

# Programación Científica y HPCI

Máster Universitario en Ingeniería Matemática y Computación

**Docente: Jesús Cigales**

## Tema 7

# ¿Cómo estudiar este tema?

## Material de Estudio

[Tema 7](#)

## Material Complementario


[Colas con el módulo queue](#)

[Concurrency in Python – Quick guide](#)

[Ejecución concurrente](#)

Lección magistral

## Test tema 7

 [TestTema7.pdf](#)



# Conceptos básicos

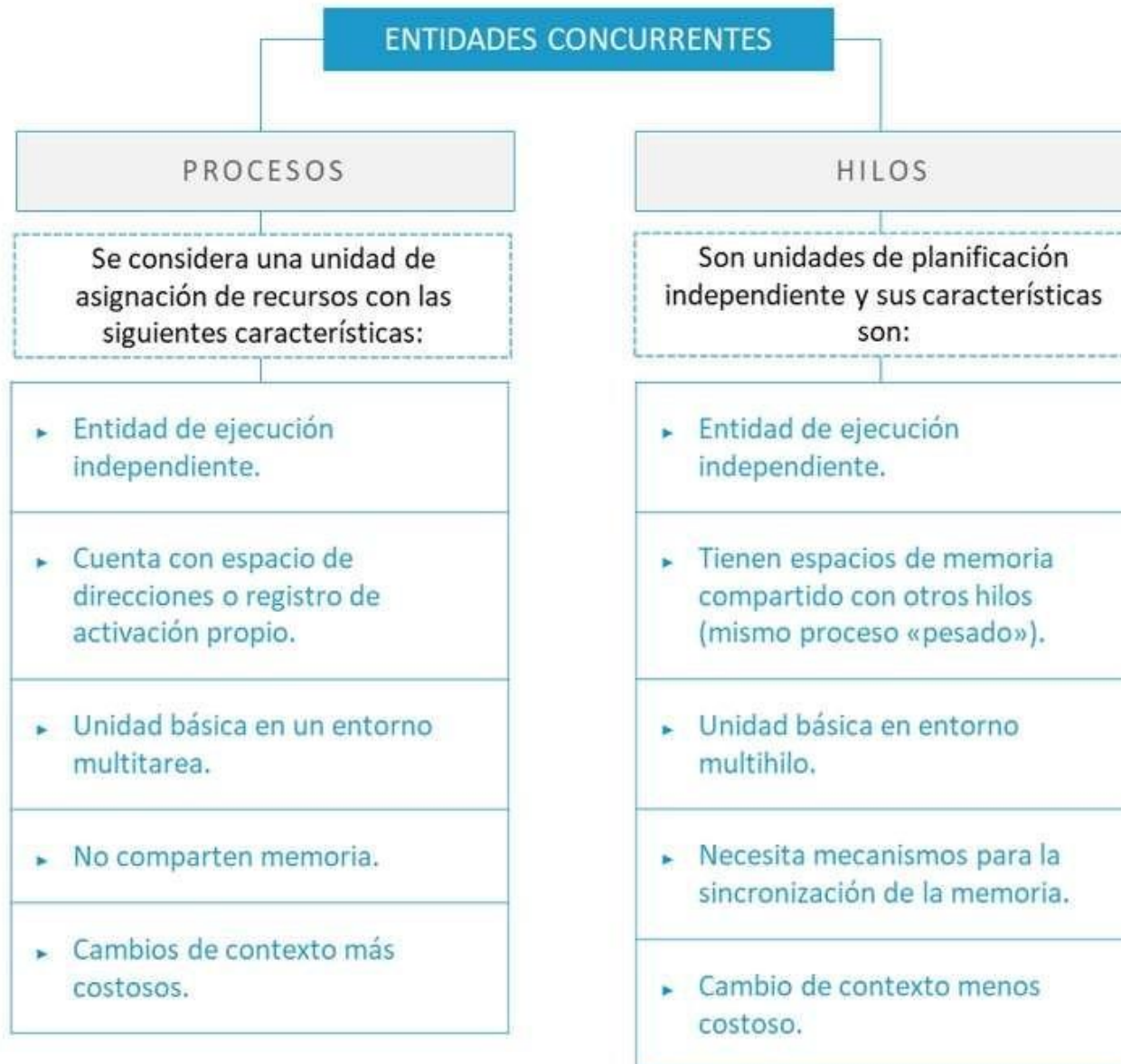
No siempre es real

Concurrencia-ejecución simultánea de conjuntos de instrucciones que guardan cierta independencia.

