

Creado por:

Isabel Maniega

Index

1. [Socket](#)
2. [Threads](#)
3. [Serialización de objetos](#)
4. [Bases de datos](#)
5. [Documentación con Sphinx](#)
6. [Pruebas con Pytest](#)

Socket

Es una conexión cliente/servidor utilizando el módulo Socket en Python, que no es más que un canal de comunicación que permite conectar dos equipos a través de la red (normalmente cliente-servidor). La separación entre cliente servidor en estas conexiones es de tipo lógico. Esto significa que no es necesario tener corriendo el servidor en una única máquina, sino que puede estar en varias, e incluso en varios programas.

Socket es un módulo estándar del lenguaje de programación Python (y de otros muchos) que proporciona una interfaz de bajo nivel que permite conexiones TCP/IP y UDP.

Crearemos dos Script y los lanzaremos con Visual Studio Code:

- Script server.py
- Script client.py

```
In [ ]: # server.py

import socket

#instanciamos un objeto para trabajar con el socket
ser = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

#Puerto y servidor que debe escuchar
ser.bind("", 8050)

#Aceptamos conexiones entrantes con el metodo listen. Por parámetro las c
ser.listen(1)

#Instanciamos un objeto cli (socket cliente) para recibir datos
cli, addr = ser.accept()

while True:
```

```

#Recibimos el mensaje, con el metodo recv recibimos datos.
# Por parametro la cantidad de bytes para recibir
data = cli.recv(1024)
# Imprimimos el mensaje que en nos envia el cliente.
print(data)
# Si no hay datos que se pare el servidor
if not data:
    #Cerramos la instancia del socket cliente y servidor
    cli.close()
    ser.close()
    print("Conexiones cerradas")
# Devolvemos el mensaje al cliente
cli.sendall(b'mensaje recibido')

```

```

In [ ]: # client.py

#Se importa el módulo
import socket
from time import sleep

#Variables
host = '127.0.0.1'
port = 8050

#Creación de un objeto socket (lado cliente)
obj = socket.socket()

#Conexión con el servidor. Parametros: IP (puede ser del tipo 192.168.1.1)
obj.connect((host, port))
print("Conectado al servidor")

#Creamos un bucle para detener la conexion a los 10 envios:
n = 0
while n < 10:
    obj.sendall(b"Mensaje desde Cliente a Servidor >> ")
    data = obj.recv(1024)
    print('Received', repr(data))

    # Paramos el envio 5 segundos
    sleep(5)
    n += 1

#Cerramos la instancia del objeto servidor
obj.close()

#Imprimimos la palabra Adios para cuando se cierre la conexion
print("Conexión cerrada")

```

A tener en cuenta:

- En el servidor, si trabajar en un servidor Apache como XAML en tu local, deberás poner localhost. Si trabajas con IPS, no olvides cambiarla.
- La misma consideración hay que tener en cuenta para el puerto, debemos cambiarlo a nuestro puerto (creo que por defecto es 9999 o 9998).
- Si utilizas Windows y falla la conexión, revisa el Firewall de Windows y añade Python como excepción.
- Los dos ficheros (tanto cliente como servidor) deben tener extensión .py

- Recuerda que debes ejecutar el código en dos ventanas SSH diferentes para que se puedan comunicar entre ellas.

Nota: Puedes ampliar más conceptos en:

<https://docs.python.org/es/3.10/library/socket.html>

Threads

Paralelismo basado en hilos.

Thread es una unidad de ejecución que forma parte de un proceso. Un proceso puede tener varios subprocesos y todos ejecutarse al mismo tiempo. Es una unidad de ejecución en programación concurrente. Un hilo es liviano y un programador puede administrarlo de forma independiente. Le ayuda a mejorar el rendimiento de la aplicación mediante el paralelismo.

Diferencia entre proceso y Threads (hilos):

- Proceso significa que un programa está en ejecución, mientras que hilo significa un segmento de un proceso.
- Un proceso no es liviano, mientras que los subprocesos son livianos.
- Un proceso tarda más en finalizar y el hilo tarda menos en finalizar.
- El proceso requiere más tiempo para su creación, mientras que Thread requiere menos tiempo para su creación.
- Es probable que el proceso requiera más tiempo para el cambio de contexto, mientras que Threads requiere menos tiempo para el cambio de contexto.
- Un proceso está mayoritariamente aislado, mientras que los subprocesos comparten memoria.
- El proceso no comparte datos y los subprocesos comparten datos entre sí.

La forma más sencilla de usar un Thread es crear una instancia con un función de destino y llamar a `start()` para que comience a funcionar.

```
In [1]: # Importamos la librería threading
import threading

def worker():
    """
        Función con la actividad
        que debe realizar el hilo
    """
    print('Worker')

# listado de hilos creados
threads = []
for i in range(5):
    # Creamos 5 subprocesos distintos y pasamos la función que deben real
    t = threading.Thread(target=worker)
    threads.append(t)
    # Iniciamos el subproceso
```

```
t.start()
print(threads)
```

Worker
Worker
Worker
Worker
Worker

```
[<Thread(Thread-5, stopped 139742112896576)>, <Thread(Thread-6, stopped 139742112896576)>, <Thread(Thread-7, stopped 139742112896576)>, <Thread(Thread-8, stopped 139742112896576)>, <Thread(Thread-9, stopped 139742112896576)>]
```

La salida son cinco líneas con "Worker" en cada una.

Es útil poder generar un hilo y pasarle argumentos para decirle que trabajo hacer.

Cualquier tipo de objeto puede ser pasado como argumento al hilo. Este ejemplo pasa un número, que luego el hilo imprime.

```
In [2]: # Importamos la librería threading
import threading

def worker(num):
    """
    Función con la actividad
    que debe realizar el hilo
    num: int
    """
    print('Worker: %s' % num)

# listado de hilos creados
threads = []
for i in range(5):
    """
    Creamos 5 subprocesos distintos y
    pasamos la función que deben realizar,
    además los argumentos de la función (args)
    """
    t = threading.Thread(target=worker, args=(i,))
    threads.append(t)
    # Iniciamos el subproceso
    t.start()
print(threads)
```

Worker: 0
Worker: 1
Worker: 2
Worker: 3
Worker: 4

```
[<Thread(Thread-10, stopped 139742112896576)>, <Thread(Thread-11, stopped 139742112896576)>, <Thread(Thread-12, stopped 139742112896576)>, <Thread(Thread-13, stopped 139742112896576)>, <Thread(Thread-14, stopped 139742112896576)>]
```

El argumento entero ahora está incluido en el mensaje impreso por cada hilo.

Determinar el hilo actual

Usar argumentos para identificar o nombrar el hilo es engorroso e innecesario. Cada instancia Thread tiene un nombre con un valor predeterminado que se puede cambiar cuando se crea el hilo. Nombrar hilos es útil en procesos de servidor con múltiples hilos de servicio manejando diferentes operaciones.

```
In [3]: import threading
import time

def worker():
    """
    Imprimimos el nombre del actual subproceso
    que esta ejecutándose:
    threading.current_thread().getName()
    """
    print(threading.current_thread().getName(), 'Starting')
    time.sleep(0.2)
    print(threading.current_thread().getName(), 'Exiting')

def my_service():
    """
    Imprimimos el nombre del actual subproceso
    que esta ejecutándose:
    threading.current_thread().getName()
    """
    print(threading.current_thread().getName(), 'Starting')
    time.sleep(0.3)
    print(threading.current_thread().getName(), 'Exiting')

# El argumento name nos permite nombrar el hilo
# Creamos dos subprocesos con funciones distintas
t = threading.Thread(name='my_service', target=my_service)
w = threading.Thread(name='worker', target=worker)
# Subproceso creado con nombre por defecto
w2 = threading.Thread(target=worker)

w.start()
w2.start()
t.start()
```

```
worker Starting
Thread-15 Starting
my_service Starting
worker Exiting
Thread-15 Exiting
my_service Exiting
```

La salida de depuración incluye el nombre del hilo actual en cada línea. Las líneas con "Thread-X" en la columna de nombre de hilo corresponden al hilo sin nombre w2.

