AI활용프로그래밍

Week 13. 쓰레드 & 미니게임

threading 기초 + 동시성으로 게임 개선하기

학습 목표

동시성(concurrency)과 쓰레드(thread)의 기본 개념을 이해한다.
threading.Thread로 쓰레드를 생성/실행(start)하고 join으로 동기화할 수 있다.
간단한 미니게임에 타이머/애니메이션 등 '동시 실행' 요소를 적용한다.

오늘의 구성

- 동시성 vs 병렬성 개념
- threading.Thread 기본(start/join)
- 공유 데이터와 경쟁 상태(race condition)
- Lock/Queue로 안전하게 통신하기
- 미니게임 설계 + With AI 실습 + 퀴즈

Step 1. 동기 실행의 문제: sleep이 전체를 멈춘다

해야 할 것

- 코드를 그대로 실행
- sleep 동안 입력이 "안 나오는" 느낌 확인
- 왜 게임/GUI에서 문제가 되는지 말로 설명

체크

3초 동안 아무 것도 못 하고 기다린 뒤에야 입력을 받는다.

실행 코드

```
1 import time
2
3 print('시작')
4 time.sleep(3)
5 x = input('입력: ')
6 print('끝', x)
```

동시성(Concurrency) vs 병렬성(Parallelism)

```
1 # 동시성: 번갈아가며 처리(논리적 동시에)
2 # 병렬성: 실제로 동시에 처리(멀티 코어)
3 # 파이썬 thread는 I/O 작업에 특히 유용
```

핵심 포인트

I/O(입출력) 대기 중에도 다른 일을 할 수 있다.
CPU 계산을 '진짜 병렬'로 만들려면 다른 접근이 필요할 수 있다.
오늘은 threading의 기초 사용법에 집중한다.

Step 2. threading.Thread 기본(start/join)

해야 할 것

- work() 함수를 만든다
- Thread(target=work) 생성
- start() 후 join()으로 종료 기다림

체크

작업이 출력된 뒤 "메인 종료"가 출력된다(순서 확인).

실행 코드

```
1 import threading
2
3 def work():
4     print('작업')
5
6 t = threading.Thread(target=work)
7 t.start()
8 t.join()
9 print('메인 종료')
```

Step 3. daemon 쓰레드(메인이 끝나면 같이 종료)

해야 할 것

- daemon=True 옵션을 준다
- 메인 프로그램이 끝나면 쓰레드도 종료됨
- 중요한 저장 작업은 daemon으로 두지 않기

체크

tick이 몇 번 출력된 뒤, "메인 끝"이 출력되며 프로그램이 종료된다.

실행 코드

```
1 import threading, time
2
3 def bg():
4     while True:
5         print('tick')
6         time.sleep(0.5)
7
8 t = threading.Thread(target=bg, daemon=True)
9 t.start()
10 time.sleep(2)
11 print('메인 끝')
```

Step 4. race condition 재현(공유 변수)

해야 할 것

- 코드를 여러 번 실행해 본다
- count가 기대보다 작게 나올 수 있음 확인
- 왜 그런지(원자성/순서) 설명

체크

실행마다 결과가 달라지거나(특히 작게) 200000이 아닐 수 있다.

실행 코드

```
1 import threading
2
3 count = 0
4
5 def inc():
6     global count
7     for _ in range(100000):
8         count += 1
9
10 threads = [threading.Thread(target=inc) for _ in range(2)]
11 [t.start() for t in threads]
12 [t.join() for t in threads]
13 print('count =', count)  # 기대: 200000
```

Step 5. Lock으로 임계구역 보호

해야 할 것

- Lock을 1개 만든다
- with lock: 안에서만 count 수정
- 다시 실행해서 200000이 안정적으로 나오는지 확인

체크

항상 count가 200000으로 나온다(재현 가능).