## Entidades concurrentes

- **Proceso (unidad de asignación de recursos)**
  - Entidad de ejecución independiente
  - Cuenta con espacio de direcciones o registro de activación (cambio de contexto caro)
  - Unidad básica en entorno multitarea
- **Hilo (unidad de planificación independiente)**
  - Entidad de ejecución independiente
  - Tienen espacios de memoria compartido con otros hilos (mismo proceso “pesado”)
  - Unidad básica en entorno multihilo

Programación concurrente-paradigma de programación que permite la creación de programas con ejecución simultánea de múltiples tareas.



# Conceptos básicos

Multiprogramación → gestión de procesos en un sistema monoprocesador.

Multiprocesamiento → gestión de procesos en un sistema multiprocesador (puede existir memoria común).

Procesamiento distribuido → gestión de procesos en un procesadores separados (memoria no compartida).

Programación concurrente → acciones que pueden ser ejecutadas de forma simultanea.

Programación paralela → programación concurrente en un sistema multiprocesador.

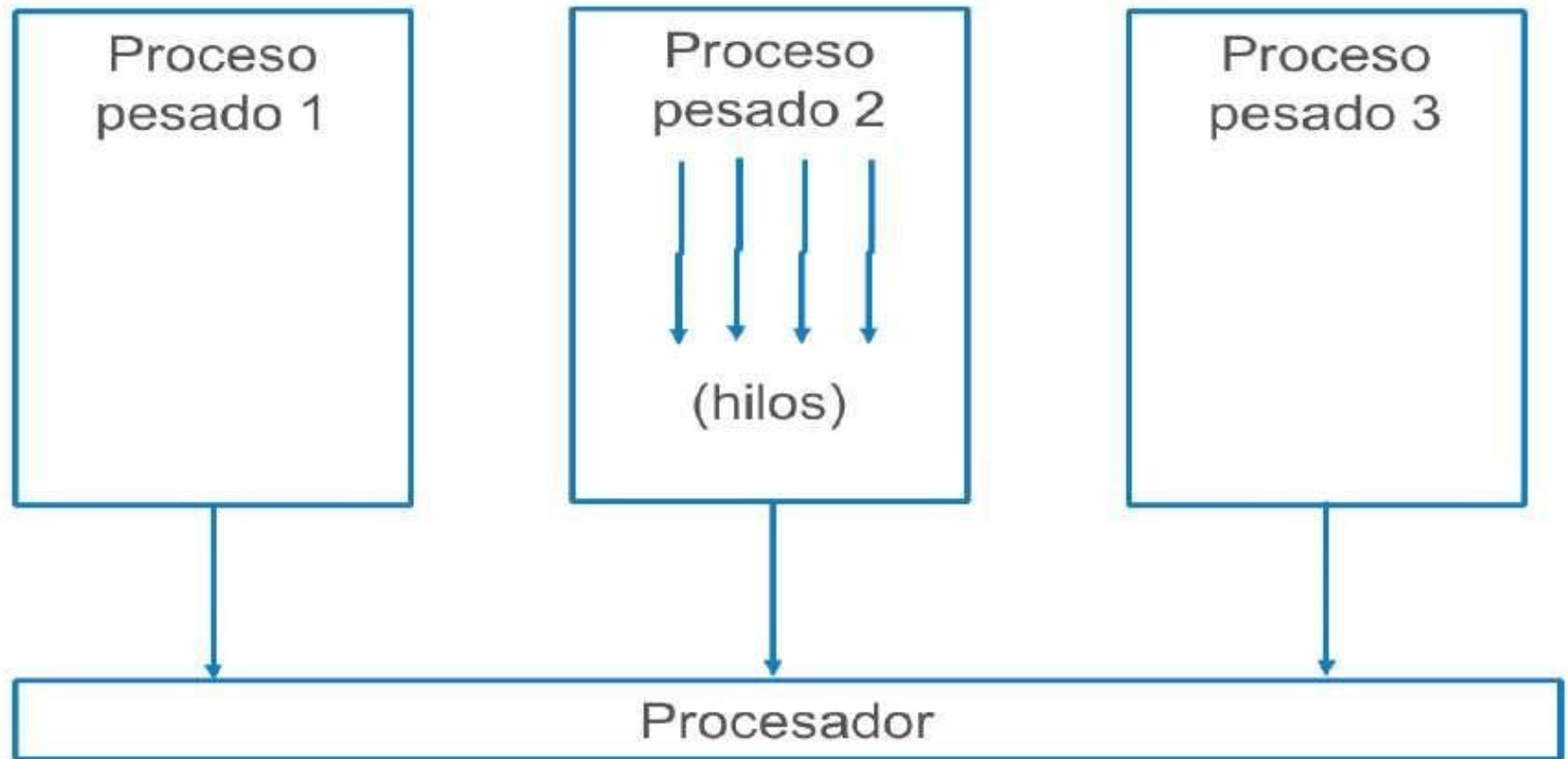
Programación distribuida → programación paralela en un sistema distribuido.



