- 뼈대 코드와 테스트 케이스가 담겨있는 exam2.py 파일을 다운받아 코드를 작성합니다.
- 파일 이름을 exam2-nnnn.py로 수정하여 제출합니다. 여기서 nnnn은 자신의 학번의 뒤 4자리입니다.

[1번 문제를 풀기 위한 연습문제]

중첩 리스트는 리스트가 리스트의 원소가 되는 리스트를 말한다. 중첩 리스트의 사례는 다음과 같다.

```
[1,[],3]
[1,[1,2,[3,4]]]
[[[[[[[[1,2,3]]]]]]],4]
```

중첩 정수리스트에 정수 원소가 몇 개 있는지 세는 count_elements 함수는 for 루프를 사용하여 다음과 같이 작성할 수 있다.

```
def count_elements(xss):
    counter = 0
    for xs in xss:
        if isinstance(xs,list):
            counter += count_elements(xs)
        else:
            counter += 1
    return counter
```

파일에 준비된 테스트케이스를 활용하여 이 함수를 실행하면서 이해해보자.

라이브러리 함수 호출 isinstance(xs,list)는 첫 인수 xs가 리스트인지를 확인해주며 다음과 같이 작동한다.

```
>>> isinstance(3,list)
False
>>> isinstance([],list)
True
>>> isinstance([3,4],list)
True
```

1. [예상 소요 시간 : 약 15분]

[3점] 중첩 리스트를 받아서 내부 원소만 나열한 리스트로 만들어 내주는 flatten 함수를 위의 count_elements 함수와 비슷한 형태로 for 루프를 사용하여 작성하시오. 실행 사례는 다음과 같다.

```
>>> flatten([])
[]
>>> flatten([1,2,3])
[1, 2, 3]
>>> flatten([1,[],3])
[1, 3]
>>> flatten([1,[1,2,[3,4]]])
[1, 1, 2, 3, 4]
>>> flatten([[[[[[[[1,2,3]]]]]]]])
[1, 2, 3]
>>> flatten([[[[3]],[4]],5,6,[7]])
[3, 4, 5, 6, 7]
>>> flatten([1,[2,2],[[3],[4,4]],[[[5,5,5,5]]],6,[7,[[8],[[9]]]]])
[1, 2, 2, 3, 4, 4, 5, 5, 5, 5, 6, 7, 8, 9]
```

2. [예상 소요 시간 : 약 15분]

[3점] 정방행렬(square matrix)은 행과 열의 개수가 동일한 다음과 같은 모양의 행렬을 말한다.

n_{00}	n_{01}	n_{02}	n_{03}
n_{10}	n_{11}	n_{12}	n_{13}
n_{20}	n_{21}	n_{22}	n_{23}
n_{30}	n_{31}	n_{32}	n_{33}

이 정방행렬은 다음과 같이 리스트의 리스트로 표현할 수 있다.

[[n_{00}, n_{01}, n_{032}, n_{03}], [n_{10}, n_{11}, n_{12}, n_{13}], [n_{20}, n_{21}, n_{22}, n_{23}], [n_{30}, n_{31}, n_{32}, n_{33}]]

대각선이 모두 1이고, 나머지가 모두 0인 정방행렬을 항등행렬(identity matrix)라고 한다.
다음은 크기가 4인 정방행렬을 리스트로 표현한 것이다.

```
[[1, 0, 0, 0], [0, 1, 0, 0], [0, 0, 1, 0], [0, 0, 0, 1]]
```

임의 크기의 정방행렬을 인수로 받아서 항등행렬인지 확인하여 `True` 또는 `False`로 답해주는 함수 `identity`를 작성하시오.

실행사례:

```
xs0 = [[1,0,0,0],[0,1,0,0],[0,0,1,0],[0,0,0,1]]

xs1 = [[1,0,0,0],[0,2,0,0],[0,0,1,0],[0,0,0,1]]

xs2 = [[0,-3,6,4],[3,0,-9,8],[-6,9,0,2],[-4,-8,-2,0]]

>>> identity(xs0)
True
>>> identity(xs1)
False
>>> identity(xs1)
False
```

3. [예상 소요 시간 : 약 40분]

```
def ascii_art(n):
    for i in range(n):
        for j in range(n):
            print('#', end=' ')
        print()
```

ascii_art(5)를 실행하면 다음과 같이 실행창에 출력한다.

```
# # # # #
# # # # #
# # # # #
# # # # #
# # # # #
```

위 함수를 적절히 수정하여 각 출력 요구사항에 맞게 코드를 작성하시오.
인수는 <u>홀수</u>만 들어온다고 가정한다. 즉, 짝수 인수는 고려하지 않아도 된다.

(1) [3점] ascii_art_cross(5)를 호출하면 다음과 같이 출력하는 ascii_art_cross 함수를 작성하시오.

```
    #
    #
# # # # #
    #
    #
```

(2) [3점] ascii_art_X(5)를 호출하면 다음과 같이 출력하는 ascii_art_X 함수를 작성하시오.

```
#       #
  #   #
    #
  #   #
#       #
```

(3) [3점] ascii_art_square(5)를 호출하면 다음과 같이 출력하는 ascii_art_square 함수를 작성하시오.

```
# # # # #
#       #
#       #
#       #
# # # # #
```

(4) [3점] ascii_art_diamond(5)를 호출하면 다음과 같이 출력하는 ascii_art_diamond 함수를 작성하시오.

```
    #
  #   #
#       #
  #   #
    #
```

4. [예상 소요 시간 : 약 40분]

부분문자열은 문자열의 앞뒤를 임의로 떼어낸 문자열 말한다. 예를 들어, "RI"는 "ERICA"의 부분문자열이다. 전체를 떼어낸 빈 문자열도 부분문자열로 보고, 하나도 떼어내지 않은 문자열 전체도 부분문자열로 본다. 그렇게 따져서 "ERICA"의 부분문자열을 모두 나열하면 다음과 같다.