실행 코드

```
1 import threading
2
3 count = 0
4 lock = threading.Lock()
5
6 def inc():
7     global count
8     for _ in range(100000):
9         with lock:
10            count += 1
11
12 threads = [threading.Thread(target=inc) for _ in range(2)]
13 [t.start() for t in threads]
14 [t.join() for t in threads]
15 print('count =', count)
```

Step 6. Queue로 안전한 통신(생산자-소비자)

해야 할 것

- 작업 쓰레드가 q.put()로 결과 전달
- 메인이 q.get()으로 받기
- 공유 변수 직접 수정 대신 "메시지"로 전달

체크

메인에서 받은 메시지: done 이 출력된다.

실행 코드

```
1 import threading
2 from queue import Queue
3
4 q = Queue()
5
6 def worker():
7     # 어떤 계산 결과를 만든다고 가정
8     q.put('done')
9
10 t = threading.Thread(target=worker)
11 t.start()
12 msg = q.get()
13 t.join()
14 print('메인에서 받은 메시지:', msg)
```

Timer/주기 작업: threading.Timer

```
import threading
def say():
    print('tick')
t = threading.Timer(1.0, say)
t.start()
```

핵심 포인트

일정 시간 뒤 함수 실행(1회)
반복 타이머는 루프+sleep 또는 스케줄링 필요
미니게임에서 제한 시간 구현에 응용 가능

Step 7. Event로 종료 신호 보내기

해야 할 것

- evt = threading.Event() 만들기
- 백그라운드 쓰레드에서 evt.is_set()로 확인
- 메인에서 evt.set()으로 종료

체크

tick이 몇 번 출력된 뒤 "종료"가 출력되고 깔끔히 끝난다.

실행 코드

```
1 import threading, time
2
3 evt = threading.Event()
4
5 def ticker():
6     while not evt.is_set():
7         print('tick')
8         time.sleep(0.5)
9
10 t = threading.Thread(target=ticker)
11 t.start()
12 time.sleep(2)
13 evt.set()
14 t.join()
15 print('종료')
```

예제: 로딩 스피너(백그라운드)

```
import threading, time
running = True
def spinner():
    while running:
        print('.', end='', flush=True)
        time.sleep(0.2)
t = threading.Thread(target=spinner,
daemon=True)
t.start()
# ... 메인 작업 ...
running = False
```

핵심 포인트

메인 작업이 진행되는 동안 UI 효과를 줄 수 있다.
공유 변수(running) 변경 시 타이밍 문제 주의
실전에서는 Lock/Event로 더 안전하게 제어

Step 8. Mini-game 스켈레톤: 5초 제한 입력

해야 할 것

- 타이머 쓰레드를 daemon으로 시작
- timeout 플래그(또는 Event)로 시간 초과 판단
- 입력 후 결과 출력

체크

5초 안에 입력하면 "입력 성공", 늦으면 "시간 초과!"가 출력된다.

실행 코드

```
1 import threading, time
2
3 timeout = False
4
5 def timer():
6     global timeout
7     time.sleep(5)
8     timeout = True
9
10 threading.Thread(target=timer, daemon=True).start()
11 ans = input('5초 안에 입력: ')
12
13 if timeout:
14     print('시간 초과!')
15 else:
16     print('입력 성공:', ans)
```

게임 설계: 상태(State)로 생각하기

```
# 상태 예시
READY -> WAITING -> GO -> DONE
# 이벤트: timer 완료, 사용자 입력
```

핵심 포인트

복잡한 프로그램은 상태로 나누면 깔끔해진다.
쓰레드는 상태 전환 이벤트를 발생시키는 역할
AI에게: 상태 다이어그램을 글로 설명하게 해보자

안전한 설계 팁

```
# 1) 공유 변수 최소화
# 2) 공유 시 Lock/Event/Queue 사용
# 3) 출력/입력은 한 곳에서 관리
```

핵심 포인트

race condition은 재현이 어려워 디버깅이 힘들다.
Queue 기반 메시지 전달이 실무에서 많이 쓰인다.
오늘 실습에서는 '단순한 구조'로 안전하게 구현하는 게 목표