Hilos de Daemon vs. No-Daemon

Hasta este punto, los programas de ejemplo han esperado implícitamente para salir hasta que todos los hilos hayan completado su trabajo. A veces los programas generan un hilo como un demonio que se ejecuta sin bloquear el programa principal de salir. El uso de hilos de demonio es útil para servicios donde puede que no haya una manera

fácil de interrumpir el hilo, o donde dejar que el el hilo muera en medio de su trabajo, no pierde ni corrompe los datos (por ejemplo, un hilo que genera «latidos del corazón» para una herramienta de monitoreo de servicio). Para marcar un hilo como demonio, pasa `daemon=True` al construirlo o llama a su método `set_daemon()` con `True`. El valor predeterminado es que los subprocesos no sean demonios.

```
In [4]: import threading
import time

def daemon():
    print(threading.current_thread().getName(), 'Starting')
    time.sleep(0.2)
    print(threading.current_thread().getName(), 'Exiting')

def non_daemon():
    print(threading.current_thread().getName(), 'Starting')
    print(threading.current_thread().getName(), 'Exiting')

"""
    Iniciamos dos subprocesos:
    - name 'daemon' como demonio a True
    - name non_daemon como demonio a False
"""
d = threading.Thread(name='daemon', target=daemon, daemon=True)
t = threading.Thread(name='non-daemon', target=non_daemon)

d.start()
t.start()
```

```
daemon Starting
non-daemon Starting
non-daemon Exiting
daemon Exiting
```

La salida no incluye el mensaje "Exiting" del hilo demonio, ya que todos los hilos no demonio (incluyendo el hilo principal) terminan antes de que el hilo demonio se despierte de la llamada `sleep()`.

Para esperar hasta que un subproceso demonio haya completado su trabajo, usa el método `join()`.

```
In [5]: import threading
import time

def daemon():
    print(threading.current_thread().getName(), 'Starting')
    time.sleep(0.2)
    print(threading.current_thread().getName(), 'Exiting')

def non_daemon():
    print(threading.current_thread().getName(), 'Starting')
    print(threading.current_thread().getName(), 'Exiting')
```

```

"""
    Iniciamos dos subprocessos:
    - name 'daemon' como demonio a True
    - name non_daemon como demonio a False
"""
d = threading.Thread(name='daemon', target=daemon, daemon=True)
t = threading.Thread(name='non-daemon', target=non_daemon)

d.start()
t.start()

d.join()
t.join()

```

```

daemon Starting
non-daemon Starting
non-daemon Exiting
daemon Exiting

```

Por defecto, `join()` bloquea indefinidamente. También es posible pasar un valor flotante que represente el número de segundos a esperar por el hilo para convertirse en inactivo. Si el hilo no se completa dentro del período de tiempo de espera, `join()` retorna de todos modos.

```

In [6]: import threading
import time

def daemon():
    print(threading.current_thread().getName(), 'Starting')
    time.sleep(0.2)
    print(threading.current_thread().getName(), 'Exiting')

def non_daemon():
    print(threading.current_thread().getName(), 'Starting')
    print(threading.current_thread().getName(), 'Exiting')

"""
    Iniciamos dos subprocessos:
    - name 'daemon' como demonio a True
    - name non_daemon como demonio a False
"""
d = threading.Thread(name='daemon', target=daemon, daemon=True)
t = threading.Thread(name='non-daemon', target=non_daemon)

d.start()
t.start()

d.join(0.1)
print('d.is_alive()', d.is_alive())
t.join()

```

```

daemon Starting
non-daemon Starting
non-daemon Exiting
d.is_alive() True
daemon Exiting

```

Dado que el tiempo de espera transcurrido es menor que la cantidad de tiempo que el hilo demonio duerme, el hilo sigue «vivo» después de join() retorna.

Señalización Entre Hilos

Aunque el punto de usar múltiples hilos es ejecutar separadamente operaciones al mismo tiempo, hay momentos en que es importante ser capaz de sincronizar las operaciones en dos o más hilos. Los objetos evento son una forma sencilla de comunicarse entre hilos de forma segura. Un Event gestiona una bandera interna que las personas que llaman pueden controlar con los métodos set() y clear(). Otros hilos pueden usar wait() para pausar hasta que se establezca la bandera, bloqueando efectivamente el avance hasta que se permita continuar.

```
In [7]: import threading
import time

def wait_for_event(e):
    """Wait for the event to be set before doing anything"""
    print(f'({threading.current_thread().getName()}) wait_for_event starting')
    event_is_set = e.wait()
    print(threading.current_thread().getName(), f'event set: {event_is_set}')

def wait_for_event_timeout(e, t):
    """Wait t seconds and then timeout"""
    while not e.is_set():
        print(f'({threading.current_thread().getName()}) wait_for_event_timeout starting')
        event_is_set = e.wait(t)
        print(threading.current_thread().getName(), f'event set: {event_is_set}')
        if event_is_set:
            print(f'({threading.current_thread().getName()}) processing event')
        else:
            print(f'({threading.current_thread().getName()}) doing other work')

e = threading.Event()
t1 = threading.Thread(name='block', target=wait_for_event, args=(e,))
t1.start()

t2 = threading.Thread(name='nonblock', target=wait_for_event_timeout, args=(e, 0.3))
t2.start()

print(f'({threading.current_thread().getName()}) Waiting before calling Event.set()')
time.sleep(0.3)
e.set()
print(f'({threading.current_thread().getName()}) Event is set')

(block) wait_for_event starting
(nonblock) wait_for_event_timeout starting
(MainThread) Waiting before calling Event.set()
(MainThread) Event is set
(nonblock) processing event
(block) event set: True
```

El método wait_for_event_timeout() toma un argumento que representa el número de segundos que el evento espera antes de que se agote el tiempo de espera. Devuelve

un booleano indicando si el evento está configurado o no, para que la persona que llama sepa por qué `wait()` regresó. El método `is_set()` puede ser usado por separado en el evento sin miedo a bloquear.

En este ejemplo, `wait_for_event_timeout()` comprueba el estatus del evento sin bloqueo indefinido. El `wait_for_event()` bloquea en la llamada a `wait()`, que no regresa hasta que el estado del evento cambie.

