Professional Coding Specialist

COS Pro 파이썬 1 급

19 강-20 강. 모의고사 4 차

1. 모의고사 4 차(1-10 번)

과정 소개

Cos Pro 1 급 파이썬 4 차 문제를 풀어보며 문제 유형을 익히고, 파이썬을 이용하여 알고리즘을 구현하기 위해 필요한 관련 지식을 익혀보도록 한다.

학습 목차

1. 문제 1
2. 문제 2
3. 문제 3
4. 문제 4
5. 문제 5
6. 문제 6
7. 문제 7
8. 문제 8
9. 문제 9
10. 문제 10

학습 목표

1. YBM IT(www.ybmit.com) 에서 제공하는 COS Pro 1 급 파이썬 샘플 문제를 풀어보며 파이썬을 이용하여 주어진 문제를 해결하기 위한 알고리즘을 구성하는 능력을 배양한다.
2. 많이 등장하는 문제 유형을 익혀서 COS Pro 1 급 시험에 대비한다.

1. 문제 1

1) 문제 코드

```python
words = []

def create_words(lev, s):
    global words
    VOWELS = ['A', 'E', 'I', 'O', 'U']
    words.append(s)
    for i in range(0, 5):
        if lev < 5:
            create_words(lev, s + VOWELS[i])

def solution(word):
    global words
    words = []
    answer = 0
    create_words(0, '')
    for idx, i in enumerate(words):
        if word == i:
            answer = idx
            break
    return answer

# 아래는 테스트케이스 출력을 해보기 위한 코드입니다. 아래에는 잘못된 부
word1 = "AAAAE"
ret1 = solution(word1)
# [실행] 버튼을 누르면 출력 값을 볼 수 있습니다.
print("solution 함수의 반환 값은 ", ret1, " 입니다.")

word2 = "AAAE"
ret2 = solution(word2)
# [실행] 버튼을 누르면 출력 값을 볼 수 있습니다.
print("solution 함수의 반환 값은 ", ret2, " 입니다.")
```

2) 문제 개요

- 제시된 과제가 바르게 수행되도록 문제 코드를 수정하는 문제.
- 재귀 호출 함수를 이용하여 주어진 알파벳으로 길이가 5 자 이하의 단어들을 생성한 후 제시된 단어가 몇 번째 단어인지 알아내는 프로그램에서 잘못된 곳을 찾아 한 줄 수정하는 문제.
- 사전의 첫 번째 단어는 'A', 두 번째 단어는 'AA', 세 번째 단어는 'AAA' 순으로 단어를 생성하여 마지막 단어는 'UUUUU' 로 마무리.

3) 재귀 함수

① 특징

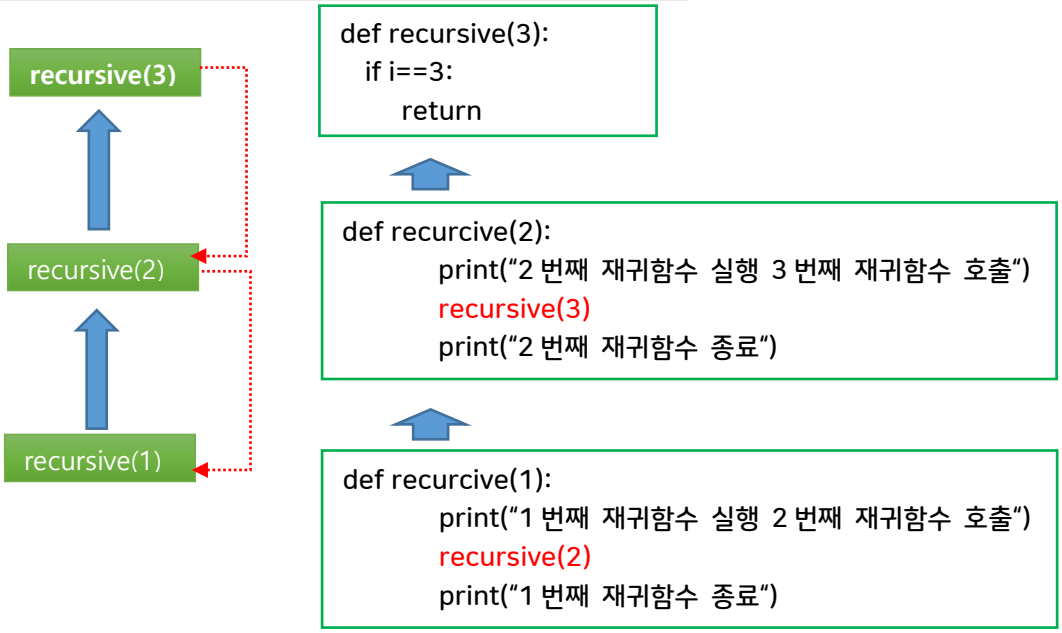
- 자기 자신을 호출하는 함수.
- 호출된 함수 실행이 끝나면 마지막으로 호출한 함수부터 꺼내는 스택(Stack) 방식.
- 재귀 함수를 호출할 때마다 함수 안의 지역 변수를 새로 생성하므로 지나친 재귀호출은 메모리를 과다 사용할 수 있고, 함수 호출/리턴하는 과정에서 프로세스 교환이 빈번히 일어나므로 성능 저하를 일으킬 수 있음.

