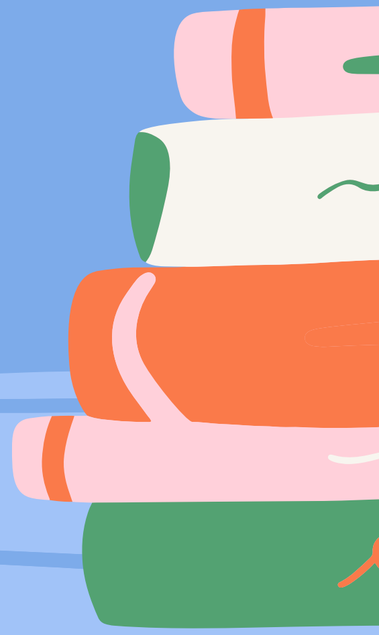
Threading





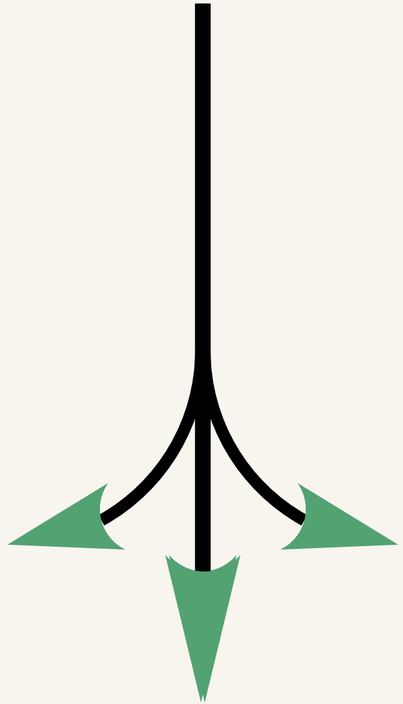
¿Qué es un thread?



Thread

Un ***thread*** (o hilo) es una unidad de ejecución de código dentro de un proceso.

Todo proceso al crearse posee al menos un ***thread*** llamado ***thread principal***



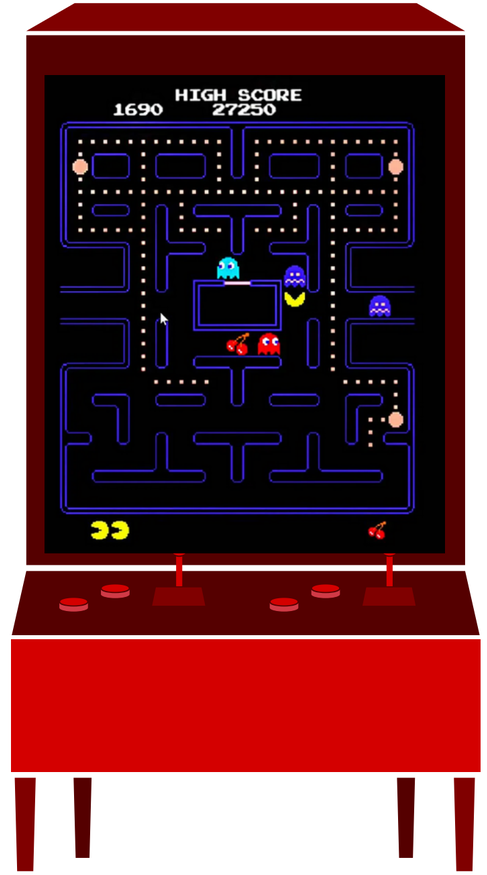
Aplicaciones

Separar funciones independientes

Interfáces gráficas

Aplicaciones multiusuario





Uber



THREADS EN PYTHON

- Necesitamos una librería

- Un thread se asocia a una función



Recomendamos NO usar Jupyter para probar su código

```python
import threading

my_thread = threading.Thread(target=fun_funtion)
```