# Definiciones previas

- **Multiprogramación:** gestión de procesos (o hilos) en un sistema monoprocesador. Da lugar a lo que se conoce como programación concurrente que, como se ha visto, consiste en la ejecución «simultánea» de tareas.
- **Multiprocesamiento:** gestión de procesos en un sistema multiprocesador en el que *puede existir memoria común*. Da lugar a lo que se conoce como programación paralela, en la que se ejecutan tareas de forma simultánea, pero sobre distintos procesadores.
- **Procesamiento distribuido:** gestión de procesos en procesadores separados con *memoria no compartida*. Da lugar a lo que se conoce como programación distribuida, en la que se ejecutan tareas de forma simultánea sobre distintos procesadores pero que no comparten memoria.

# Conceptos básicos



# GIL (Global Interpreter Lock )

- El bloqueo o cerrojo global del interprete es un bloqueo asociado a un proceso o hilo.
  - Sólo un hilo puede acceder a un recurso particular e impide el uso de objetos y *bytecodes* a la vez.
- Cada proceso trata al propio intérprete de Python como un recurso.
- Se produce lo que se conoce como el “time slicing” en la planificación de los hilos

¡Problema en  
procesadores  
multinúcleo!

Recolector de  
basura → contador de  
referencias (acceso en  
exclusión mútua)