② 재귀 함수 예

```
def recursive(i):
  if i==3:
    return

  print(i,"번째 재귀함수 실행", i+1,'번째 재귀함수 호출')
  recursive(i+1)
  print(i,'번째 재귀함수 종료')

recursive(1)
```



➔ recursive(2) 에서 recursive(3) 를 호출하기 때문에 recursive(3) 종료 후 recursive(2)에서 recursive(3) 를 호출한 다음 명령으로 돌아감.

➔ recursive(1) 에서 recursive(2) 를 호출하기 때문에 recursive(2) 종료 후 recursive(1)에서 recursive(2) 를 호출한 다음 명령으로 돌아감.

<실행 결과>

```
1 번째 재귀함수 실행 2 번째 재귀함수 호출
2 번째 재귀함수 실행 3 번째 재귀함수 호출
2 번째 재귀함수 종료
1 번째 재귀함수 종료
```

③ 꼬리 재귀(Tail Recursion)

- 재귀 함수 실행이 종료된 후 재귀 호출을 한 함수 내에서 재귀 함수의 결과에 대해 추가 연산을 할 필요가 없도록 구현한 방식.

- 재귀 함수 형태로 작성된 코드이지만, 꼬리 재귀로 작성된 것을 컴파일러가 인식하면 그 부분을 반복문 형태로 바꾸어 재귀 호출에 의한 메모리 과다 사용과 성능 저하 문제를 해결.

Tail Recursion 을 이용한 팩토리얼

```
def f(n,a):
    if (n == 0):
        return a
    print(n,a," 실행")

    return f(n-1,n*a)

print(f(4,1))
```

< 결과 >

```
4 1  실행
3 4  실행
2 12  실행
1 24  실행
24
```

Non Tail Recursion 을 이용한 팩토리얼

```
def f(n):
    if (n <= 1):
        return 1
    print(n, "! 실행")

    return n*f(n-1)

print(f(4))
```

< 결과 >

```
4 ! 실행
3 ! 실행
2 ! 실행
24
```

➔ Tail Recursion 을 이용한 팩토리얼 구하기 : 재귀 함수의 리턴 값을 이용한 연산이 없음.

➔ Non Tail Recursion 을 이용한 팩토리얼 구하기 : 재귀함수의 리턴 값을 이용한 연산이 있기 때문에 재귀 호출에 대한 정보를 스택에 저장해야 함.

4) 정답

- 사전식 단어 생성 절차

생성 순서 \ 글자 수	1	2	3	4	5
1	A				
2	A	A			
3	A	A	A		
4	A	A	A	A	
5	A	A	A	A	A
6	A	A	A	A	E
7	A	A	A	A	I
8	A	A	A	A	O
9	A	A	A	A	U
10	A	A	A	E	
11	A	A	A	E	A

생성 단어 수

1 글자 수 단어 (5)
+ 2 글자 수 단어 (5*5)
+ 3 글자 수 단어 (5*5*5)
+ 4 글자 수 단어 (5*5*5*5)
+ 5 글자 수 단어 (5*5*5*5*5)
= 3905 개

...	...

- 정답 코드

```
words = []

def create_words(lev, s):
    global words
    VOWELS = ['A', 'E', 'I', 'O', 'U']
①   words.append(s)
②   for i in range(0, 5):
        if lev < 5:
            create_words(lev + 1, s + VOWELS[i])

def solution(word):
    global words
    words = []
    answer = 0
    create_words(0, '')
    for idx, i in enumerate(words):
③       if word == i:
            answer = idx
            break
    return answer

# 아래는 테스트케이스 출력을 해보기 위한 코드입니다. 아래에는 잘못된
word1 = "AAAAE"
ret1 = solution(word1)
# [실행] 버튼을 누르면 출력 값을 볼 수 있습니다.
print("solution 함수의 반환 값은 ", ret1, " 입니다.")

word2 = "AAAE"
ret2 = solution(word2)
# [실행] 버튼을 누르면 출력 값을 볼 수 있습니다.
print("solution 함수의 반환 값은 ", ret2, " 입니다.")
```

① 매개변수 s 에 있는 문자열을 words 리스트에 추가.

② for 문 반복을 이용하여 VOWELS 의 알파벳으로 구성된 5 자 이하의 단어를 생성 : 사전순으로 조합하여 단어를 생성.

- 매개변수 lev 의 값이 5 보다 작으면(현재 s 가 갖고 있는 문자열의 길이가 5 보다 작으면) 재귀 호출.
- 재귀호출 할 때, lev 의 값을 1 만큼 증가시킨 값과 매개변수 s 에 있는 문자열에 VOWELS[i] 번째 문자를 합한 것을 인수로 전달.
- 문제코드에서 제시한 것처럼 재귀호출 할 때, lev 값을 그대로 전달하면 무한하게 재귀호출하는 문제가 발생함.