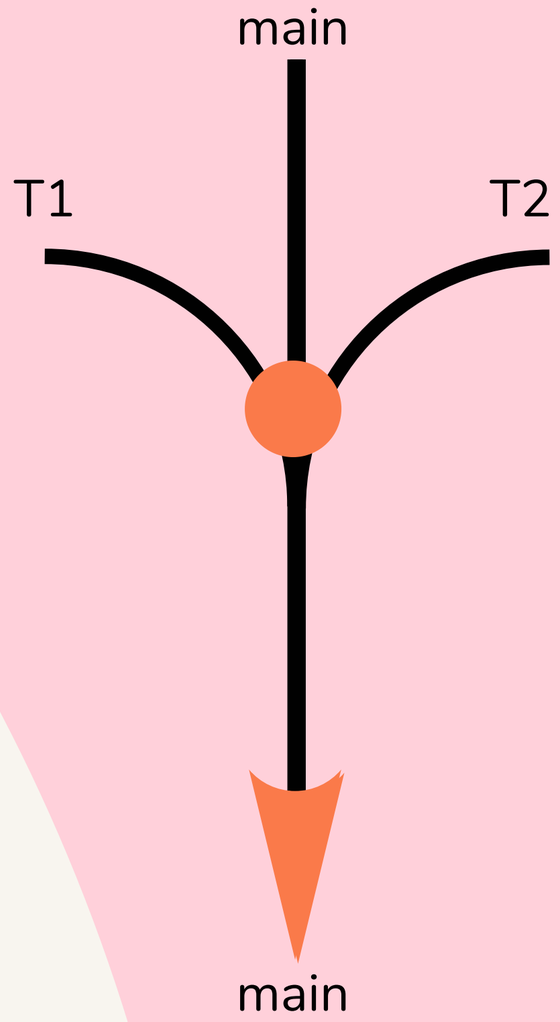


Ejecutemos código...

JOIN

- A veces se necesita esperar que los threads terminen su ejecución.
- La herramienta es el método join:

thread.join(timeout=None)



Nota: timeout recibe None o una cantidad de segundos

JOIN

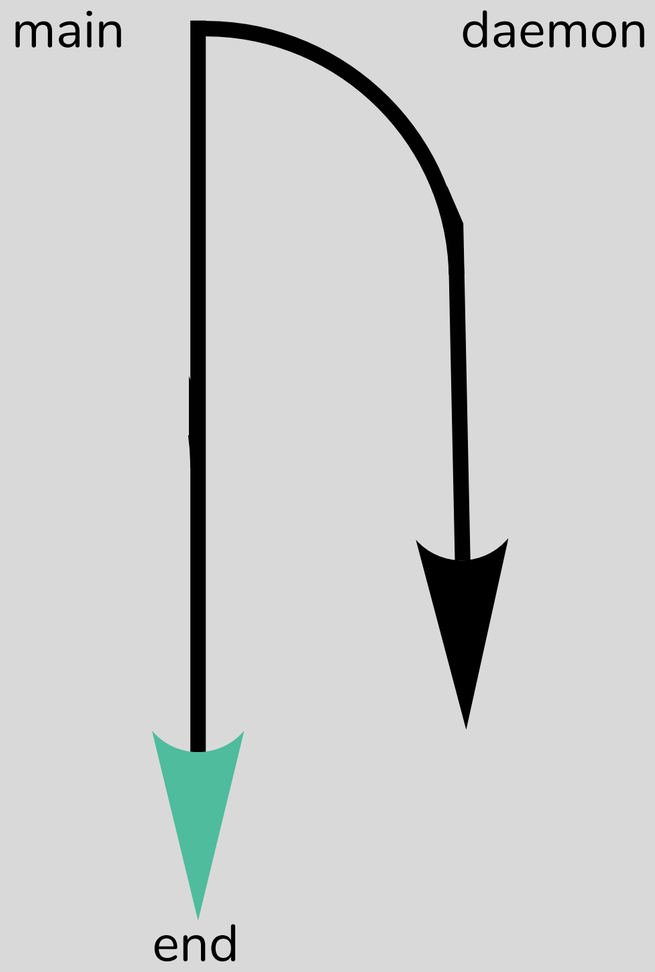
Ejemplo



DAEMON THREADS

- Threads que el programa principal no espera para terminar
- Daemon es un atributo del thread