Sincronizar hilos

Además de usar Events, otra forma de sincronizar los hilos son a través del uso de un objeto Condition. Porque Condition utiliza un Lock, se puede vincular a un recurso compartido, permitiendo que múltiples hilos esperen a que el recurso sea actualizado. En este ejemplo, los hilos `consumer()` esperan el Condition que se establezca antes de continuar. El hilo `producer()` es responsable de establecer la condición y notificar a los otros hilos que pueden continuar.

```
In [8]: import threading
import time

def consumer(cond):
    """wait for the condition and use the resource"""
    print(f'({threading.current_thread().getName()} ) Starting consumer t
    with cond:
        cond.wait()
        print(f'({threading.current_thread().getName()} ) Resource is ava

def producer(cond):
    """set up the resource to be used by the consumer"""
    print(f'({threading.current_thread().getName()} ) Starting producer t
    with cond:
        print(f'({threading.current_thread().getName()} ) Making resource
        cond.notifyAll()

condition = threading.Condition()
c1 = threading.Thread(name='c1', target=consumer, args=(condition,))
c2 = threading.Thread(name='c2', target=consumer, args=(condition,))
p = threading.Thread(name='p', target=producer, args=(condition,))

c1.start()
time.sleep(0.2)
c2.start()
time.sleep(0.2)
p.start()
```

```
(c1 ) Starting consumer thread
(c2 ) Starting consumer thread
(p ) Starting producer thread
(p ) Making resource available
(c1 ) Resource is available to consumer
(c2 ) Resource is available to consumer
```

Los hilos usan `with` para adquirir el bloqueo asociado con la `Condition`. Usando los métodos `capture()` y `release()` explícitamente también funcionan.

Las barreras son otro mecanismo de sincronización de hilos. Una `Barrier` establece un punto de control y todos los hilos participantes bloquean hasta que todas las «partes» participantes hayan alcanzado ese punto. Permite que los hilos se inicien por separado y luego se pause hasta que todos están listos para continuar.

```
In [9]: import threading
import time

def worker(barrier):
    print(threading.current_thread().name, 'waiting for barrier with {} o
worker_id = barrier.wait()
    print(threading.current_thread().name, 'after barrier', worker_id)

NUM_THREADS = 3

barrier = threading.Barrier(NUM_THREADS)

threads = [threading.Thread(name='worker-%s' % i, target=worker, args=(ba
for i in range(NUM_THREADS))]

for t in threads:
    print(t.name, 'starting')
    t.start()
    time.sleep(0.1)

for t in threads:
    t.join()
```

```
worker-0 starting
worker-0 waiting for barrier with 0 others
worker-1 starting
worker-1 waiting for barrier with 1 others
worker-2 starting
worker-2 waiting for barrier with 2 others
worker-2 after barrier 2
worker-1 after barrier 1
worker-0 after barrier 0
```

En este ejemplo, la `Barrier` está configurada para bloquear hasta que tres hilos estén esperando. Cuando se cumple la condición, todos los hilos se liberan más allá del punto de control al mismo tiempo. Los valores de retorno de `wait()` indica el número de la parte que está siendo liberada, y puede usarse para limitar algunos subprocesos de realizar una acción como limpiar un recurso compartido.

Colas o Queue (FIFO)

FIFO: primero en entrar, primero en salir

```
In [10]: import queue

q1 = queue.Queue(5)
```

```
q1.put(1)
q1.put(2)
q1.put(3)
q1.__dict__
```

```
Out[10]: {'maxsize': 5,
         'queue': deque([1, 2, 3]),
         'mutex': <unlocked _thread.lock object at 0x7f185c4a4240>,
         'not_empty': <Condition(<unlocked _thread.lock object at 0x7f185c4a4240>, 0)>,
         'not_full': <Condition(<unlocked _thread.lock object at 0x7f185c4a4240>, 0)>,
         'all_tasks_done': <Condition(<unlocked _thread.lock object at 0x7f185c4a4240>, 0)>,
         'unfinished_tasks': 3}
```

```
In [11]: q1.get()
```

```
Out[11]: 1
```

```
In [12]: q1.get()
```

```
Out[12]: 2
```

```
In [13]: q1.__dict__
```

```
Out[13]: {'maxsize': 5,
         'queue': deque([3]),
         'mutex': <unlocked _thread.lock object at 0x7f185c4a4240>,
         'not_empty': <Condition(<unlocked _thread.lock object at 0x7f185c4a4240>, 0)>,
         'not_full': <Condition(<unlocked _thread.lock object at 0x7f185c4a4240>, 0)>,
         'all_tasks_done': <Condition(<unlocked _thread.lock object at 0x7f185c4a4240>, 0)>,
         'unfinished_tasks': 3}
```

Compartir datos entre threads

```
In [17]: import threading
import queue

def busqueda1(num,cola):
    for i in range(5):
        if num == i:
            cola.put("Busqueda1 confirmó el número")

def busqueda2(num,cola):
    for i in range(5,10):
        if(num == i):
            cola.put("Busqueda2 confirmó el número")

numero = 3
cola = queue.Queue()

thread1 = threading.Thread(target=busqueda1,args=(numero,cola))
thread2 = threading.Thread(target=busqueda2,args=(numero,cola))

thread1.start()
```

```

thread2.start()

thread1.join()
thread2.join()

print('Resultado: ', cola.get())

```

Resultado: Busqueda1 confirmó el número

```

In [18]: import queue
import random
import threading
import time

def producer(queue, event):
    """Pretend we're getting a number from the network."""
    while not event.is_set():
        message = random.randint(1, 10)
        print("%s got message: %s" % (threading.current_thread().name, message))
        queue.put(message)
        event.wait()

    print(f"{threading.current_thread().name} received event. Exiting")

def consumer(queue, event):
    """Pretend we're saving a number in the database."""
    print('Valores en la cola: ', list(queue.queue))
    while not event.is_set() or not queue.empty():
        message = queue.get()
        print("%s storing message: %s (size=%d)" % (threading.current_thread().name, message, queue.qsize()))
        event.wait()

    print(f"{threading.current_thread().name} received event. Exiting")

cola = queue.Queue()
event = threading.Event()

thread1 = threading.Thread(name='Producer1', target=producer, args=(cola, event))
thread3 = threading.Thread(name='Producer2', target=producer, args=(cola, event))
thread2 = threading.Thread(name='Consumer1', target=consumer, args=(cola, event))
thread4 = threading.Thread(name='Consumer2', target=consumer, args=(cola, event))

thread1.start()
thread3.start()
thread2.start()
thread4.start()

time.sleep(0.1)
print("Main: about to set event")
event.set()

```

```
Producer1 got message: 10
Producer2 got message: 9
Valores en la cola: [10, 9]
Consumer1 storing message: 10 (size=1)
Valores en la cola: [9]
Consumer2 storing message: 9 (size=0)
Main: about to set event
Consumer2 received event. Exiting
Consumer1 received event. Exiting
Producer2 received event. Exiting
Producer1 received event. Exiting
```

Serialización de objetos

Módulo Json

JSON se ha convertido en el estándar por defecto para el intercambio de información.

Sus usos son:

- Transportar datos Tal vez esté
- Recopilando información a través de una API
- Almacenando sus datos en una base de datos de documentos.

Ejemplo de Json:

```
{
    "firstName": "Jasmine",
    "lastName": "Doe",
    "hobbies": ["running", "cooking", "singing"],
    "age": 35,
    "children": [
        {
            "firstName": "Alice",
            "age": 6
        },
        {
            "firstName": "Bob",
            "age": 8
        }
    ]
}
```

Esto nos recuerda a los diccionarios de Python, por lo tanto, ya sabemos que lo podemos recorrer como clave: valor.