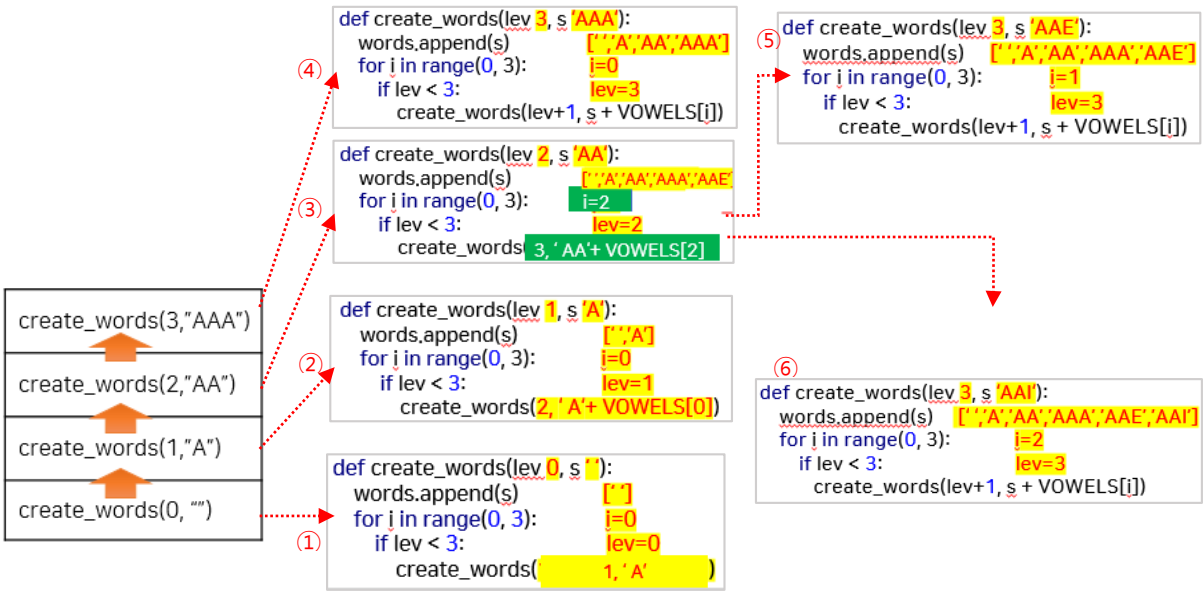
③ 5 자 이하의 단어들을 순서대로 모아 놓은 리스트 words 에서 enumerate( ) 함수를 이용하여 인덱스와 단어를 가져와 idx 와 i 에 저장.

- 매개변수 word 와 항목값 i 가 같으면 해당 항목을 가리키는 idx 를 answer 로 저장한 후 break 명령을 이용하여 반복문 실행 종료.

5) 정답 코드의 create_words( ) 함수의 실행 process

: 알파벳 3 개 'A', E', 'I' 만을 사용하여 단어 만드는 경우

```
def create_words(lev, s):
    global words
    VOWELS = ['A', 'E', 'I']
    words.append(s)
    for i in range(0, 3):
        if lev < 3:
            create_words(lev+1, s + VOWELS[i])
```



① create_words(0," ")를 실행하면 words 에 " "을 추가하고 create_words(1,"A") 호출.

② create_words(1,"A")를 실행하면 words 에 "**A**"을 추가하고 create_words(2,"AA") 호출.

③ create_words(2,"AA")를 실행하면 words 에 "**AA**"을 추가하고 create_words(3,"AAA") 호출.

④ create_words(3,"AAA")를 실행하면 words 에 **"AAA"**을 추가하고 종료.

➔ create_words(3,"AAA")가 종료되면 recursion stack 에서 create_words(3, "AAA") 관련 정보를 삭제하고, create_words(2, "AA")로 돌아가서 create_words(3,"AAE") 호출.

⑤. create_words(3,"AAE")를 실행하면 words 에 **"AAE"**을 추가하고 종료.

➔ create_words(3,"AAE")가 종료되면 recursion stack 에서 create_words(3, "AAE") 관련 정보를 삭제하고, create_words(2, "AA")로 돌아가서 create_words(3,"AAI") 호출.

⑥. create_words(3,"AAI")를 실행하면 words 에 **"AAI"**을 추가하고 종료.

➔ create_words(3,"AAI")가 종료되면 recursion stack 에서 create_words(3, "AAI") 관련 정보를 삭제한 후 create_words(2, "AA")로 돌아가고, create_words(2,"AA") 안에서 for 문에 의한 모든 반복이 실행되었으므로 create_words(2,"AA")를 종료.

➔ create_words(2,"AA")가 종료되면 recursion stack 에서 create_words(2,"AA") 관련 정보를 삭제하고, create_words(1,"A")로 돌아가서 create_words(2,"AE") 호출.

➔ 이러한 과정을 create_words(3,"III")를 실행할 때까지 진행.

6) 브루트 포스 방식으로 구현한 사전식 단어 구성

```
def solution(word):
    global words
    words = ['']
    answer = 0
    VOWELS = ['A', 'E', 'I', 'O', 'U']

    for i1 in range(0, 5):
        words.append( VOWELS[i1])
①       for i2 in range(0,5):
            words.append(VOWELS[i1] + VOWELS[i2])
②           for i3 in range(0, 5):
                words.append(VOWELS[i1] + VOWELS[i2]+VOWELS[i3])
③               for i4 in range(0, 5):
                    words.append(VOWELS[i1] + VOWELS[i2] + VOWELS[i3]+VOWELS[i4])
④                   for i5 in range(0, 5):
                        words.append(VOWELS[i1] + VOWELS[i2] + VOWELS[i3] + VOWELS[i4]+ VOWELS[i5])
```

⑤

- 5 개의 중첩 for 문을 이용하여 1 글자 단어부터 5 글자 단어까지 사전 순으로 생성하여 words 에 저장.
  - ①. 1 글자 단어 추가
  - ②. 2 글자 단어 추가
  - ③. 3 글자 단어 추가
  - ④. 4 글자 단어 추가
  - ⑤. 5 글자 단어 추가
- 재귀 함수로 구현할 때보다 실행 속도가 빠름.