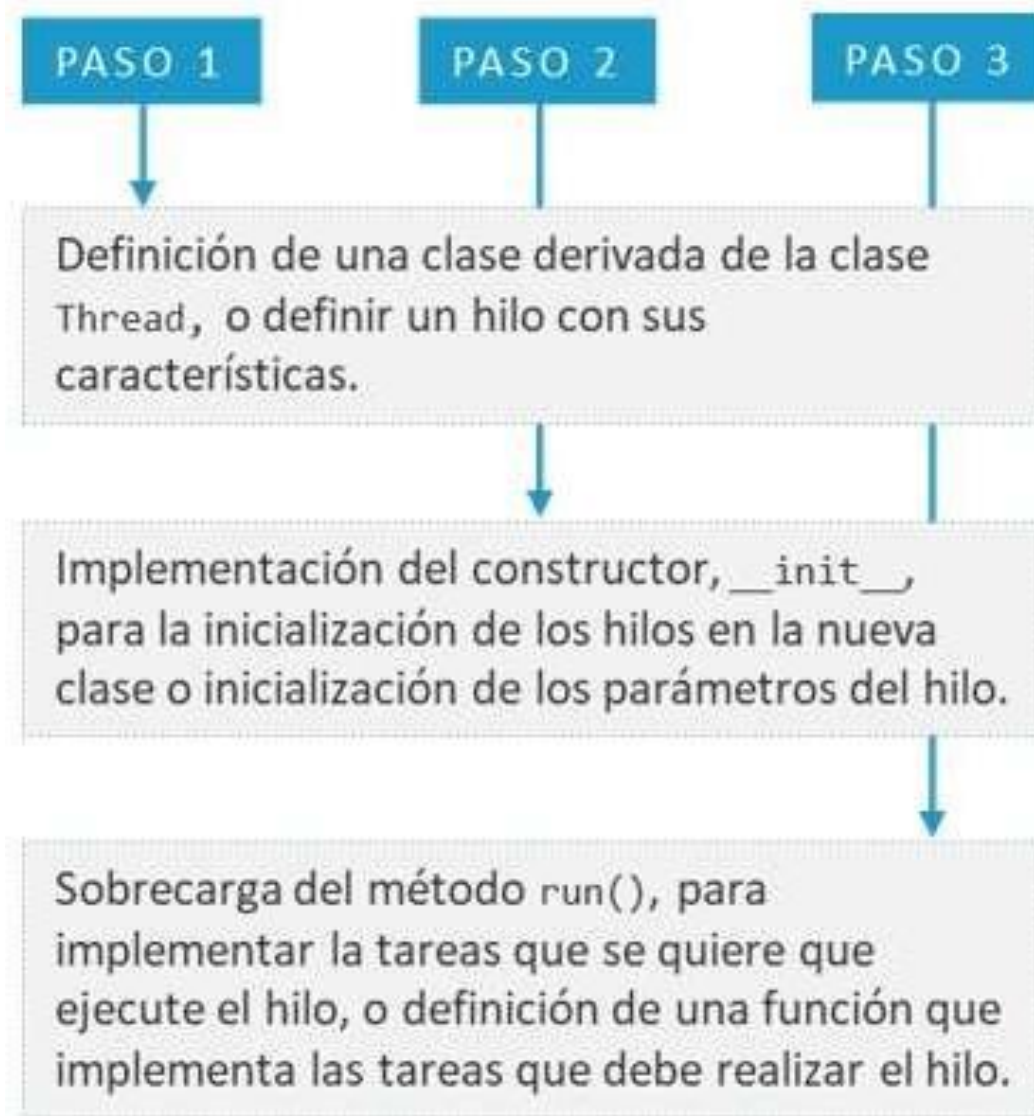
# Aspectos importantes en PC

- Planificación de los hilos o procesos, que determinará que hilo o proceso estará activo en cada momento.
- Asignación de memoria a los hilos o procesos.
- Sincronización de acceso a recursos compartidos o la falta de recursos.
- Prevención de problemas de falta de vitalidad asociados a la programación concurrente como, por ejemplo, los bloqueos entre hilos o procesos.