Para poder manejar estos datos usaremos el paquete **json**

Concepto básico:

- **Serialización:** Consiste en convertir un objeto de Python (normalmente una lista o diccionario) en un string.
- **Deserialización:** Consiste en convertir un string en un objeto de Python (normalmente una lista o diccionario).

```
In [19]: # Ejemplo de serialización
```

```
import json

data = {
    "presidente": {
        "nombre": "Zaphod Beeblebrox",
        "especie": "Betelgeusian"
    }
}

json_string = json.dumps(data)
json_string
```

```
Out[19]: '{"presidente": {"nombre": "Zaphod Beeblebrox", "especie": "Betelgeusia"}}
```

```
In [20]: type(json_string)
```

```
Out[20]: str
```

```
In [21]: json_string["presidente"]
```

```
# TypeError: string indices must be integers
# No podemos recorrerlo para ello será necesario decodificarlo
```

```
-----
-
TypeError                                Traceback (most recent call last)
Cell In[21], line 1
----> 1 json_string["presidente"]
      3 # TypeError: string indices must be integers
      4 # No podemos recorrerlo para ello será necesario decodificarlo

TypeError: string indices must be integers
```

```
In [22]: # La opción de dumps nos sirve para escribirlo en un archivo de formato J
```

```
with open("archivo_data.json", "w") as write_file:
    json.dump(data, write_file)
```

```
In [23]: # Indent nos permite añadir un salto a cada componente del Json:
```

```
with open("archivo_data.json", "w") as write_file:
    json.dump(data, write_file, indent=2)
```

```
In [24]: # sort_keys= True: la salida de los diccionarios se ordenará por clave:
# Salida esperada: {"age": 30, "city": "New York", "name": "John"}

data = {
    "name": "John",
    "age": 30,
    "city": "New York"
}

with open("archivo_data.json", "w") as write_file:
    json.dump(data, write_file, indent=2, sort_keys=True)
```

```
In [25]: # Ejemplo de deserialización
# Convertir datos codificados en JSON en objetos de Python.

json_decoded = json.loads(json_string)
json_decoded
```

```
Out[25]: {'presidente': {'nombre': 'Zaphod Beeblebrox', 'especie': 'Betelgeusia
n'}}
```

```
In [26]: type(json_decoded)

# Con loads podemos recorrer el diccionario y extraer la información.
```

```
Out[26]: dict
```

```
In [27]: json_decoded['presidente']
```

```
Out[27]: {'nombre': 'Zaphod Beeblebrox', 'especie': 'Betelgeusian'}
```

```
In [28]: json_decoded['presidente']['nombre']
```

```
Out[28]: 'Zaphod Beeblebrox'
```

Python JSON dic object list o tupla array str string int y float number True true False false None null

Bases de datos

SQLite

SQLite también es un gestor de bases de datos relacional pero con objetivos muy diferentes a MySQL, SQLServer, Oracle etc. Este gestor de base de datos tiene por objetivo ser parte de la misma aplicación con la que colabora, es decir no cumple los conceptos de cliente y servidor.

Para entender sus usos podemos dar algunos ejemplos donde se utiliza el gestor SQLite:

- Firefox usa SQLite para almacenar los favoritos, el historial, las cookies etc.
- También el navegador Opera usa SQLite.
- La aplicación de comunicaciones Skype de Microsoft utiliza SQLite
- Los sistemas operativos Android y iOS adoptan SQLite para permitir el almacenamiento y recuperación de datos.

SQLite es Open Source y se ha instalado por defecto con Python, es decir forma parte de la biblioteca estándar, no tenemos que instalar ningún módulo con pip.

Si nuestra aplicación necesita almacenar gran cantidad de información local con cierta estructura el empleo de SQLite es nuestra principal opción.

Creamos una tabla llamada Articulos donde tiene 3 columnas:

- Código que será la clave primaria para relacionar con otras tablas, es de tipo entero.
- Descripción de tipo texto
- Precio de tipo número.

articulos codigo descripcion precio ----- 1 boligrafo 1.2 2 libro 20 3 lapicero 0.85

Realizaremos el CRUB (Create Read Update Delete):

```
In [30]: # Importamos las librería sqlite3:
import sqlite3

# Creamos un archivo como extensión .db donde se almacenará la información
conexion=sqlite3.connect("database.db")
try:
    # Creamos la tabla anteriormente explicada:
    conexion.execute("""CREATE TABLE articulos (codigo integer primary key
                        descripcion text, precio real)""")
    print("se creo la tabla articulos")
except sqlite3.OperationalError:
    # si ocurre un error al crear la tabla nos mostrará el siguiente error
    print("La tabla articulos ya existe")
conexion.close()
```

La tabla articulos ya existe

```
In [31]: # Crear:
# Insertamos información en la tabla creada anteriormente de articulos:

conexion=sqlite3.connect("database.db")
conexion.execute("INSERT INTO articulos(descripcion,precio) VALUES (?,?)")
conexion.execute("INSERT INTO articulos(descripcion,precio) VALUES (?,?)")
conexion.execute("INSERT INTO articulos(descripcion,precio) VALUES (?,?)")
conexion.commit()
conexion.close()
```

```
In [32]: # Read
# Leer todos los datos de la tabla:

conexion = sqlite3.connect("database.db")
cursor = conexion.execute("SELECT codigo,descripcion,precio FROM articulos")
for fila in cursor:
    print(fila)
conexion.close()
```

```
(1, 'boligrafo', 1.2)
(2, 'libro', 20.0)
(3, 'lapicero', 0.85)
```



```
In [34]: # Read
# Leer sólo un dato:

conexion = sqlite3.connect("database.db")

# Recogemos el código que el usuario quiere buscar:
codigo = int(input("Ingrese el código de un artículo:"))

# filtramos por el código y que nos muestre la descripción y precio:
cursor = conexion.execute("SELECT descripcion,precio FROM articulos WHERE
# El método fetchone de la clase Cursor retorna una tupla
# con la fila de la tabla que coincide con el código ingresado o retorna
fila = cursor.fetchone()

if fila != None:
    print(fila)
else:
    print("No existe un artículo con dicho código.")
conexion.close()
```

No existe un artículo con dicho código.

```
In [36]: # Recuperar varias filas con una coincidencia en este caso por precio:

conexion = sqlite3.connect("database.db")

# Filtrar los datos por el precio que elija el usuario:
precio = float(input("Ingrese un precio:"))

# Filtrar por dicho valor que sea inferior:
cursor = conexion.execute("SELECT descripcion FROM articulos WHERE precio

# Llamamos al método 'fetchall' de la clase Cursor y
# nos retorna una lista con todas las filas de la tabla
# que cumplen la condición de tener un precio inferior al ingresado:
filas=cursor.fetchall()

if len(filas)>0:
    for fila in filas:
        print(fila)
else:
    print("No existen artículos con un precio menor al ingresado.")
conexion.close()
```

```
('boligrafo',)
('lapicero',)
```

```
In [37]: # Update
# Actualizar alguno de los datos buscando por el código

conexion = sqlite3.connect("database.db")

# Buscar el código y añadir el precio por el usuario:
codigo = int(input("Ingrese un código:"))
precio = float(input("Ingresa un precio:"))

# Actualizar el precio por codigo :
try:
    cursor = conexion.execute("UPDATE articulos SET precio=? WHERE codigo
# Para que se realice el cambio añadimos .commit
```

```
conexion.commit()
except Exception as e:
    print('Error %s: ' % str(e))

conexion.close()
```

```
In [38]: # Delete
# Eliminar un determinado dato

conexion = sqlite3.connect("database.db")

# Eliminar el código que el usuario quiera.
codigo = int(input("Ingrese un código:"))

# Eliminar la fila por el codigo:
try:
    cursor = conexion.execute("DELETE FROM articulos WHERE codigo=?;", (c
    # Para que se realice el cambio añadimos .commit
    conexion.commit()
except Exception as e:
    print('Error %s: ' % str(e))

conexion.close()
```

Documentación con Sphinx

Para documentar nuestro código apropiadamente, vamos a usar la herramienta adecuada. Sphinx es un paquete que se encargará del trabajo pesado de organizar la información, extraerla de los lugares adecuados, y presentarla en un bonito HTML o PDF (o en algún otro formato). Esto te permitirá concentrarte más en el contenido que en el formato. Sphinx es, además, el estándar para elaborar la documentación técnica en Python.

```
In [40]: # pip install sphinx
```

```
In [42]: # pip install sphinx_rtd_theme
```

sphinx_rtd_theme nos permite mostrar la vista como documentación de Python.