2. 문제 2

1) 문제 코드

```python
def solution(s):
    s = s.lower()
    answer = ""
    previous = s[0]
    counter = 1
    for alphabet in s[1:]:
        if alphabet == previous:
            counter += 1
        else:
            answer += previous + str(counter)
            counter = 1
            previous = s[0]
    answer += previous + str(counter)
    return answer

# 아래는 테스트케이스 출력을 해보기 위한 코드입니다. 아래에는
s = "YYYYYbbbBbbBBBMmmM"
ret = solution(s)

# [실행] 버튼을 누르면 출력 값을 볼 수 있습니다.
print("solution 함수의 반환 값은", ret, "입니다.")
```

2) 문제 개요

- 제시된 과제가 바르게 수행되도록 문제 코드를 수정하는 문제.
- 여러 개의 문자가 붙어 있는 단어를 문자와 빈도수로 압축하여 표현하는 문제에서 잘못된 곳을 찾아 수정하는 문제.

3) 정답

```python
def solution(s):
    s = s.lower()
    answer = ""
①   previous = s[0]
    counter = 1
    for alphabet in s[1:]:
        if alphabet == previous:
            counter += 1
②       else:
            answer += previous + str(counter)
            counter = 1
            previous = alphabet
③   answer += previous + str(counter)
    return answer

# 아래는 테스트케이스 출력을 해보기 위한 코드입니다. 아래에는
s = "YYYYYbbbBbbBBBMmmM"
ret = solution(s)

# [실행] 버튼을 누르면 출력 값을 볼 수 있습니다.
print("solution 함수의 반환 값은", ret, "입니다.")
```

① 이전 문자를 저장하는 변수 previous 를 s[0]로 초기화.

② for 문을 이용하여 s[1]부터 마지막 문자까지 차례대로 한 글자씩 alphabet 에 받아와서 previous 와 같은 지 비교.

- alphabet 과 previous 가 같으면, 반복되는 횟수를 저장하는 변수 counter 를 증가.

- alphabet 과 previous 가 같지 않으면, previous 와 previous 의 빈도수를 저장하고 있는 counter 를 문자열로 바꿔서 붙인 후 answer 변수에 더하고 counter 는 1 로, previous 는 alphabet 으로 재설정.
- 문제 코드에는 previous 를 s[0] 로 재설정하기 때문에 현재 문자와 이전 문자를 비교하는 것이 아니라, s[1]부터 마지막 문자까지 계속 s 의 첫 번째 문자와 비교하므로 원하는 결과를 얻을 수 없음.

③. s 에 있는 마지막 문자와 그 빈도수를 answer 에 추가.

3. 문제 3

1) 문제 코드

```python
def solution(one_day_price, multi_day, multi_day_price, n):
    if one_day_price * multi_day <= multi_day_price:
        return n * one_day_price
    else:
        return (n % multi_day) * multi_day_price + (n // multi_day) * one_day_price

# 아래는 테스트케이스 출력을 해보기 위한 코드입니다. 아래에는 잘못된 부분이 없으니, 위의 코드만 수정하세.
one_day_price1 = 3
multi_day1 = 5
multi_day_price1 = 14
n1 = 6
ret1 = solution(one_day_price1, multi_day1, multi_day_price1, n1)

# [실행] 버튼을 누르면 출력 값을 볼 수 있습니다.
print("solution 함수의 반환 값은", ret1, "입니다.")

one_day_price2 = 2
multi_day2 = 3
multi_day_price2 = 5
n2 = 11
ret2 = solution(one_day_price2, multi_day2, multi_day_price2, n2)

# [실행] 버튼을 누르면 출력 값을 볼 수 있습니다.
print("solution 함수의 반환 값은", ret2, "입니다.")
```

2) 문제 개요

- 제시된 과제가 바르게 수행되도록 문제 코드를 수정하는 문제.
- one_day 티켓은 구매한 날 하루 동안 이용할 수 있는 티켓으로 가격은 one_day_price 에 저장.
- multi_day 티켓은 구매한 날로부터 multi_day 동안 이용할 수 있는 티켓으로 가격은 multi_day_price 에 저장.
- n 일 동안 이용하는데 필요한 **최소 비용**을 구하기 위해 몫과 나머지를 이용하는 코드에서 한 줄 수정하는 문제.

