python #8

▼ 문제

- 다음 요구 사항에 맞춰서 코드를 작성하고 결과를 비교
- 두 가지 버전의 함수를 작성
  (함수 :
  입력 : 정수 리스트 → 출력 : 각 원소의 제곱을 더한 합, 최소 10,000까지 진행)
  - A 버전 : 타입 힌트를 사용하지 않은 함수
  - B 버전 : 타입 힌트를 적용한 함수
- mypy를 이용해 버전 B의 타입을 검사해보고, 결과를 확인
- timeit을 사용하여 두 버전의 실행 성능을 각각 측정하고 성능 차이를 비교

▼ 코드

▼ 전체 코드

```
import timeit
import random
from typing import List

# timeit을 사용하여 실행 성능 측정
def measure_time(func): # 함수를 실행하고 소요 시간을 측정
    def wrapper(*args, **kwargs):
        elapsed_time = timeit.timeit(lambda: func(*args, **kwargs), number=100) # 전달받은 함수를 100번 실행하고 총 소요 시간을 반환
        return elapsed_time # 측정된 총 실행 시간 반환
    return wrapper # wrapper 실행, wrapper 내부에서 시간 측정 후 원래 함수를 호출

# A 버전 : 타입 힌트를 사용하지 않은 함수
def sum_of_squares_no_type(lst): # 인자는 lst: 자료형 제한 없음 (타입 힌트 없음)
    return sum(x * x for x in lst) # 각 원소를 제곱하고 모두 더함

# B 버전 : 타입 힌트를 적용한 함수
def sum_of_squares_typed(lst: List[int]) -> int: # 인자 lst는 정수(int) 요소들로 이루어진 리스트, 반환값은 정수 (타입 힌트)
    return sum(x * x for x in lst) # 각 원소를 제곱하고 모두 더함

# 각 원소 제곱 후 합산 함수
@measure_time # 데코레이터: run_no_type 실행시 자동으로 실행 시간을 측정
def run_no_type(test_data): # test data: 성능 측정 대상 데이터(정수 리스트)
    sum_of_squares_no_type(test_data) # test_data 리스트 내 각 원소 제곱 후 합산

@measure_time # 데코레이터: run_typed 실행시 자동으로 실행 시간을 측정
def run_typed(test_data): # test data: 성능 측정 대상 데이터(정수 리스트)
    sum_of_squares_typed(test_data) # test_data 리스트 내 각 원소 제곱 후 합산

if __name__ == "__main__":
    # 테스트 데이터 생성
    test_data = [random.randint(1, 100) for _ in range(1_000_000)]

    # 실행 시간 측정
    time_no_type = run_no_type(test_data) # test_data 내 원소 제곱 후 합산 -> @measure_time 데코레이터로 실행 시간 측정
```

```
    time_typed = run_typed(test_data)

    print(f"No Type Hint Time : {time_no_type:.4f} sec") # 타입 힌트 없는 버전 결과 출력
    print(f"With Type Hint Time : {time_typed:.4f} sec") # 타입 힌트 있는 버전으로 결과 출력
```

▼ measure_time 데코레이터

```
# timeit을 사용하여 실행 성능 측정
def measure_time(func): # 함수를 실행하고 소요 시간을 측정
    def wrapper(*args, **kwargs):
        elapsed_time = timeit.timeit(lambda: func(*args, **kwargs), number=100) # 전달받은 함수를 100번 실행하고 총 소요 시간을 반환
        return elapsed_time # 측정된 총 실행 시간 반환
    return wrapper # wrapper 실행, wrapper 내부에서 시간 측정 후 원래 함수를 호출
```

- measure_time(func)
  - func: 데코레이터가 적용될 함수 객체
- wrapper
  - timeit.timeit()으로 원래 함수(func)를 100번 실행하고 걸린 시간을 반환.
  - (lambda: func(*args, **kwargs), number=100)
    - 전달받은 함수를 100번 실행하고 총 소요 시간을 반환
  - return elapsed_time
    - 측정된 총 실행 시간 반환
- return wrapper
  - wrapper 함수를 반환: measure 함수가 원래 함수 대신 wrapper 실행해서 wrapper 내부에서 시간 측정 후 원래 함수를 호출.