Creemos la carpeta de nuestro proyecto Documentation para este fin, ahora nos iremos a la consola:

1. Crea la carpeta doc: `mkdir docs`
2. Entra en la carpeta doc: `cd docs`
3. Vemos el contenido de la carpeta: `ls` (Linux) ó `dir` (Windows), observamos que está vacío.
4. Ejecutamos el siguiente comando: `sphinx-quickstart`

Nota: Fijarse que estamos dentro del entorno virtual y comprobar con `pip list` que tenemos instalado Sphinx.

5. Daremos por defecto los campos que nos pide excepto el nombre del proyecto, autor y versión:

```
In [2]: from IPython import display
display.Image("images/sphinx_1.png")
```

```
Out[2]: (env) isabelmaniega@isabelmaniega:~/Documentos/Python_Básico_cas$ cd Documentation/
(env) isabelmaniega@isabelmaniega:~/Documentos/Python_Básico_cas/Documentation$ mkdir docs
(env) isabelmaniega@isabelmaniega:~/Documentos/Python_Básico_cas/Documentation$ ls
docs
(env) isabelmaniega@isabelmaniega:~/Documentos/Python_Básico_cas/Documentation$
```

```
In [2]: display.Image('images/sphinx_2.png')
```

```
Out[2]: (env) isabelmaniega@isabelmaniega:~/Documentos/Python_Básico_cas/Documentation/docs$ sphinx-quickstart
Bienvenido a la utilidad de inicio rápido de Sphinx 7.1.2.

Ingrese los valores para las siguientes configuraciones (solo presione Entrar para
aceptar un valor predeterminado, si se da uno entre paréntesis).

Ruta raíz seleccionada: .

Tiene dos opciones para colocar el directorio de compilación para la salida de Sphinx.
O usas un directorio "build" dentro de la ruta raíz, o separas
directorios "fuente" y "compilación" dentro de la ruta raíz.
> Separar directorios fuente y compilado (y/n) [n]: y

El nombre del proyecto aparecerá en varios lugares en la documentación construida.
> Nombre de proyecto: Example_sphinx
> Autor(es): Isabel Maniega
> Liberación del proyecto []: v0.1

Si los documentos deben escribirse en un idioma que no sea inglés,
puede seleccionar un idioma aquí por su código de idioma. Sphinx entonces
traducir el texto que genera a ese idioma.

Para obtener una lista de códigos compatibles, vea
https://www.sphinx-doc.org/en/master/usage/configuration.html#confval-language.
> Lenguaje del proyecto [en]:

Creando archivo /home/isabelmaniega/Documentos/Python_Básico_cas/Documentation/docs/source/conf.py.
Creando archivo /home/isabelmaniega/Documentos/Python_Básico_cas/Documentation/docs/source/index.rst.
Creando archivo /home/isabelmaniega/Documentos/Python_Básico_cas/Documentation/docs/Makefile.
Creando archivo /home/isabelmaniega/Documentos/Python_Básico_cas/Documentation/docs/make.bat.

Terminado: se ha creado una estructura de directorio inicial.

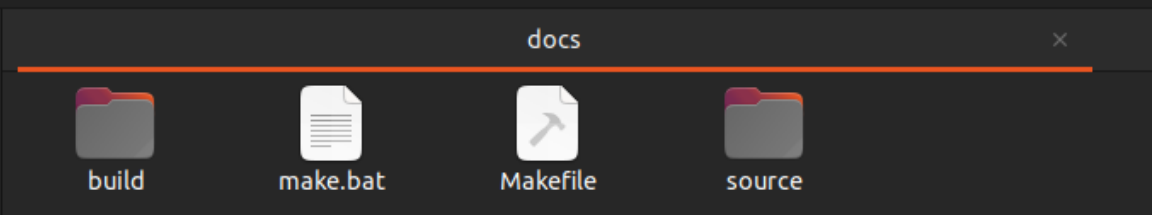
Ahora debe completar su archivo maestro /home/isabelmaniega/Documentos/Python_Básico_cas/Documentation/docs/source/index.rst y crear otros archivos fuente
de documentación. Use el archivo Makefile para compilar los documentos, así ejecute el comando:
    make builder
donde "builder" es uno de los constructores compatibles, por ejemplo, html, latex o linkcheck.

(env) isabelmaniega@isabelmaniega:~/Documentos/Python_Básico_cas/Documentation/docs$
```

Observamos que se nos han creado varios archivos:

```
In [3]: display.Image('images/sphinx_3.png')
```

```
Out[3]:
```



Podemos modificar el diseño de la página de documentación, aquí puedes encontrar los distintos aspectos:

- <https://sphinx-themes.org/#themes>
- <https://www.writethedocs.org/guide/tools/sphinx-themes/>

Para ello vamos a usar el aspecto 'Read the Docs', modificamos en la carpeta source/conf.py la línea:

línea 27:

```
html_theme = 'alabaster'
```

por:

```
html_theme = 'sphinx_rtd_theme'
```

Sphinx utiliza como source archivos tipo .rst creados con un lenguaje de marcas denominado reStructuredText (o simplemente reST). Se trata de un tipo de formato similar a markdown, pero con una serie de marcas ampliadas especialmente diseñadas para facilitar el trabajo de documentación automática de código. Puedes encontrar más información en la documentación de Sphinx.

<https://www.sphinx-doc.org/en/master/usage/restructuredtext/basics.html>

Creamos un archivo Installation.rst:

```
In [ ]: # Installation
# =====

# rst file example
# -----

# Esto es un ejemplo de rst.

# Para ejecutar nuestro proyecto pondremos:

# ``python <nombre archivo>.py``

# Esto está en negrita y esto está en cursiva

# El código se marca así:

# ``import sphinx``

# +-----+-----+
# | Columna 1 | Columna 2 |
# +=====+=====+
# | Dato 1    | Dato A    |
# +-----+-----+
# | Dato 2    | Dato B    |
# +-----+-----+
# | Dato 3    | Dato C    |
# +-----+-----+

# Tabla 1

# * Esto es una lista con viñetas
# * Si tiene dos items, el segundo
#   item usa dos líneas.

# 1. Esto es una lista numerada.
# 2. Este es el segundo item de la lista numerada.
```

```
In [4]: display.Image('./images/rst.png')
```