3) 정답

```
def solution(one_day_price, multi_day, multi_day_price, n):
    if one_day_price * multi_day <= multi_day_price:
        return n * one_day_price
    else:
        return (n // multi_day) * multi_day_price + (n % multi_day) * one_day_price

# 아래는 테스트케이스 출력을 해보기 위한 코드입니다. 아래에는 잘못된 부분이 없으니, 위의 코드만 수정하
one_day_price1 = 3
multi_day1 = 5
multi_day_price1 = 14
n1 = 6
ret1 = solution(one_day_price1, multi_day1, multi_day_price1, n1)

# [실행] 버튼을 누르면 출력 값을 볼 수 있습니다.
print("solution 함수의 반환 값은", ret1, "입니다.")

one_day_price2 = 2
multi_day2 = 3
multi_day_price2 = 5
n2 = 11
ret2 = solution(one_day_price2, multi_day2, multi_day_price2, n2)

# [실행] 버튼을 누르면 출력 값을 볼 수 있습니다.
print("solution 함수의 반환 값은", ret2, "입니다.")
```

① (일일 이용권 금액 x multi_day)의 계산 결과가 multi_day 이용권 금액보다 작거나 같으면 (n x 일일 이용권 금액)을 계산하여 return.

② 그렇지 않다면 multi_day 에 대해서는 multi_day 이용권을 사용하는 것이 저렴함.

- (이용일 수가 저장된 n 을 multi_day 로 나눈 몫 x multi_day 이용권 금액) + (이용일 수가 저장된 n 을 multi_day 로 나눈 나머지 x 일일 이용권 금액) 을 계산하여 return.
- 문제에서 제시된 코드는 계산식을 잘못 구성하였음.

4. 문제 4

1) 문제 코드

```python
def func_a(matrix):
    n = 4
    ret = []
    exist = [False for _ in range(n*n + 1)]
    for i in range(0, n):
        for j in range(0, n):
                exist[matrix[i][j]] = True
    for i in range(1, n*n+1):
        if exist[i] == False:
            ret.append(i)
    return ret

def func_b(matrix):
    n = 4
    ret = []
    for i in range(0, n):
        for j in range(0, n):
            if matrix[i][j] == 0:
                ret.append([i, j])
    return ret

def func_c(matrix):
    n = 4
    goal_sum = sum(range(1, n*n+1))//n
    for i in range(0, n):
        row_sum = 0
        col_sum = 0
        for j in range(0, n):
            row_sum += matrix[i][j]
            col_sum += matrix[j][i]
        if row_sum != goal_sum or col_sum != goal_sum:
            return False
    main_diagonal_sum = 0
    skew_diagonal_sum = 0
    for i in range(0, n):
        main_diagonal_sum += matrix[i][i]
        skew_diagonal_sum += matrix[i][n-1-i]
    if main_diagonal_sum != goal_sum or skew_diagonal_sum != goal_sum:
        return False
    return True
```

```
41
42  def solution(matrix):
43      answer = []
44      coords = func_@@@(@@@)
45      nums = func_@@@(@@@)
46      matrix[coords[0][0]][coords[0][1]] = nums[0]
47      matrix[coords[1][0]][coords[1][1]] = nums[1]
48      if func_@@@(@@@):
49          for i in range(0, 2):
50              answer.append(coords[i][0] + 1)
51              answer.append(coords[i][1] + 1)
52              answer.append(nums[i])
53      else:
54          matrix[coords[0][0]][coords[0][1]] = nums[1]
55          matrix[coords[1][0]][coords[1][1]] = nums[0]
56          for i in range(0, 2):
57              answer.append(coords[1-i][0] + 1)
58              answer.append(coords[1-i][1] + 1)
59              answer.append(nums[i])
60      return answer
61
62  #아래는 테스트케이스 출력을 해보기 위한 코드입니다.
63  matrix = [[16,2,3,13],[5,11,10,0],[9,7,6,12],[0,14,15,1]]
64  ret = solution(matrix)
65
66  #[실행] 버튼을 누르면 출력 값을 볼 수 있습니다.
67  print("solution 함수의 반환 값은 ", ret, " 입니다.")
```

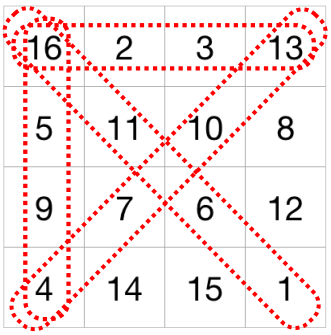
2) 문제 개요

- 문제 코드 안에 작성된 함수를 파악한 후 제시된 과제를 해결하기 위한 알고리즘대로 알맞은 함수를 호출하도록 코드를 완성하는 문제.
- 4 x 4 행렬의 빈 곳을 채우고 채워진 4 x 4 행렬 마방진이 맞는지 확인하여 그 결과를 return 하는 프로그램에서 각 함수의 기능을 확인하여 코드가 바르게 실행하도록 func_a, func_b, func_c 와 매개변수를 채우는 문제.

3) 마방진(Magic Square)

① 마방진 소개

- 가로, 세로, 대각선 방향의 수를 더한 합이 모두 같은 정사각형 행렬.
- 1 부터 정사각형 넓이까지 모든 수가 중복되지 않고 한 번씩 사용되어야 함.
- 오른쪽 그림은 가로, 세로, 대각선 수의 합이 34 인 4x4 마방진

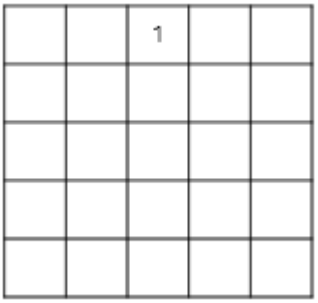


② 참고 : 홀수 N x N 마방진 만들기

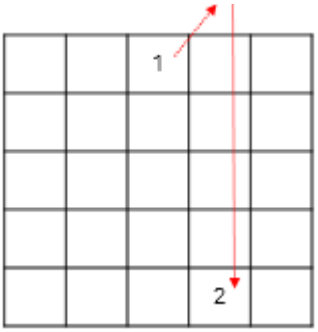
❖ 알고리즘

- 첫 번째 수 1 을 첫 번째 행 가운데에 할당.