"", "E", "R", "I", "C", "A", "ER", "RI", "IC", "CA", "ERI", "RIC", "ICA", "ERIC", "RICA", "ERICA"

(1) [3점] 정수 k와 문자열 s를 인수로 받아서 길이가 k인 s의 부분문자열을 모두 찾아주는 함수 substrings_of_length를 작성하시오. 실행 결과는 다음과 같아야 한다.

```
>>> substrings_of_length(0,"ERICA")
['']
>>> substrings_of_length(1,"ERICA")
['E', 'R', 'I', 'C', 'A']
>>> substrings_of_length(2,"ERICA")
['ER', 'RI', 'IC', 'CA']
>>> substrings_of_length(3,"ERICA")
['ERI', 'RIC', 'ICA']
>>> substrings_of_length(4,"ERICA")
['ERIC', 'RICA']
>>> substrings_of_length(5,"ERICA"]
['ERICA']
>>> substrings_of_length(6,"ERICA")
None
```

(2) [3점] 임의의 문자열을 받아서 부분문자열을 모두 찾아서 리스트로 모아서 내주는 함수 substrings를 작성하시오. (1)번 문제에서 작성한 substrings_of_length 함수를 사용해도 좋다.

```
>>> substrings("ERICA")
['', 'E', 'R', 'I', 'C', 'A', 'ER', 'RI', 'IC', 'CA', 'ERI', 'RIC', 'ICA', 'ERIC',
'RICA', 'ERICA']
>>> substrings("")
['']
```

5. [예상 소요 시간 : 약 40분]

순수하게 증가하는 (뒷 수가 앞 수보다 항상 큰) 리스트를 **오르막 리스트**라고 하자. 그리고, 오르막 리스트가 되려면 최소한 원소가 2개 이상은 있어야 한다고 하자. 예를 들어 다음의 리스트는 모두 오르막 리스트이다.

```
[1, 3]
[2, 3, 6, 8, 12, 17]
```

그러나 다음 정수 리스트는 오르막 리스트가 아니다.

```
[]
[2]
[3, 3, 3, 3, 3]
[1, 2, 2, 3]
[2, 3, 1]
```

부분리스트는 리스트의 일부를 떼어낸 리스트를 말한다. 예를 들어, [3, 4, 5]는 [1, 2, 3, 4, 5, 6]의 부분리스트이다. 전체를 떼어낸 빈 리스트도 부분리스트로 보고, 하나도 떼어내지 않은 리스트 전체도 부분리스트로 본다. [1, 2, 3]의 부분리스트를 모두 찾아보면, [], [1], [2], [3], [1, 2], [2, 3], [1, 2, 3] 이다.

(1) [3점] 임의의 정수 리스트를 받아서 **가장 긴 오르막 부분리스트**를 찾아 내주는 함수 longest_ascending_sublist를 작성하시오. 여럿 있으면 맨 처음 나오는 것을 선택한다.

```
>>> longest_ascending_sublist([1,5,3,4,8,2,3,5])
[3, 4, 8]
>>> longest_ascending_sublist([])
[]
>>> sample = [59, 4, 38, 54, 33, 75, 19, 73, 49, 7, 86, 28, 54, 13, 6, 42, 97, 84, 26,
69, 86, 14, 79, 27, 68, 57, 35, 53, 92, 58, 68, 49, 93, 28, 31, 63, 51, 1, 44, 62, 14,
40, 53, 40, 5, 69, 81, 95, 58, 55, 90, 56, 91, 40, 55, 14, 65, 28, 37, 61, 66, 89, 26,
63, 98, 59, 7, 23, 34, 67, 77, 30, 49, 55, 31, 58, 10, 27, 15, 45, 42, 77, 11, 14, 9, 55,
88, 44, 53, 12, 54, 95, 25, 91, 29, 8, 25, 90, 34, 55]
>>> longest_ascending_sublist(sample)
[28, 37, 61, 66, 89]
```

(2) [3점] 오르막 리스트에서 기울기는 마지막 수에서 첫 수를 뺀 값에 리스트의 길이를 나누어 구한다. 즉, [1, 2]의 기울기는 (2-1)/2 = 0.5, [1, 3, 6]의 기울기는 (6-1)/3 = 2.6666.., [2, 3, 6, 8, 12, 17]의 기울기는 (17-2)/6 = 2.5 이다. 임의의 정수 리스트를 받아서 **가장 길면서 기울기가 가장 가파른 오르막 부분리스트**를 찾아 내주는 함수 longest_steepest_ascending_sublist를 작성하시오.

```
>>> longest_steepest_ascending_sublist([1,5,3,4,8,2,3,5])
[3, 4, 8]
>>> longest_steepest_ascending_sublist([])
[]
>>> sample = [59, 4, 38, 54, 33, 75, 19, 73, 49, 7, 86, 28, 54, 13, 6, 42, 97, 84, 26,
69, 86, 14, 79, 27, 68, 57, 35, 53, 92, 58, 68, 49, 93, 28, 31, 63, 51, 1, 44, 62, 14,
40, 53, 40, 5, 69, 81, 95, 58, 55, 90, 56, 91, 40, 55, 14, 65, 28, 37, 61, 66, 89, 26,
63, 98, 59, 7, 23, 34, 67, 77, 30, 49, 55, 31, 58, 10, 27, 15, 45, 42, 77, 11, 14, 9, 55,
88, 44, 53, 12, 54, 95, 25, 91, 29, 8, 25, 90, 34, 55]
>>> longest_steepest_ascending_sublist(sample)
[7, 23, 34, 67, 77]
```