Out[4]:

```
(env) isabelmaniega@isabelmaniega:~/Documentos/Python_Básico_cas/Documentation/docs/source$ cat Installation.rst
Installation
=====

rst file example
-----

Esto es un ejemplo de rst.

Para ejecutar nuestro proyecto pondremos:

``python <nombre archivo>.py``

**Esto está en negrita** y *esto está en cursiva*

El código se marca así:

``import sphinx``

+-----+
| Columna 1 | Columna 2 |
+-----+
| Dato 1    | Dato A    |
+-----+
| Dato 2    | Dato B    |
+-----+
| Dato 3    | Dato C    |
+-----+

Tabla 1

* Esto es una lista con viñetas
* Si tiene dos items, el segundo
  item usa dos líneas.

1. Esto es una lista numerada.
2. Este es el segundo item de la lista numerada.
(env) isabelmaniega@isabelmaniega:~/Documentos/Python_Básico_cas/Documentation/docs/source$
```

Una vez creado el archivo debemos añadirlo al archivo index.rst para que nos lo agregue al menú:

```
In [ ]: # .. Example_sphinx documentation master file, created by
#       sphinx-quickstart on Tue Jan 23 14:47:58 2024.
#       You can adapt this file completely to your liking, but it should at
#       contain the root `toctree` directive.

# Welcome to Example_sphinx's documentation!
# =====

# .. toctree::
#   :maxdepth: 2
#   :caption: Contents:

#   Installation # Añadimos el archivo .rst anteriormente creado


# Indices and tables
# =====

# * :ref:`genindex`
# * :ref:`modindex`
# * :ref:`search`
```

Volvemos a la carpeta docs y ejecutamos:

make html

```
In [5]: display.Image('./images/sphinx_4.png')
```

```

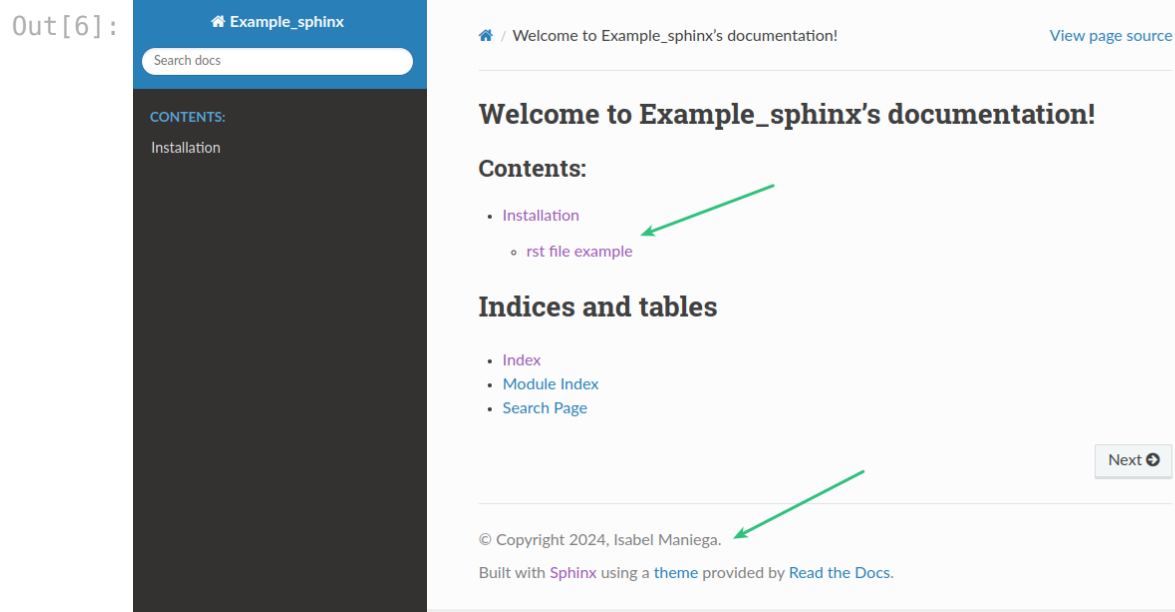
Out[5]: (env) isabelmaniega@isabelmaniega:~/Documentos/Python_Básico_cas/Documentation/docs$ make html
Ejecutando Sphinx v7.1.2
cargando el ambiente pickled... hecho
compilando [mo]: los objetivos para 0 los archivos po que estan desactualizados
escribiendo salida...
compilando [html]: los objetivos para 1 los archivos fuentes que estan desactualizados
actualizando ambiente: 0añadido, 1 cambiado, 0 removido
leyendo fuentes... [100%] Installation
buscando por archivos no actualizados... no encontrado
preparando ambiente... hecho
verificando consistencia... hecho
preparando documentos... hecho
copying assets... copiar archivos estáticos... hecho
copiando archivos extras... hecho
hecho
escribiendo salida... [100%] index
generando índices... genindex hecho
escribiendo páginas adicionales... search hecho
volcar el índice de búsqueda en English (code: en)... hecho
volcar inventario de objetos... hecho
construir exitoso.

Las páginas HTML están en build/html.
(env) isabelmaniega@isabelmaniega:~/Documentos/Python_Básico_cas/Documentation/docs$

```

Vamos hasta la carpeta docs/build/html hacemos click sobre el archivo index.html, nos muestra la documentación que hemos agregado a nuestro manual:

```
In [6]: display.Image('./images/sphinx_5.png')
```



Vemos que la información que hemos creado anteriormente se visualiza en contenido, si seleccionamos la instalación, nos muestra la información creada en el archivo Installation.rst:m

```
In [7]: display.Image('./images/sphinx_6.png')
```

Out[7]:

Example_sphinx

Search docs

CONTENTS:

- Installation
- rst file example

Installation

rst file example

Esto es un ejemplo de rst.

Para ejecutar nuestro proyecto pondremos:

```
python <nombre_archivo>.py
```

Esto está en negrita y *esto está en cursiva*

El código se marca así:

```
import sphinx
```

Columna 1	Columna 2
Dato 1	Dato A
Dato 2	Dato B
Dato 3	Dato C

Tabla 1

- Esto es una lista con viñetas
- Si tiene dos items, el segundo item usa dos líneas.

1. Esto es una lista numerada.
2. Este es el segundo item de la lista numerada.

Previous

Para más información: <https://www.sphinx-doc.org/en/master/index.html>

Pruebas con Pytest

Testear el código puedes obtener una amplia variedad de beneficios. Aumenta la confianza, que el código funcione como se espera, y garantizar que los cambios en el código no provocarán regresiones. Escribir y mantener pruebas es un trabajo duro, por lo que debes aprovechar todas las herramientas a tu disposición para hacerlo lo menos complicado posible. pytest es una de las mejores herramientas que puede utilizar para aumentar la productividad de sus pruebas.