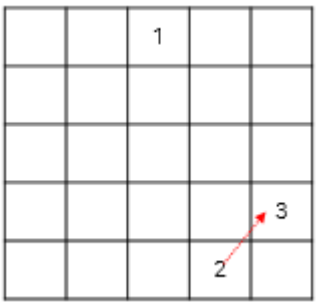
ex) N = 5 인 경우



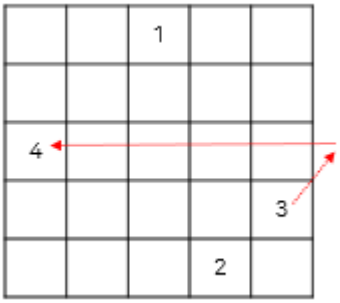
- 이전에 할당한 수가 N 의 배수가 아니면 오른쪽 대각선 위 방향 칸에 새로운 수를 할당. 단, 새로운 위치의 행이 0 보다 작으면 행의 위치를 마지막 행으로 변경.



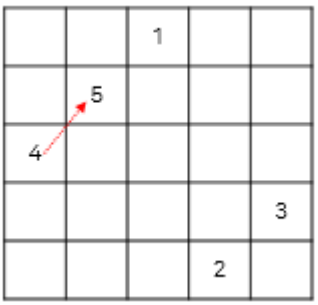
- 이전에 할당한 수가 N 의 배수가 아니면 오른쪽 대각선 위 방향 칸에 새로운 수를 할당.



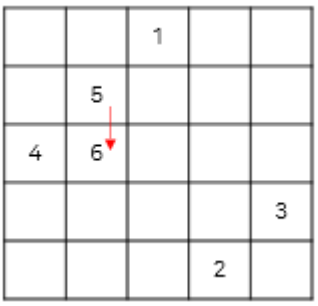
- 이전에 할당한 수가 N 의 배수가 아니면 오른쪽 대각선 위 방향 칸에 새로운 수를 할당. 단, 새로운 위치의 열이 마지막 열을 벗어나면 열의 위치를 첫 번째 열로 변경.



- 이전에 할당한 수가 N 의 배수가 아니면 오른쪽 대각선 위 방향 칸에 새로운 수를 할당.

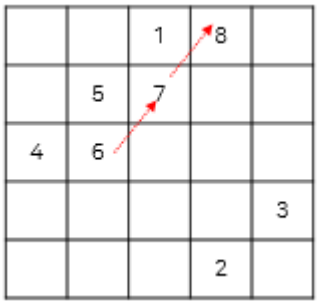


- 이전에 할당한 수가 N 의 배수이면 새로 할당할 수의 위치는 이전 행의 바로 아래 행으로 지정.

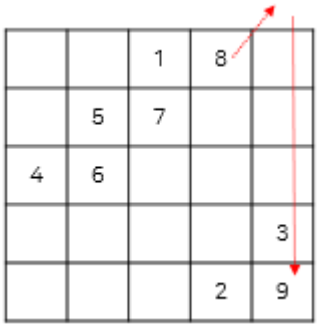


ex) N=5 일 때 이전에 할당한 수 5 가 5 의 배수이므로 5 의 바로 아래 행에 6 을 할당.

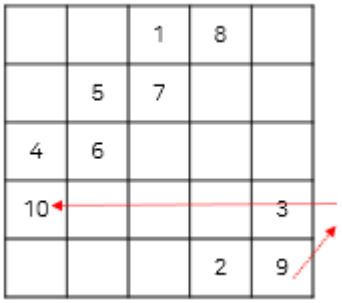
- 이전에 할당한 수가 N 의 배수가 아니면 오른쪽 대각선 위 방향 칸에 새로운 수를 할당.



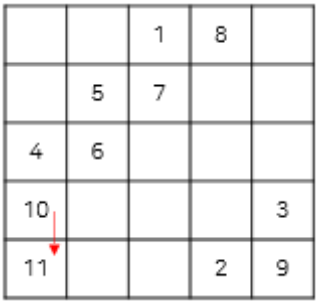
- 이전에 할당한 수가 N 의 배수가 아니면 오른쪽 대각선 위 방향 칸에 새로운 수를 할당. 단, 새로운 위치의 행이 0 보다 작으면 행의 위치를 마지막 행으로 변경.



- 이전에 할당한 수가 N 의 배수가 아니면 오른쪽 대각선 위 방향 칸에 새로운 수를 할당. 단, 새로운 위치의 열이 마지막 열을 벗어나면 열의 위치를 첫 번째 열로 변경



- 이전에 할당한 수가 N 의 배수이면 새로 할당할 수의 위치는 이전 행의 바로 아래 행으로 지정.



ex) N=5 인 경우, 이전 수 10 이 5 의 배수이므로 10 의 바로 아래 행에 11 을 할당.