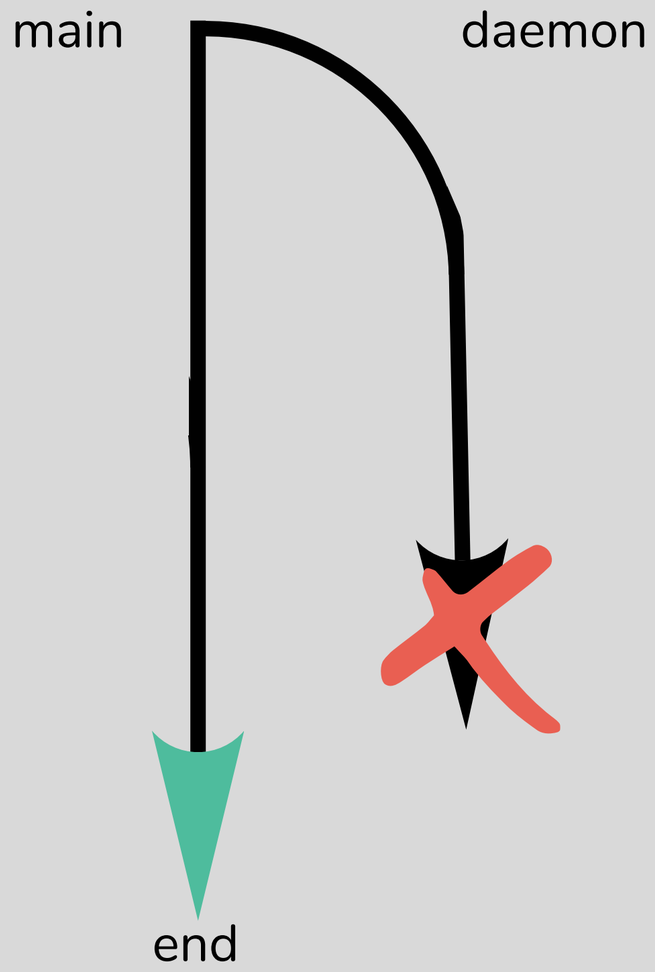
thread.daemon = True



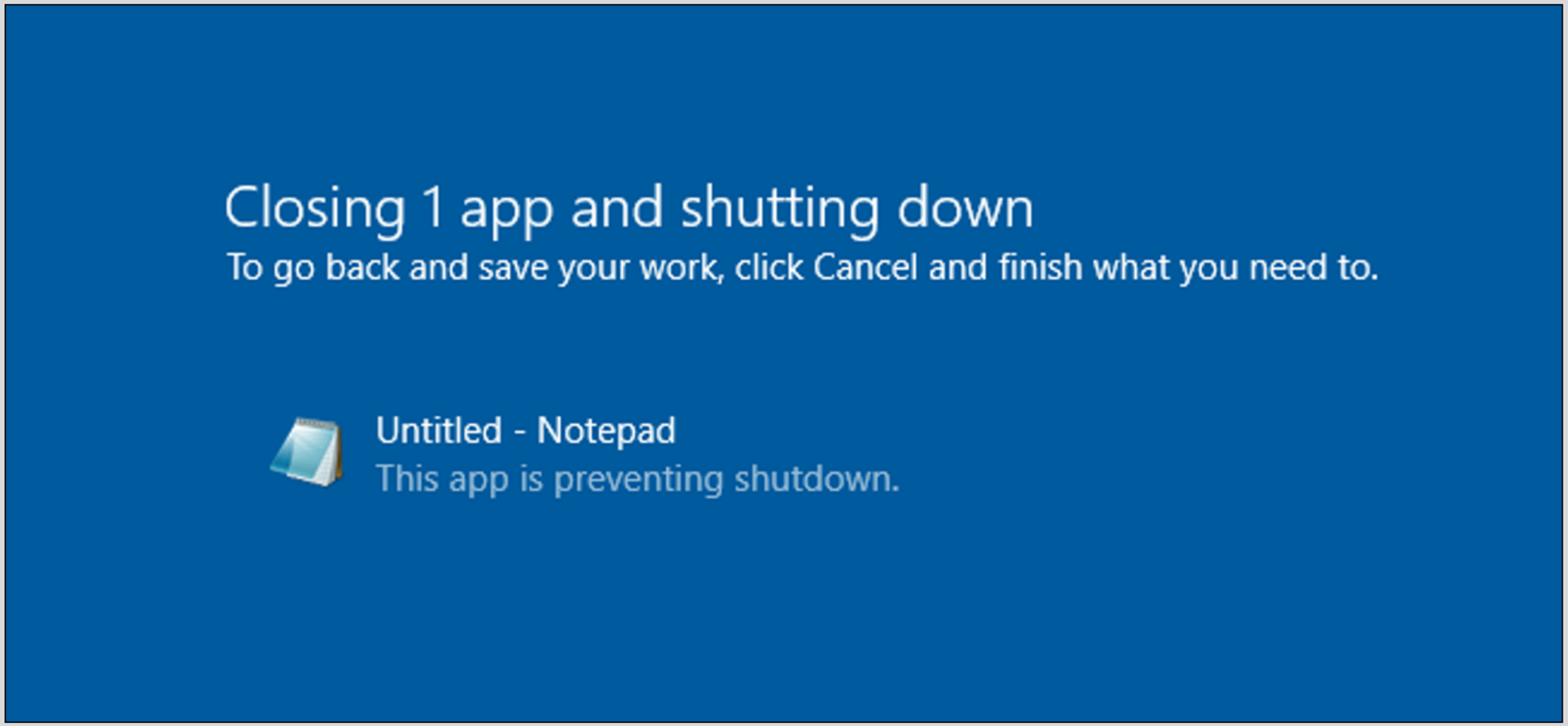
DAEMON THREADS

- Threads que el programa principal no espera para terminar
- Daemon es un atributo del thread