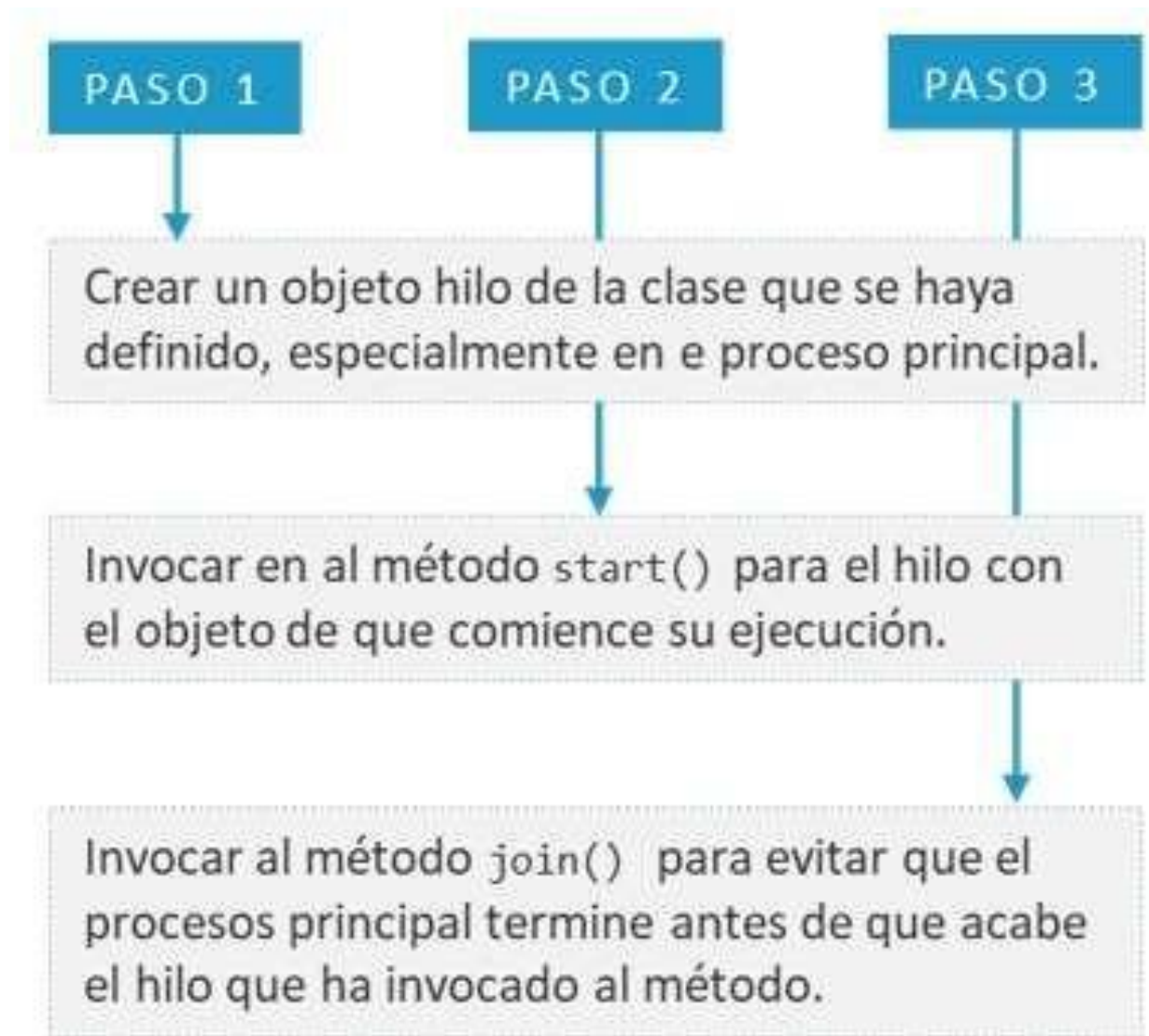
# Módulo threading

Elementos	Descripción
Clase Thread	definición de hilos
Métodos	Inicialización, activación y consulta de estados
Objetos para la sincronización	Cerrosos y condiciones de espera
Excepciones	Problemas durante la ejecución

# Creación de hilos



# Ejecución de hilos



Métodos	Tipo	Descripción
<code>isAlive()</code>	Método de objeto	Devuelve <i>True</i> si el hilo está vivo.
<code>start()</code>	Método de clase	Es un método que se invoca una única vez para que el hilo comience su ejecución.
<code>run()</code>	Método de clase	Este método define las tareas que se van a ejecutar para un hilo en concreto. Este método se sobrecarga para cada clase de hilo que se define.
<code>join()</code>	Método de clase	Bloquea la finalización de otro código hasta que el hilo en el que se llamó al método <code>join()</code> finalice.



# Definición de hilos

## Objetos de Thread

Definición del objetos

Inicialización con constructor de Thread

Asociar una función para su ejecución

Excepciones

Activar hilo start()

Invocar join() para que el hilo principal espere la finalización

```
import threading
import time

def tareaHilo(tiempo):
    print("comienzo de la ejecución del %s \n" %threading.currentThread().name)
    i=5
    while i:
        time.sleep(tiempo)
        print("Ejecutandose %s \n" %threading.currentThread().name)
        i=i-1
    print("%s terminado" %threading.currentThread().name )

if __name__=="__main__":
    hilo1=threading.Thread(name="Hilo Uno", target=tareaHilo, args=(3,))
    hilo2=threading.Thread(name="Hilo Dos", target=tareaHilo, args=(3,))
    hilo1.start()
    hilo2.start()
    hilo1.join()
    hilo2.join()
```

```
def tareaHilo(tiempo):
    print("comienzo de la ejecución del %s \n" %threading.currentThread().name)
    i=5
    while i:
        time.sleep(tiempo)
        print("Ejecutandose %s \n" %threading.currentThread().name)
        i=i-1
    print("%s terminado" %threading.currentThread().name )

if __name__=="__main__":

    hilo1=threading.Thread(name="Hilo Uno", target=tareaHilo, args=(3,))
    hilo2=threading.Thread(name="Hilo Dos", target=tareaHilo, args=(5,))
    hilo1.start()
    hilo2.start()
    hilo1.join()
    hilo2.join()
```