쓰레드 디버깅 팁

- 로그를 남겨 흐름 확인
- join으로 종료를 보장
- 데드락(서로 기다림) 주의

핵심 포인트

문제가 생기면 '언제 어떤 순서로 실행됐는지'가 핵심
print 대신 logging을 쓰면 더 좋지만, 오늘은 개념만
AI에게: 재현 방법 + 최소 코드로 줄여달라고 요청 가능

미니게임 스켈레톤: 제한 시간 퀴즈

```
import threading, time
timeout = False
def timer():
    global timeout
    time.sleep(5)
    timeout = True

threading.Thread(target=timer,
daemon=True).start()
ans = input('5초 안에 입력: ')
if timeout:
    print('시간 초과!')
else:
    print('입력 성공')
```

핵심 포인트

입력 대기 중에도 timer 쓰레드가 돌아간다.

timeout 공유 변수 → Event로 바꾸면 더 안전(선택)

With AI: 'timeout 경쟁 상태' 가능성을 분석해보게 하자

With AI: 쓰레드 코드 생성 프롬프트 템플릿

역할: 너는 파이썬 동시성 전문가다.
목표: (미니게임/타이머/로딩) 기능을 쓰레드로 구현하고 싶다.
조건: 공유 데이터는 최소화하고, 필요하면 Lock/Event/Queue를 사용
요청: (1) 설계 설명(상태/흐름) (2) 코드 (3) 테스트 시나리오 (4) 위험 요소(race) 점검
추가: 내 코드가 있으면 버그 재현과 수정안을 제시해줘.

실습 1: 백그라운드 타이머 만들기

문제: 1초마다 'tick'을 출력하는 쓰레드를 만들고, 5초 후 종료하라.
힌트: daemon 또는 Event 사용
제출: 코드 + 동작 설명 3문장
With AI 팁: '종료 방식(플래그/이벤트)'을 비교해보자.

실습 2: 디버깅 — 경쟁 상태 재현

```
import threading
count = 0
def inc():
    global count
    for _ in range(100000):
        count += 1
threads = [threading.Thread(target=inc) for
_ in range(2)]
[t.start() for t in threads]
[t.join() for t in threads]
print(count)  # 기대: 200000
```

핵심 포인트

결과가 200000보다 작게 나올 수 있다(race).
Lock을 적용해 값이 안정적으로 나오는지 확인하라.
AI에게: 왜 이런 현상이 생기는지(원자성) 설명 요청

실습 3: 리팩터링 — Queue로 안전하게

```
from queue import Queue
# TODO: 작업 쓰레드가 결과를 Queue에 put
# 메인 쓰레드가 get해서 출력
```

핵심 포인트

공유 변수를 직접 수정하는 대신 메시지로 전달
구조가 명확해지고 디버깅이 쉬워진다.
AI에게: 생산자-소비자 구조를 코드로 만들어달라고 요청

연습문제(쓰레드)

1) 쓰레드 2개를 만들어 각각 다른 문장을 출력하라.
2) join을 사용해 메인에서 종료를 기다려라.
3) Lock을 사용해 공유 변수(count) 증가를 안전하게 만들어라.
4) Queue를 사용해 쓰레드가 만든 결과를 메인으로 전달하라.

미니 퀴즈(5문항)

문항
1) start()의 역할은?
A. 쓰레드 객체 생성 B. 쓰레드 실행 시작 C. 종료 대기
2) join()의 역할은?
A. 실행 시작 B. 결과 출력 C. 종료까지 기다림
3) race condition이란?
A. 속도 경주 B. 실행 순서에 따라 결과가 달라짐 C. 문법 오류
4) 공유 자원을 보호하는 도구는?
A. Lock B. sort C. split
5) Queue의 장점은?
A. 자동 정렬 B. thread-safe 통신 C. 파일 저장

요약 & 다음 주 예고

오늘의 핵심

- threading.Thread: start/join
- 공유 데이터는 race condition 위험 → Lock/Event/Queue
- 미니게임에 타이머/효과를 추가해 '동시 실행' 경험

다음 주(Week 14): 개인별 창의적 프로그램 과제 발표