- 위의 과정을 빈 칸을 다 채울 때까지 반복하여 N = 5 인 5 x 5 마방진을 오른쪽 그림과 같이 완성.

17	24	1	8	15
23	5	7	14	16
4	6	13	20	22
10	12	19	21	3
11	18	25	2	9

❖ 파이썬 코드로 구현

```
1
2   N = int(input("행렬의 크기를 홀수로 입력>"))
3 ① matrix = [[0 for _ in range(N)] for _ in range(N)]
4
5 ② row = 0
6   column = N//2
7 ③ for num in range(1, N*N+1):
8 ④     matrix[row][column] = num
9
10 ⑤    if num%N == 0:
11          row += 1
12      else:
13 ⑥        row -= 1
14          column += 1
15          if row < 0:
16 ⑦            row = N - 1
17
18          if column >= N:
19 ⑧            column = 0
20  #한 행씩 출력
21  for m in matrix:
22      print(m)
```

① 0 으로 초기화한 N x N 리스트를 생성.

② 1 을 할당할 초기 좌표 지정.

③ for 문을 이용하여 1 부터 N * N 까지 수를 num 으로 받아옴.

④ 지정된 좌표에 num 할당.

⑤ num 이 N 의 배수이면 다음 순번 수를 할당할 위치 좌표는 (현재 좌표의 행 + 1, 현재 좌표의 열)로 지정..

⑥ num 이 N 의 배수가 아니면 다음 순번 수를 할당할 위치 좌표는 (현재 좌표의 행 - 1, 현재 좌표의 열 + 1) 로 지정.

⑦ 새로 지정한 위치 좌표의 행이 첫 번째 행보다 작으면 마지막 행으로 변경.

⑧ 새로 지정한 위치 좌표의 열이 마지막 열을 넘어가면 첫 번째 열로 변경.

4) 정답

- func_a( ) ~ func_c( ) 코드 정리

```python
def func_a(matrix):
    n = 4
    ret = []
①   exist = [False for _ in range(n*n + 1)]
    for i in range(0, n):
        for j in range(0, n):
②               exist[matrix[i][j]] = True
    for i in range(1, n*n+1):
③       if exist[i] == False:
            ret.append(i)
    return ret

def func_b(matrix):
    n = 4
    ret = []
    for i in range(0, n):
        for j in range(0, n):
④           if matrix[i][j] == 0:
                ret.append([i, j])
    return ret

def func_c(matrix):
    n = 4
⑤   goal_sum = sum(range(1, n*n+1))//n
    for i in range(0, n):
        row_sum = 0
        col_sum = 0
        for j in range(0, n):
            row_sum += matrix[i][j]
⑥           col_sum += matrix[j][i]
        if row_sum != goal_sum or col_sum != goal_sum:
            return False

    main_diagonal_sum = 0
    skew_diagonal_sum = 0
    for i in range(0, n):
        main_diagonal_sum += matrix[i][i]
⑦       skew_diagonal_sum += matrix[i][n-1-i]
    if main_diagonal_sum != goal_sum or skew_diagonal_sum != goal_sum:
        return False
    return True

```

- ❖ func_a( ) 함수는 4 x 4 형태의 2 차원 리스트로 전달 받은 매개변수 matrix 에서 1 부터 16 까지의 수 중 존재하지 않는 수를 찾아 return.
  - ①. False 를 17 개 갖는 exist 리스트를 생성.
  - ②. exist 의 인덱스가 matrix 리스트의 항목값과 같은 항목의 값을 True 로 변경.
  - ③. exist 의 항목값이 False 로 남아 있는 인덱스를 모두 찾아 ret 에 항목으로 추가한 후 return.
- ❖ func_b( ) 함수는 4 x 4 형태의 2 차원 리스트로 전달 받은 매개변수 matrix 에서 항목값이 0 인 위치의 인덱스를 찾아 return.
  - ④. 중첩 for 문을 이용하여 matrix 의 항목값이 0 인 것을 찾아 해당 인덱스를 2 차원 리스트 형식으로 결과를 저장하는 리스트 ret 에 추가.
- ❖ func_c( ) 함수는 4 x 4 배열로 전달 받은 매개변수 matrix 에서 같은 행에 있는 항목들의 합계, 같은 열에 있는 항목들의 합계, 대각선 방향에 있는 항목들의 합계가 동일한 지 확인하여 그 결과를 return.

⑤. goal_sum 변수는 1 부터 16 까지의 합을 4 로 나눈 몫으로 초기화.

⑥. 중첩 for 문을 이용하여 matrix 의 각 행에 있는 항목값의 합계와 각 열에 있는 항목값의 합계를 구한 후 그 합계값이 goal_sum 변수의 값과 같지 않지 않으면 False 를 return.