Help Variable Explorer Plots Files

× Console 1/A

Hilo Uno terminado

In [2]: runfile('/Users/MLUISA/Library/Mobile Documents/com~apple~CloudDocs/MAP/EJERCICIOS/thread.py', wdir='/Users/MLUISA/Library/Mobile Documents/com~apple~CloudDocs/MAP/EJERCICIOS')  
comienzo de la ejecución del Hilo Uno  
comienzo de la ejecución del Hilo Dos

Ejecutandose Hilo Uno

Ejecutandose Hilo Dos

Ejecutandose Hilo Uno

Ejecutandose Hilo Uno

Ejecutandose Hilo Dos

Ejecutandose Hilo Uno

Ejecutandose Hilo Dos

Ejecutandose Hilo Uno

Hilo Uno terminado

Ejecutandose Hilo Dos

Ejecutandose Hilo Dos

Hilo Dos terminado

# Definición de hilos

## Objetos clases derivadas de Thread

Definición de la clase y de objetos

Inicialización con constructor de clases derivadas

Sobrecarga del método run()

Problemas durante la ejecución

Activar hilo *str()*

Invocar *join()* para que el hilo principal espere la finalización

```
import threading
import time

class HiloEjemplo (threading.Thread):
    def __init__(self, id, nombre, tiempo):
        threading.Thread.__init__(self)
        self.id=id
        self.name=nombre
        self.tiempo=tiempo

    def run(self):
        print("comienzo de la ejecución del %s \n" %self.name)
        i=5
        while i:
            time.sleep(self.tiempo)
            print("Ejecutandose %s \n" %self.name)
            i=i-1
        print("%s terminado" %self.name )

def tareaHilo(tiempo):
    print("comienzo de la ejecución del %s \n" %threading.currentThread().name)
    i=5
    while i:
        time.sleep(tiempo)
        print("Ejecutandose %s \n" %threading.currentThread().name)
        i=i-1
    print("%s terminado" %threading.currentThread().name )

if __name__=="__main__":
    hilo1=HiloEjemplo(1, "Hilo Uno", 5)
    hilo2=HiloEjemplo(1, "Hilo Dos", 5)
    hilo1.start()
    hilo2.start()
    hilo1.join()
    hilo2.join()
```