Qué beneficios ofrece pytest:

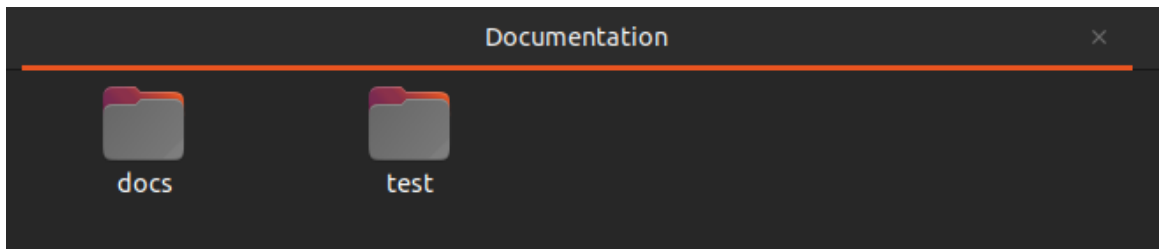
- Puede ejecutar varias pruebas en paralelo, lo que reduce el tiempo de ejecución del conjunto de pruebas.
- Tiene su propia forma de detectar el archivo de prueba y las funciones de prueba automáticamente, si no se menciona explícitamente.
- Nos permite omitir un subconjunto de pruebas durante la ejecución.
- Nos permite ejecutar un subconjunto de todo el conjunto de pruebas.
- Es gratuito y de código abierto.
- Debido a su sintaxis simple, es muy fácil comenzar con pytest.

```
In [9]: # pip install pytest
```

Creamos dentro de la carpeta de Documentación creada en la parte de Sphinx, una carpeta llamada test:

```
In [3]: display.Image('./images/test_1.png')
```

```
Out[3]:
```



Pytest le permite escribir funciones de prueba utilizando declaraciones estándar de afirmación de Python, lo que hace que sus pruebas sean limpias y legibles. Para crear una prueba, simplemente defina una función con un nombre que comience con test_y use aserciones para verificar si se cumple el comportamiento esperado. Aquí hay un ejemplo simple:

```
In [ ]: ## test.py

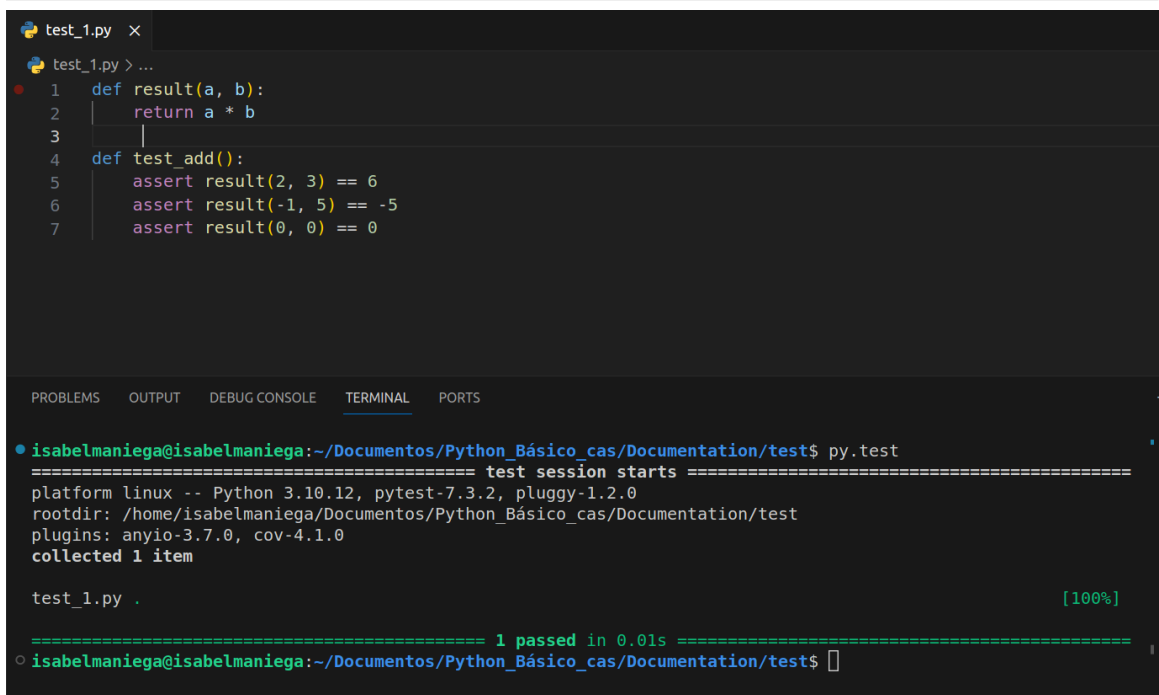
# def result(a, b):
#     return a * b

# def test_result():
#     assert result(2, 3) == 6
#     assert result(-1, 5) == -5
#     assert result(0, 0) == 0
```

Ejecutamos en la consola el script con py.test, como se ve en la imagen siguiente:

```
In [4]: display.Image('./images/test_2.png')
```

```
Out[4]:
```



Observamos que todas las pruebas que hemos realizado pasan el test sin ningún error.

Accesorios Pytest

Los accesorios en pytest brindan una forma conveniente de configurar y eliminar recursos reutilizables, como conexiones de bases de datos, archivos temporales o datos de prueba. Le ayudan a mantener un conjunto de pruebas limpio y modular. Para crear un accesorio, use el decorador `@pytest.fixture` que se muestra en el ejemplo a continuación.

```
In [ ]: # test_2.py

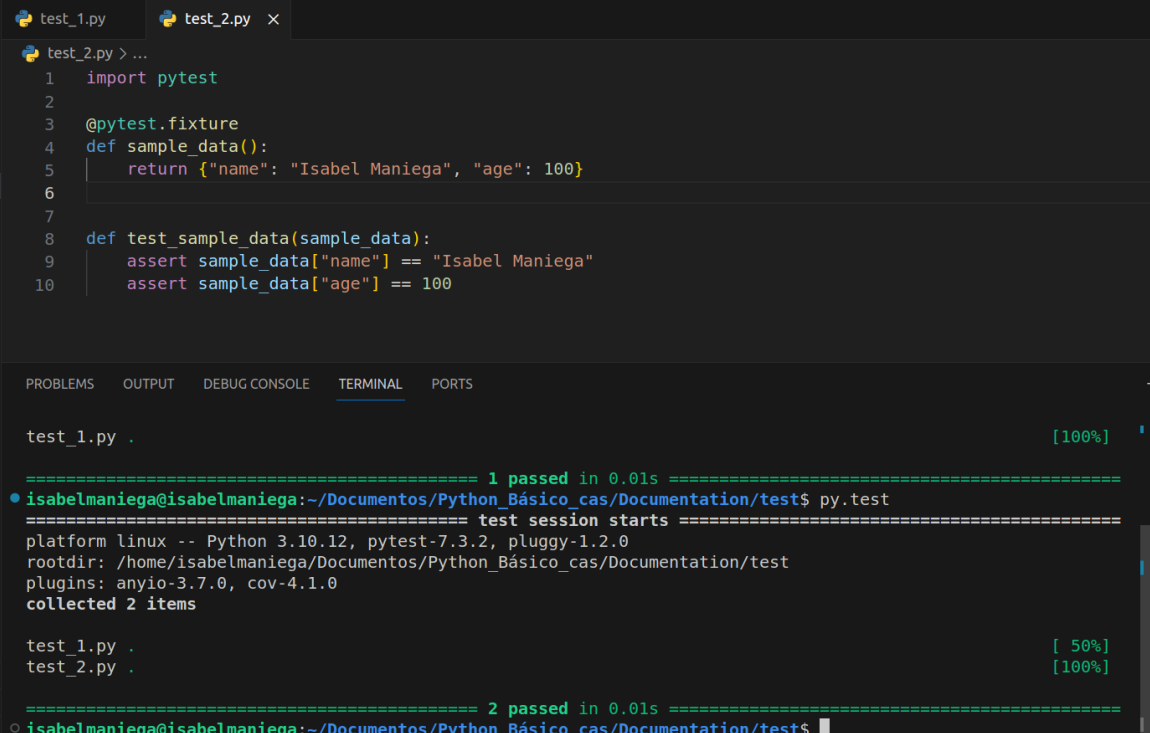
# import pytest

# @pytest.fixture
# def sample_data():
#     return {"name": "Isabel Maniega", "age": 100}

# def test_sample_data(sample_data):
#     assert sample_data["name"] == "Isabel Maniega"
#     assert sample_data["age"] == 100
```

```
In [5]: display.Image('./images/test_3.png')
```

```
Out[5]:
```



```
test_1.py  test_2.py  x
test_2.py > ...
1  import pytest
2
3  @pytest.fixture
4  def sample_data():
5      return {"name": "Isabel Maniega", "age": 100}
6
7
8  def test_sample_data(sample_data):
9      assert sample_data["name"] == "Isabel Maniega"
10     assert sample_data["age"] == 100

PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

test_1.py . [100%]

===== 1 passed in 0.01s =====
• isabelmaniega@isabelmaniega:~/Documentos/Python_Básico_cas/Documentation/test$ py.test
===== test session starts =====
platform linux -- Python 3.10.12, pytest-7.3.2, pluggy-1.2.0
rootdir: /home/isabelmaniega/Documentos/Python_Básico_cas/Documentation/test
plugins: anyio-3.7.0, cov-4.1.0
collected 2 items

test_1.py . [ 50%]
test_2.py . [100%]

===== 2 passed in 0.01s =====
○ isabelmaniega@isabelmaniega:~/Documentos/Python_Básico_cas/Documentation/test$
```

