Crear y administrar tareas concurrentes utilizando hilos en Python

Crear un Hilo Simple

```
import threading

def print_numbers():
    for i in range(1, 6):
        print(i)

# Crear un hilo
thread = threading.Thread(target=print_numbers)

# Iniciar el hilo
thread.start()

# Esperar a que el hilo termine
thread.join()

print("Hilo completado")
```

Crear Múltiples Hilos

```
import threading
import time

def print_numbers(thread_name):
    for i in range(1, 6):
        print(f"{thread_name}: {i}")
```

```
        time.sleep(1)

# Crear varios hilos
threads = []
for i in range(3):
       thread  =  threading.Thread(target=print_numbers,
args=(f"Hilo-{i+1}",))
    threads.append(thread)
    thread.start()

# Esperar a que todos los hilos terminen
for thread in threads:
    thread.join()

print("Todos los hilos completados")
```

Uso de un Hilo con una Clase

```
import threading
import time

class PrintNumbersThread(threading.Thread):
    def __init__(self, name):
        threading.Thread.__init__(self)
        self.name = name

    def run(self):
        for i in range(1, 6):
            print(f"{self.name}: {i}")
            time.sleep(1)

# Crear y empezar el hilo
thread = PrintNumbersThread("Hilo-Clase")
thread.start()

# Esperar a que el hilo termine
thread.join()

print("Hilo clase completado")
```

Sincronización de Hilos con Bloqueos

```
import threading
import time

lock = threading.Lock()

def print_numbers(thread_name):
    lock.acquire()
    try:
        for i in range(1, 6):
            print(f"{thread_name}: {i}")
            time.sleep(1)
    finally:
        lock.release()

# Crear varios hilos
threads = []
for i in range(3):
       thread = threading.Thread(target=print_numbers,
args=(f"Hilo-{i+1}",))
    threads.append(thread)
    thread.start()

# Esperar a que todos los hilos terminen
for thread in threads:
    thread.join()

print("Todos los hilos completados con bloqueo")
```

Este código de Python utiliza la biblioteca threading para crear y gestionar múltiples hilos. A continuación se explica en detalle el funcionamiento del código:

1. Importación de Módulos

```
import threading
import time
```

Se importan los módulos threading y time. El módulo threading proporciona herramientas para trabajar con hilos, y time se utiliza para introducir pausas en la ejecución del código.

2. Creación de un Lock

```
lock = threading.Lock()
```

Se crea un objeto Lock de threading. Un Lock es un mecanismo de sincronización que permite que un hilo acceda a un recurso compartido a la vez. Esto es esencial para evitar condiciones de carrera (race conditions).

3. Definición de la Función print_numbers

```
def print_numbers(thread_name):
lock.acquire()
try:
for i in range(1, 6):
print(f"{thread_name}: {i}")
time.sleep(1)
finally:
lock.release()
```

Se define una función llamada print_numbers que toma un argumento thread_name (nombre del hilo). La función realiza las siguientes acciones:

- **Adquirir el Lock:** lock.acquire(). Esto asegura que solo un hilo puede ejecutar el bloque de código protegido por el lock a la vez.
- **Imprimir Números:** Un bucle for que va de 1 a 5 (inclusive). En cada iteración, imprime el nombre del hilo y el número actual, y luego duerme durante 1 segundo (time.sleep(1)).
- **Liberar el Lock:** Esto ocurre en el bloque finally para asegurarse de que el lock se libere incluso si ocurre una excepción durante la ejecución del bloque try.

4. Creación y Inicio de Hilos

```
# Crear varios hilos
threads = []
for i in range(3):
thread = threading.Thread(target=print_numbers, args=(f"Hilo-{i+1}",))
threads.append(thread)
thread.start()
```

En este bloque de código:

- Se inicializa una lista threads para almacenar los objetos Thread.
- Se crea un bucle for que se ejecuta 3 veces. En cada iteración:

5. Esperar a que Todos los Hilos Terminen

```
# Esperar a que todos los hilos terminen
for thread in threads:
thread.join()
```

Aquí, el programa principal espera a que todos los hilos terminen su ejecución:

- Se itera sobre la lista threads.
- Para cada hilo, se llama a thread.join(), lo que bloquea la ejecución del programa principal hasta que el hilo específico haya terminado.

6. Imprimir Mensaje Final

```
print("Todos los hilos completados con bloqueo")
```

Finalmente, se imprime un mensaje indicando que todos los hilos han completado su ejecución y el bloqueo se ha gestionado correctamente.