1. LRU Cache + optymalizacja rekursji

Napisz funkcję `count_paths(n, m)`, która oblicza liczbę różnych ścieżek z lewego górnego rogu tablicy `n × m` do prawego dolnego, poruszając się tylko w dół i w prawo.
Zoptymalizuj obliczenia za pomocą `@lru_cache`.
Warunek:

- Funkcja ma działać **wydajnie** dla wartości `n` i `m` nawet rzędu 100.
- Zbadaj i wydrukuj, ile razy wywołano faktycznie *ciało* funkcji.

---

2. Dekorator log execution

Zadanie:
Napisz dekorator `log_execution_time`, który:

- wypisuje w konsoli czas wykonania dekorowanej funkcji,
- zachowuje nazwę i docstring oryginalnej funkcji dzięki `@wraps`.

Warunek:

- Pokaż, że bez `wraps` nazwa i docstring ulegają zmianie.
- Użyj dekoratora do funkcji liczącej dużą liczbę Fibonacciego z `lru_cache`.

---

3. Funkcja częściowo aplikowana (`partial`)

Masz funkcję:

```
def power(base, exponent):
    return base ** exponent
```

Stwórz funkcje `square`, `cube` i `sqrt` za pomocą `partial`.
Następnie użyj ich w programie, który:

- przyjmuje listę liczb od użytkownika,
- używa wybranej wersji funkcji do przekształcenia wszystkich elementów.

---

4. Połączenie `partial` + dekorator

Napisz dekorator `log_calls(prefix="CALL")`, który:

- wypisuje w konsoli `<prefix>: nazwa_funkcji(arg1, arg2, ...) -> wynik`
- pozwala ustawić `prefix` za pomocą `partial`, np.:

```
@special_log
def add(a, b):
    return a + b
```

gdzie `special_log = partial(log_calls, prefix="[DEBUG]")`

---

5. Reduce

Napisz program, który:

1. Ma listę słowników z informacjami o zamówieniach, np.:

```
orders = [
    {"product": "Laptop", "price": 3500, "quantity": 2},
    {"product": "Mouse", "price": 80, "quantity": 5},
    {"product": "Keyboard", "price": 200, "quantity": 3},
    {"product": "Monitor", "price": 900, "quantity": 2},
]
```

2. Za pomocą `functools.reduce` oblicza **łączną wartość wszystkich zamówień** (`price * quantity` dla każdego produktu).

3. **Dodatkowo:** znajdź produkt, który w sumie kosztował najwięcej (czyli `price * quantity` maksymalne).