En el ejemplo anterior, el accesorio `sample_data` se pasa automáticamente a cualquier función de prueba que lo solicite como parámetro, lo que garantiza datos de prueba coherentes en todo el conjunto de pruebas.

Observamos que todos los archivos nombrados con `test_` son ejecutados con el comando `py.test` y nos avisan si hay algún error, en ellos.

Vamos a ver un ejemplo donde esperamos que el test no pase:

```
In [ ]: # # test_3.py

# import pytest

# def division(x, y):
#     result = x / y
#     return result

# def test_zero_division():
#     assert division(3, 1) == 3
#     assert division(3, 0) == 0
```

```
In [6]: display.Image('./images/test_4.png')
```

```
Out[6]: © isabelmaniega@isabelmaniega:~/Documentos/Python_Básico_cas/Documentation/test$ py.test
===== test session starts =====
platform linux -- Python 3.10.12, pytest-7.3.2, pluggy-1.2.0
rootdir: /home/isabelmaniega/Documentos/Python_Básico_cas/Documentation/test
plugins: anyio-3.7.0, cov-4.1.0
collected 3 items

test_1.py . [ 33%]
test_2.py . [ 66%]
test_3.py F [100%]

===== FAILURES =====
_____ test_zero_division _____

    def test_zero_division():
        # with pytest.raises(ZeroDivisionError):
        assert division(3, 1) == 3
>       assert division(3, 0) == 0
E       ZeroDivisionError: division by zero

test_3.py:12:
-----
x = 3, y = 0

    def division(x, y):
        # assert y != 0, 'ZeroDivisionError'
        result = x / y
>       ZeroDivisionError: division by zero
E       ZeroDivisionError: division by zero

test_3.py:5: ZeroDivisionError
===== short test summary info =====
FAILED test_3.py::test_zero_division - ZeroDivisionError: division by zero
===== 1 failed, 2 passed in 0.08s =====
```

Se observa con el test_3.py nos pone una F de test erróneo (Fracaso), ya que la división entre 0 nos da un error de Zero Division, por lo tanto el test falla.

Si usamos el módulo de `pytest.raises()` para capturar este error, ya identificado:

```
In [ ]: # # test_3.py

# import pytest

# def division(x, y):
#     result = x / y
#     return result

# def test_zero_division():
#     with pytest.raises(ZeroDivisionError):
#         division(3, 1)
#         division(3, 0)
```

```
In [7]: display.Image('./images/test_5.png')
```

```
Out[7]: • isabelmaniega@isabelmaniega:~/Documentos/Python_Básico_cas/Documentation/test$ py.test
===== test session starts =====
platform linux -- Python 3.10.12, pytest-7.3.2, pluggy-1.2.0
rootdir: /home/isabelmaniega/Documentos/Python_Básico_cas/Documentation/test
plugins: anyio-3.7.0, cov-4.1.0
collected 3 items

test_1.py . [ 33%]
test_2.py . [ 66%]
test_3.py . [100%]

===== 3 passed in 0.01s =====
○ isabelmaniega@isabelmaniega:~/Documentos/Python_Básico_cas/Documentation/test$
```

Observamos que pasa el test, ya que lo hemos capturado.

Otra forma es poner es probar que da un error de ZeroDivision, en el asset capturando la respuesta, esto pasará igualmente el test:

```
In [ ]: # test_3.py

# import pytest

# def division(x, y):
#     result = x / y
#     return result

# def test_zero_division():
#     with pytest.raises(ZeroDivisionError) as excinfo:
#         division(3, 1)
#         division(3, 0)
#     assert "division by zero" in str(excinfo.value)
```

Vamos a usar otra forma de capturar el error como es el decorador: @pytest.mark.xfail

```
In [ ]: # # test_3.py

# import pytest

# def division(x, y):
#     result = x / y
#     return result

# @pytest.mark.xfail(raises=ZeroDivisionError)
# def test_zero_division():
#     division(3, 1)
#     division(3, 0)
```

```
In [8]: display.Image('./images/test_6.png')
```

```
Out[8]: • isabelmaniega@isabelmaniega:~/Documentos/Python_Básico_cas/Documentation/test$ py.test
===== test session starts =====
platform linux -- Python 3.10.12, pytest-7.3.2, pluggy-1.2.0
rootdir: /home/isabelmaniega/Documentos/Python_Básico_cas/Documentation/test
plugins: anyio-3.7.0, cov-4.1.0
collected 3 items

test_1.py . [ 33%]
test_2.py . [ 66%]
test_3.py x [100%]

===== 2 passed, 1 xfailed in 0.02s =====
○ isabelmaniega@isabelmaniega:~/Documentos/Python_Básico_cas/Documentation/test$
```

En este caso falla el test por que detecta un error que hemos tenido en cuenta.

Es probable que usar `pytest.raises` sea mejor para los casos en los que está probando excepciones que su propio código genera deliberadamente, mientras que usar `@pytest.mark.xfail` con una función de verificación probablemente sea mejor para algo como documentar errores no corregidos (donde la prueba describe qué "debería" suceder) o errores en las dependencias.

Otra manera de ejecutar es usar el comando `pytest -v`, nos dará algo más de información:

In [9]: `display.Image('./images/test_7.png')`

Out[9]:

```
• isabelmaniega@isabelmaniega:~/Documentos/Python_Básico_cas/Documentation/test$ pytest -v
===== test session starts =====
platform linux -- Python 3.10.12, pytest-7.3.2, pluggy-1.2.0 -- /usr/bin/python3
cachedir: .pytest_cache
rootdir: /home/isabelmaniega/Documentos/Python_Básico_cas/Documentation/test
plugins: anyio-3.7.0, cov-4.1.0
collected 3 items

test_1.py::test_result PASSED [ 33%]
test_2.py::test_sample_data PASSED [ 66%]
test_3.py::test_zero_division XFAIL [100%]

===== 2 passed, 1 xfailed in 0.03s =====
```

Creado por:

Isabel Maniega