thread.daemon = True



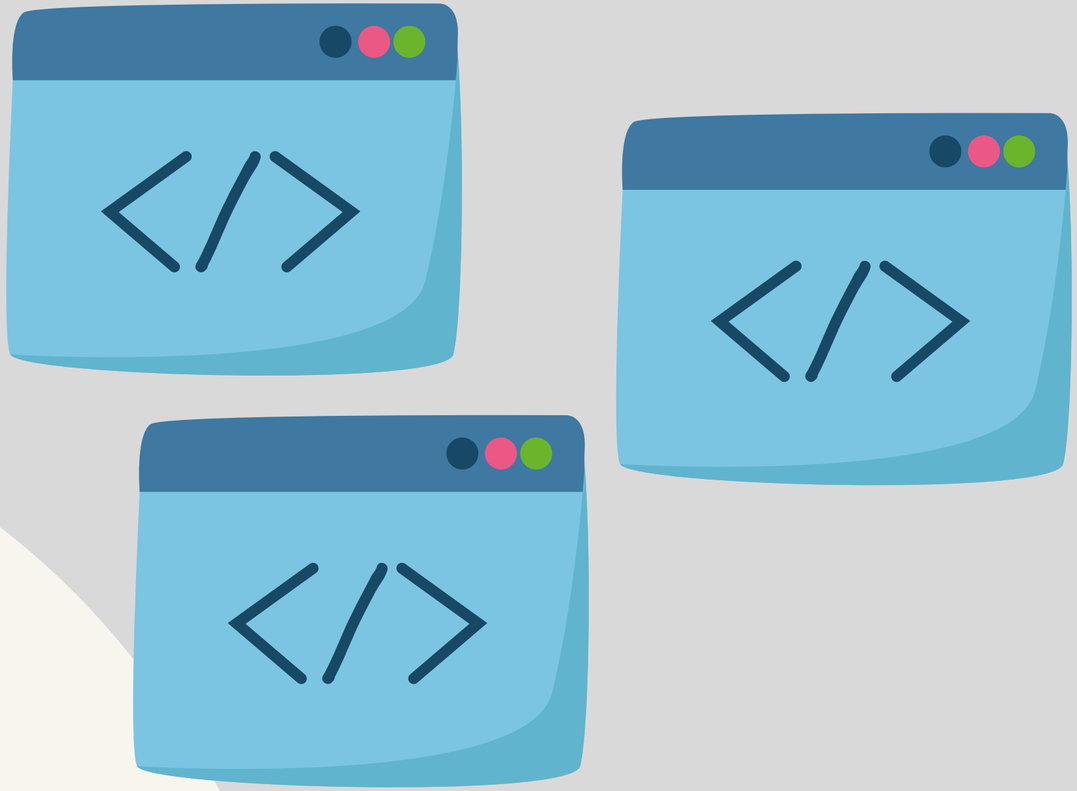
DEAMON THREADS



DEAMON THREADS

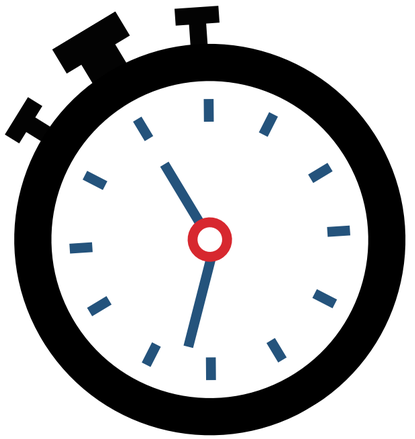
