", platform.python_build())
print("Compilador: ", platform.python_compiler())
print("Implementación: ", platform.python_implementation())
print("-------")
```

2.7 Python webserver

- Python cuenta con un servidor web básico que permite hacer pruebas en un servidor web local sin tener que instalar Apache https://httpd.apache.org/ o Nginx https://www.nginx.com/.
- Para ejecutarlo, simplemente se escribe la orden:

[ ]:
```
python3 -m http.server 8000
```

[ ]:
```
python -m SimpleHTTPServer 8000 # Python 2
```

- Este servidor no es útil para ambientes de producción dado que no cuenta con funcionalidades para establecer una seguridad adecuada.

2.8 SQLAlchemy ORM

- SQLAlchemy https://www.sqlalchemy.org/ es una biblioteca de código abierto con herramientas SQL.
- Contiene su propio mapeador relacional de objetos para Python (ORM - *Object Relational Mapper*).
- Permite a los desarrolladores de aplicaciones conectarse a una base de datos relacional (PostgreSQL, Oracle, MariaDB, etc.).

- SQLAlchemy utiliza la API creada para Python DBAPI (DataBase API) para especificar cómo los módulos que se integran con bases de datos deben exponer sus interfaces a ellas.
- La documentación de esta API se encuentra en https://www.python.org/dev/peps/pep-0249/ , https://docs.sqlalchemy.org/en/14/index.html y la lista de bases de datos soportadas en https://wiki.python.org/moin/DatabaseInterfaces.

[ ]:
```
pip install SQLAlchemy # Instalación de SQLAlchemy
```

[ ]:
```
# Fichero para crear DB: fichero_sql_tablas.py
from sqlalchemy import create_engine, ForeignKey
from sqlalchemy import Column, Date, Integer, String
from sqlalchemy.ext.declarative import declarative_base
from sqlalchemy.orm import relationship, backref

engine = create_engine('sqlite:///estudiantes.db', echo=True)
Base = declarative_base()

class Estudiante(Base):
    __tablename__ = "estudiante"

    id = Column(Integer, primary_key=True)
    usuario = Column(String)
    nombre = Column(String)
    apellido1 = Column(String)
    apellido2 = Column(String)
    universidad = Column(String)

    def __init__(self, usuario, nombre, apellido1, apellido2, universidad):
        self.usuario = usuario
        self.nombre = nombre
        self.apellido1 = apellido1
        self.apellido2 = apellido2
```

```
        self.universidad = universidad

# crea la tabla
Base.metadata.create_all(engine)
```

[ ]:
```
# Fichero para inserir datos en la DB: fichero_sql_datos.py
from sqlalchemy.orm import sessionmaker
from fichero_sql_tablas import Estudiante, create_engine


engine = create_engine('sqlite:///estudiantes.db', echo=True)

# crea una sesión
Session = sessionmaker(bind=engine)
session = Session()

# Crea las instancias
usuario = Estudiante("juan", "Juan","Perez","Lopez","Complutense")
session.add(usuario)

usuario = Estudiante("Maria", "María","García","Gomez","UPC")
session.add(usuario)

usuario = Estudiante("Beatriz", "Beatriz","Suarez","Gonzalez","Carlos III")
session.add(usuario)

# commit the record the database
session.commit()
```

[ ]:
```
# Fichero para consultar datos en la DB: fichero_sql_query.py
from sqlalchemy.orm import sessionmaker
from fichero_sql_tablas import Estudiante, create_engine

engine = create_engine('sqlite:///estudiantes.db', echo=True)

# Crea una sesion
Session = sessionmaker(bind=engine)
session = Session()

# Crea objetos
for Estudiante in session.query(Estudiante).order_by(Estudiante.id):
    print(Estudiante.nombre, Estudiante.apellido1, Estudiante.apellido2)
```

[ ]:
```
# Ejemplo seleccionar objeto
for Estudiante in session.query(Estudiante).filter(Estudiante.universidad ==␣
↪'UPC'):
    print(Estudiante.nombre, Estudiante.apellido1, Estudiante.apellido2)
```