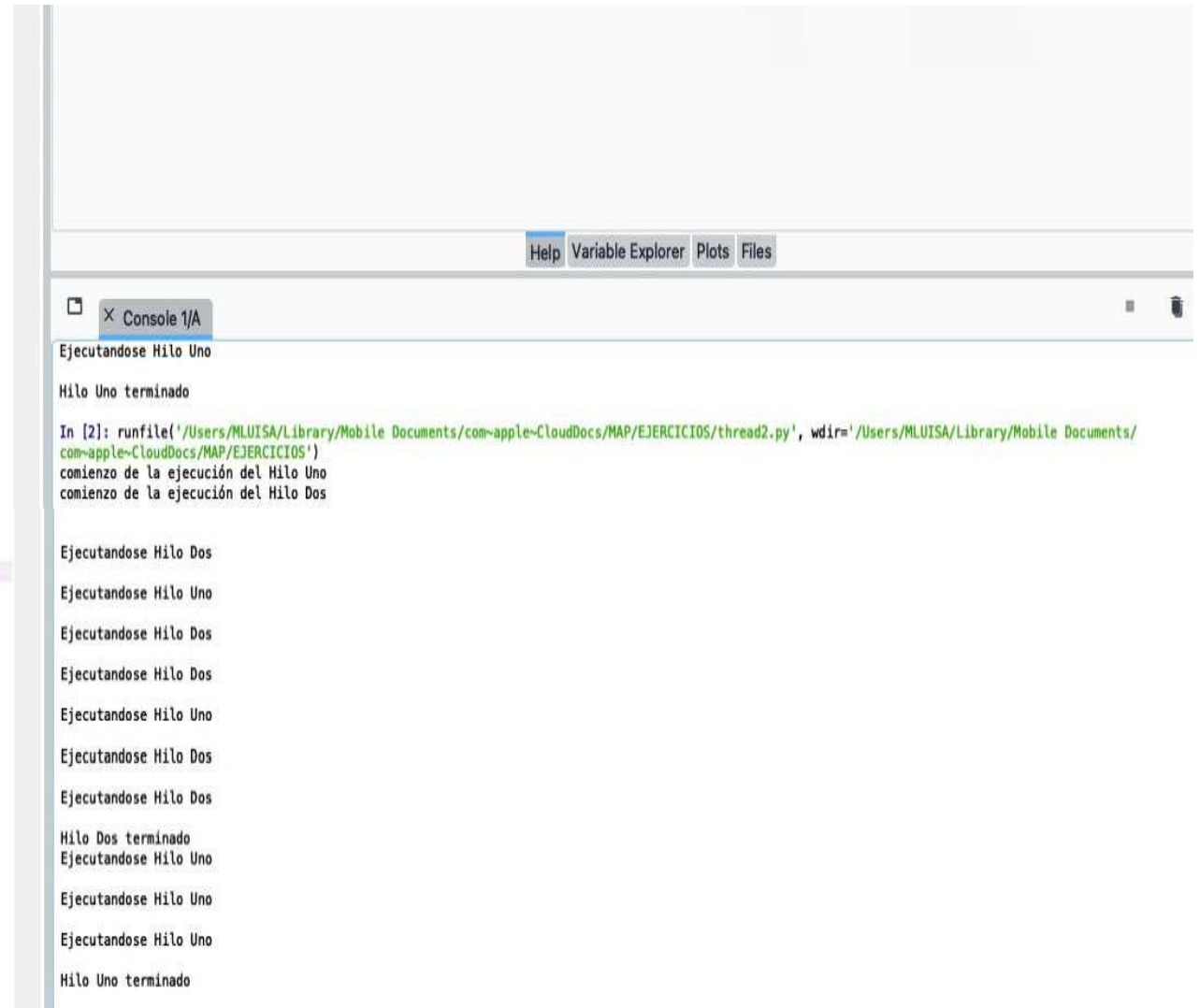
```

def run(self):
    print("comienzo de la ejecución del %s \n" %self.name)
    i=5
    while i:
        time.sleep(self.tiempo)
        print("Ejecutandose %s \n" %self.name)
        i=i-1
    print("%s terminado" %self.name )

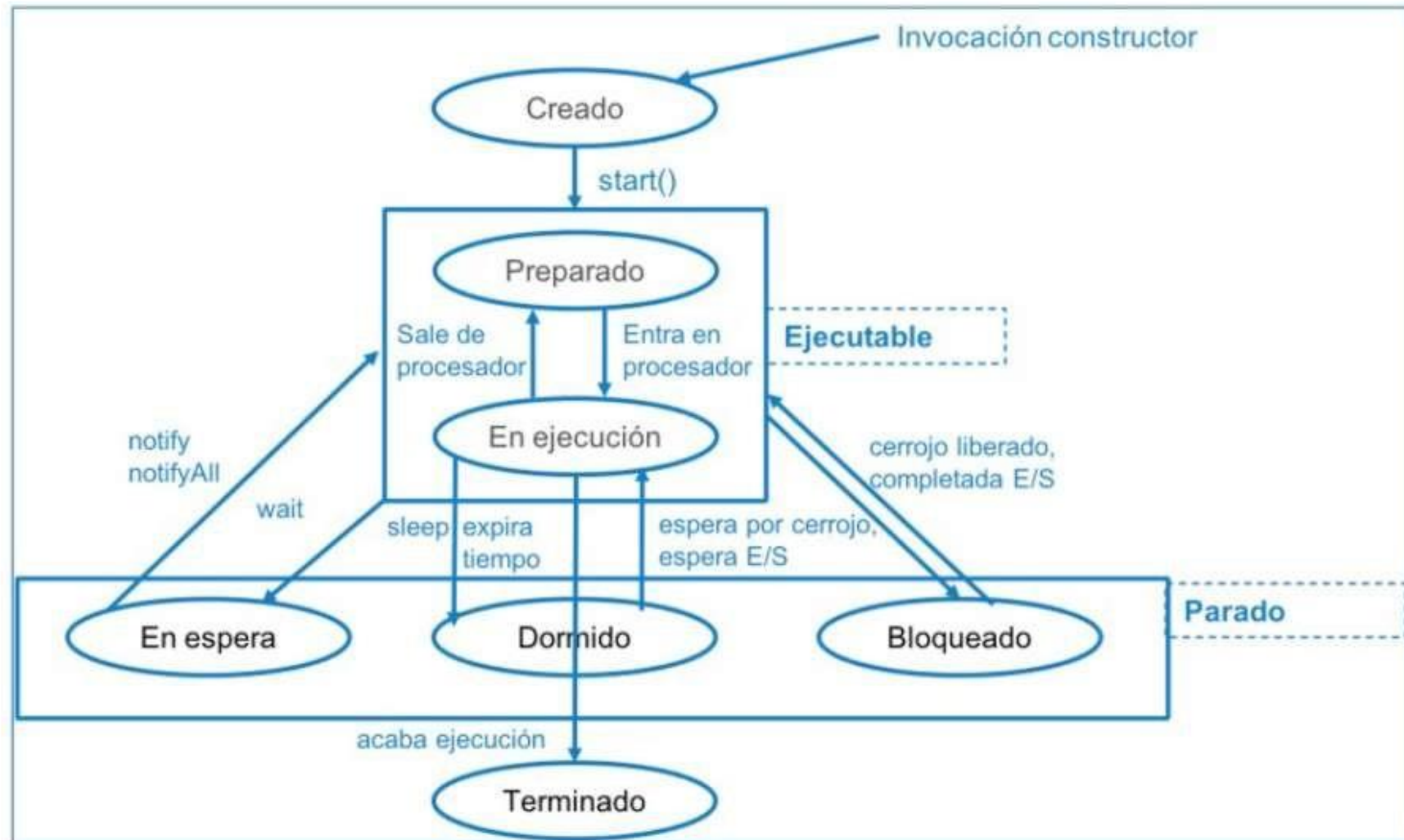
def tareaHilo(tiempo):
    print("comienzo de la ejecución del %s \n" %threading.currentThread().name)
    i=5
    while i:
        time.sleep(tiempo)
        print("Ejecutandose %s \n" %threading.currentThread().name)
        i=i-1
    print("%s terminado" %threading.currentThread().name )

if __name__=="__main__":
    hilo1=HiloEjemplo(1, "Hilo Uno", 10)
    hilo2=HiloEjemplo(2, "Hilo Dos", 5)
    hilo1.start()
    hilo2.start()
    hilo1.join()
    hilo2.join()