▼ sum_of_squares_no_type, sum_of_squares_typed

```
# A 버전 : 타입 힌트를 사용하지 않은 함수
def sum_of_squares_no_type(lst): # 인자는 lst: 자료형 제한 없음 (타입 힌트 없음)
    return sum(x * x for x in lst) # 각 원소를 제곱하고 모두 더함

# B 버전 : 타입 힌트를 적용한 함수
def sum_of_squares_typed(lst: List[int]) -> int: # 인자 lst는 정수(int) 요소들로 이루어진 리스트, 반환값은 정수 (타입 힌트)
    return sum(x * x for x in lst) # 각 원소를 제곱하고 모두 더함
```

- sum_of_squares_no_type
  - 인자 `lst`
    - 자료형 제한 없음, int/float/str 다 가능
  - sum(x * x for x in lst)
    - 각 원소를 제곱하고 모두 더함
- sum_of_squares_typed
  - 인자 `lst: List[int]) -> int`
    - `lst: List[int])` : 인자 lst는 정수(int) 요소들로 이루어진 리스트여야 함 (타입 힌트 적용)
    - `-> int` : 반환값은 정수.
  - sum(x * x for x in lst)

        - 각 원소를 제곱하고 모두 더함

▼ run_no_type, run_typed

```
# 성능 비교용 함수들
@measure_time # 데코레이터: run_no_type 실행시 자동으로 실행 시간을 측정
def run_no_type(test_data): # test data: 성능 측정 대상 데이터(정수 리스트)
    sum_of_squares_no_type(test_data) # test_data 리스트 내 각 원소 제곱 후 합산

@measure_time # 데코레이터: run_typed 실행시 자동으로 실행 시간을 측정
def run_typed(test_data): # test data: 성능 측정 대상 데이터(정수 리스트)
    sum_of_squares_typed(test_data) # test_data 리스트 내 각 원소 제곱 후 합산
```

- @measure_time
    - run_no_type이 실행될 때 자동으로 실행 시간을 측정
- test_data
    - 성능 측정 대상 데이터(정수 리스트)
- sum_of_squares_no_type, sum_of_squares_typed
    - test_data 리스트 내 각 원소 제곱 후 합산

▼ __main__

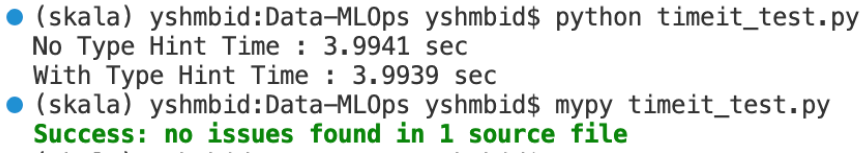
```
if __name__ == "__main__":
    # 테스트 데이터 생성
    test_data = [random.randint(1, 100) for _ in range(1_000_000)]

    # 실행 시간 측정
    time_no_type = run_no_type(test_data) # test_data 내 원소 제곱 후 합산 -> @measure_time 데코레이터로 실행 시간 측정
    time_typed = run_typed(test_data)

    print(f"No Type Hint Time : {time_no_type:.4f} sec") # 타입 힌트 없는 버전 결과 출력
    print(f"With Type Hint Time : {time_typed:.4f} sec") # 타입 힌트 있는 버전으로 결과 출력
```

- [random.randint(1, 100) for _ in range(1_000_000)]
    - 길이 1,000,000의 리스트 생성
    - 값은 1~99 정수
- run_no_type, run_typed
    - 타입 힌트 없는 버전과 있는 버전으로 test_data 리스트 내 각 원소 제곱 후 합산 수행
    - @measure_time 데코레이터로 run_no_type과 run_typed이 실행될 때 자동으로 실행 시간을 측정

▼ **수행 결과**



▼ **결과 설명**

1. 소요 시간 측정
    - 버전 A - 3.9941 sec, 버전 B - 3.9939 sec 으로 두 값은 거의 동일했습니다.

2. mypy 결과
    - mypy가 `sum_of_squares_typed(lst: List[int]) -> int (정수들로 이뤄진 리스트를 받고, 정수를 반환한다)` 를 확인 결과 위반 없음 (Success).