Ejemplo





TIMERS

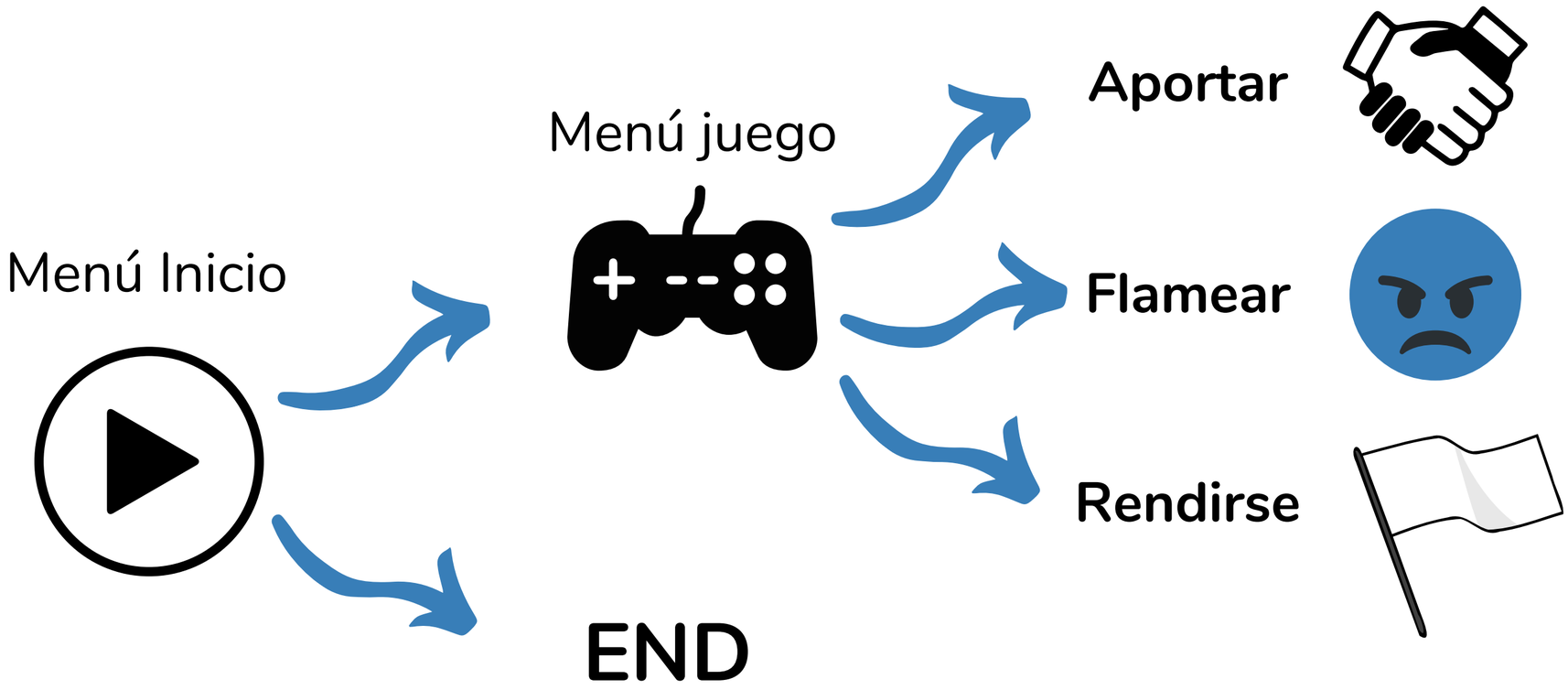
- Threads que se ejecutan después de un intervalo de tiempo.