```



# Estados de un hilo en Python





# Métodos de sincronización

## ► Conflictos de lectura/escritura

- un hilo consulta el estado de un objeto a la vez que otro intenta modificarlo.
- No en todas las ocasiones (condiciones de carrera).

## ► Conflictos de escritura/escritura

- Dos hilos tratan de modificar el estado de un objeto simultáneamente.
- Las acciones no son atómicas

- Cerrojos o bloqueos sobre recursos.
- Bloqueos o cerrojos recursivos o reentrantes
- Semáforos
- Condiciones
- Eventos
- Barreras

# Lock

```
import threading
import time

cerrojo = threading.Lock()
def incrementar():
    global contador

    for i in range(1000000):
        cerrojo.acquire()
        contador+=1
        cerrojo.release()

def decrementar():
    global contador
    for i in range(1000000):
        cerrojo.acquire()
        contador-=1
        cerrojo.release()

if __name__ == "__main__":
    contador=0

    hilo1 = threading.Thread(target=incrementar)
    hilo2 = threading.Thread(target=decrementar)
```

acquire()

release()

# Semáforos

```
class GestorConexiones(object):
    def __init__(self):
        threading.Thread.__init__(self)
        self.activas = []
        self.cerrojo = threading.Lock()
    def activar(self, name):
        with self.cerrojo:
            self.activas.append(name)
            logging.debug('Conexiones activas: %s', self.activas)
    def desactivar(self, name):
        with self.cerrojo:
            self.activas.remove(name)
            logging.debug('Conexiones activas: %s', self.activas)

def conexion(s, gestor):
    logging.debug('Esperando para unirse al gestor')
    with s:
        nombre = threading.currentThread().getName()
        gestor.activar(nombre)
        time.sleep(0.1)
        gestor.desactivar(nombre)

gestor = GestorConexiones()
semaforo = threading.Semaphore(2)
for i in range(4):
    t = threading.Thread(target=conexion, name="Hilo "+str(i), args=(semaforo, gestor))
```

uso por contexto

definición e inicialización

asignación



[www.unir.net](http://www.unir.net)