threading.Timer(intervalo, funcion, *args, **kwargs)

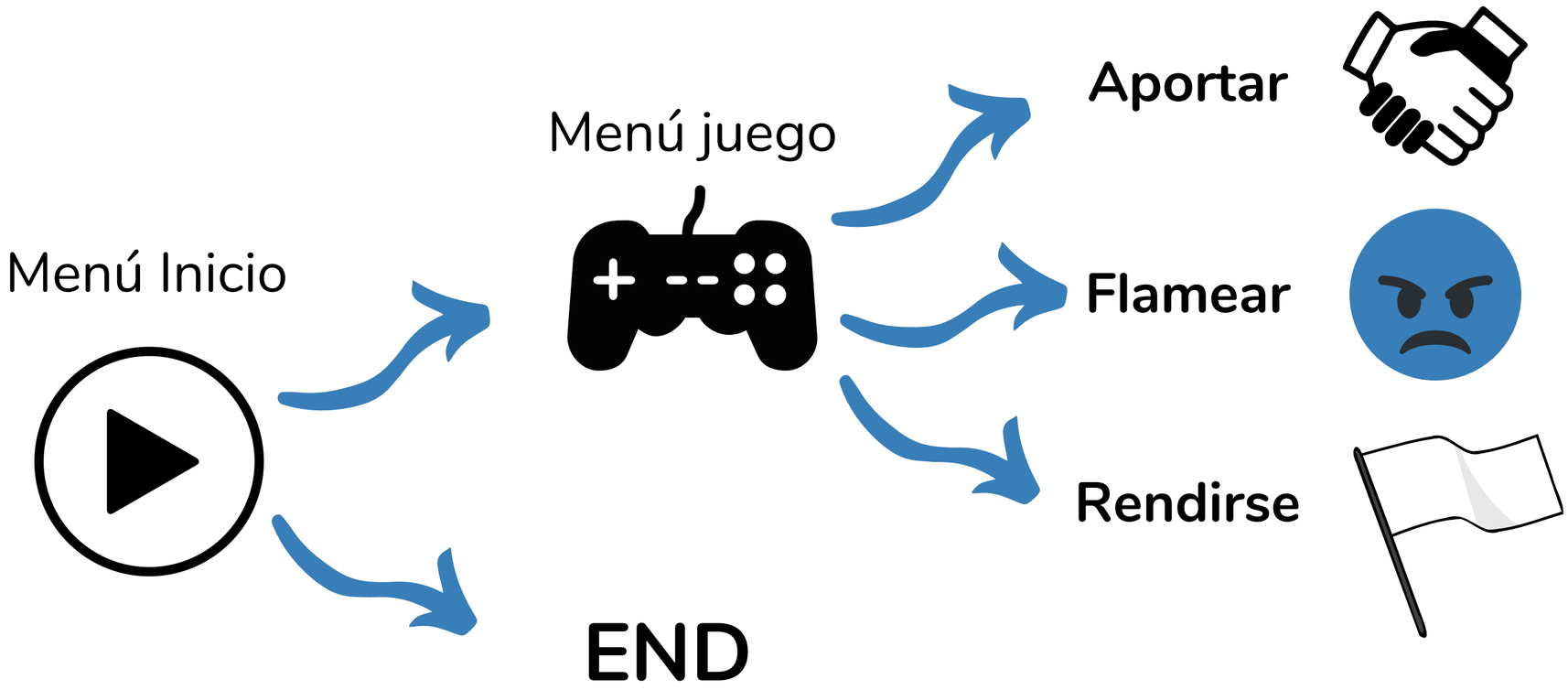
TIMERS

Ejemplo



TIMERS

Ejemplo

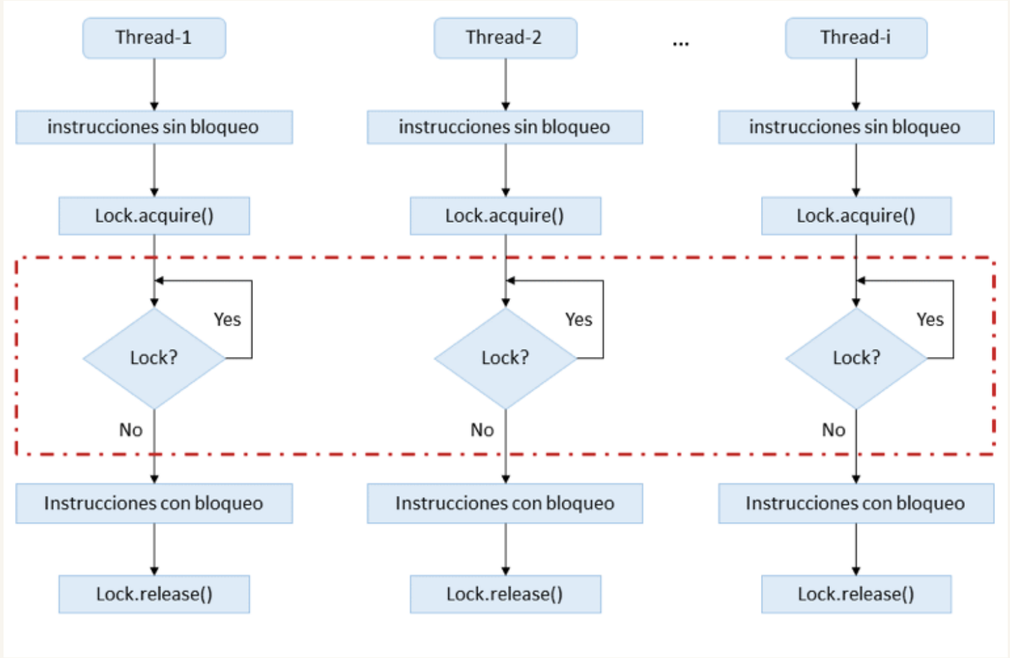


Mala actitud tiene un **banneo** por un **tiempo**

SINCRONIZACIÓN: LOCKS

¿PORQUÉ USAR LOCKS?

Existen casos en que necesitamos que dos threads ajusten un mismo valor. La manera en que esto funciona no es predecible, es decir, el programa puede variar entre Threads en momentos críticos de calculo (recordar que python solo usa un núcleo, por lo que los Threads son "simulados").



SINCRONIZACIÓN: LOCKS

EJEMPLO:

Sabemos que Facebook es dueño de Instagram tambien. En ese contexto, se quiere implementar un nuevo sistema que cuente los likes de un determinado post desde Facebook y instagram. En este caso los Thread son cada red social, y se encargan de usar la función count_likes_from_2_sources para sumar los likes.

SINCRONIZACIÓN: EVENTOS

¿PORQUÉ USAR EVENTOS?

Es normal que al hacer procesos paralelos nos surja el interés por sincronizar los distintos Thread, por ejemplo para esperar que ambos terminen al mismo tiempo.

SINCRONIZACIÓN: EVENTOS



EJEMPLO:

Cuando se usa git para un trabajo con más de un developer, es común que existan algunos problemas de sincronización entre las personas, como por ejemplo que dos personas distintas editen el mismo archivo. El error común es el siguiente:

```
 ! [rejected]        main -> main (fetch first)
error: failed to push some refs to 'https://github.com/iic2233/private.git'
hint: Updates were rejected because the remote contains work that you do
hint: not have locally. This is usually caused by another repository pushing
hint: to the same ref. You may want to first integrate the remote changes
hint: (e.g., 'git pull ...') before pushing again.
hint: See the 'Note about fast-forwards' in 'git push --help' for details.
```

SINCRONIZACIÓN: EVENTOS



EJEMPLO:

Para simular esto vamos a usar eventos.
Haremos un evento que diga cuando termino de trabajar un usuario, para que en este momento el siguiente pueda empezar a trabajar.

```
 ! [rejected]        main -> main (fetch first)
error: failed to push some refs to 'https://github.com/iic2233/private.git'
hint: Updates were rejected because the remote contains work that you do
hint: not have locally. This is usually caused by another repository pushing
hint: to the same ref. You may want to first integrate the remote changes
hint: (e.g., 'git pull ...') before pushing again.
hint: See the 'Note about fast-forwards' in 'git push --help' for details.
```

EJERCICIO PROPUESTO