⑦. 중첩 for 문을 이용하여 matrix 의 2 개의 대각선 방향에 있는 항목값의 합계를 구한 후 그 합계 값이 goal_sum 변수의 값과 같지 않으면 False 를 return,

- solution( ) 정리

```
43  def solution(matrix):
44      answer = []
45  ①   coords = func_b(matrix)
46  ②   nums = func_a(matrix)
47
48  ③   matrix[coords[0][0]][coords[0][1]] = nums[0]
49      matrix[coords[1][0]][coords[1][1]] = nums[1]
50  ④   if func_c(matrix):
51          for i in range(0, 2):
52              answer.append(coords[i][0] + 1)
53              answer.append(coords[i][1] + 1)
54              answer.append(nums[i])
55      else:
56          matrix[coords[0][0]][coords[0][1]] = nums[1]
57  ⑤       matrix[coords[1][0]][coords[1][1]] = nums[0]
58          for i in range(0, 2):
59              answer.append(coords[1-i][0] + 1)
60              answer.append(coords[1-i][1] + 1)
61              answer.append(nums[i])
62      return answer
63
64  #아래는 테스트케이스 출력을 해보기 위한 코드입니다.
65  matrix = [[16,2,3,13],[5,11,10,0],[9,7,6,12],[0,14,15,1]]
66  ret = solution(matrix)
67
68  #[실행] 버튼을 누르면 출력 값을 볼 수 있습니다.
69  print("solution 함수의 반환 값은 ", ret, " 입니다.")
```

①. func_b( ) 를 이용하여 matrix 리스트에서 항목값이 0 인 인덱스를 찾아 coords 에 2 차원 형태의 리스트로 저장(coords 저장 예 : [[1, 2], [3, 4]])

②. func_a( ) 를 이용하여 matrix 에서 1 부터 16 까지의 수 중에서 존재하지 않는 두 개의 수를 찾아 nums 에 저장.

③. nums[0], nums[1] 값을 coords[0] 번째 항목이 가리키는 matrix 의 인덱스와 coords[1] 번째 항목이 가리키는 matrix 인덱스에 차례대로 할당.

④. func_c( )를 이용하여 matrix 가 마방진이 되는 지 if 문을 통해 확인.

- 현 상태에서 matrix 가 마방진이라면 for 문을 이용하여 coords[0] 이 갖고 있는 (행 인덱스+1) 값과 (열 인덱스 +1)값을 answer 에 추가한 뒤 num[0] 을 추가.
- for 문에 의한 그 다음 반복에서는 coords[1] 이 갖고 있는 (행 인덱스+1)값과 (열 인덱스+1) 값을 answer 에 추가한 뒤 num[1] 를 추가.

⑤. if 문에 의한 조건을 만족하지 못하면 nums[1], nums[0] 값을 coords[0] 번째 항목이 가리키는 matrix 인덱스와 coords[1] 번째 항목이 가리키는 matrix 인덱스에 차례대로 할당.

- for 문을 이용하여 coords[1] 이 갖고 있는 (행 번호+1)값과 (열 번호+1)값을 answer 에 추가한 뒤 num[0] 을 answer 에 추가.
- for 문에 의한 그 다음 반복에서는 coords[0] 이 갖고 있는 (행 번호+1)값과 (열 번호+1)값을 answer 에 추가한 뒤 num[1] 을 answer 에 추가.

5. 문제 5

1) 문제 코드

```python
def solution(n):
    answer = ''
    for i in range(n):
        answer += str(@@@)
        answer = answer[@@@]
    return answer

#아래는 테스트케이스 출력을 해보기 위한 코드입니다.
n = 5
ret = solution(n)

#[실행] 버튼을 누르면 출력 값을 볼 수 있습니다.
print("solution 함수의 반환 값은", ret, "입니다.")
```

2) 문제 개요

- 제시된 과제를 수행하기 위해 코드의 빈 곳을 완성하는 문제.
- 1 부터 9 까지의 숫자를 차례대로 문자열 뒤에 붙인 후 문자열 전체의 앞뒤를 뒤집어 나타내는 작업을 n 번째까지 하여 그 결과를 출력하는 문제.

3) 예시 설명 : n = 5 인 경우

첫 번째 수	1	
두 번째 수	21	1 뒤에 2 를 붙인 후 순서를 뒤집음.
세 번째 수	312	1 뒤에 3 을 붙인 후 순서를 뒤집음.
네 번째 수	4213	2 뒤에 4 를 붙인 후 순서를 뒤집음.
다섯 번째 수	53124	3 뒤에 5 를 붙인 후 순서를 뒤집음.

4) 정답

```
def solution(n):
    answer = ''
    for i in range(n):
①       answer += str(i%9+1)
②       answer = answer[-1::-1]
    return answer

#아래는 테스트케이스 출력을 해보기 위한 코드입니다.
n = 5
ret = solution(n)

#[실행] 버튼을 누르면 출력 값을 볼 수 있습니다.
print("solution 함수의 반환 값은", ret, "입니다.")
```

①. n 번 반복하는 동안 1 부터 9 까지의 수를 반복해서 붙이기 위해 i 를 9 로 나눈 나머지에 1 을 더한 결과를 문자열로 변환하여 answer 가 갖고 있는 문자열의 끝에 붙임.

- i 를 9 로 나눈 나머지는 0 부터 8 까지 나올 수 있는데, 이렇게 계산된 나머지에 1 을 더하면 1 부터 9 까지의 결과를 산출할 수 있음.

②. 새로 생성한 answer 를 문자열의 마지막을 가리키는 인덱스 -1 부터 문자열의 처음까지 -1 만큼 증가시켜 문자열을 슬라이싱 하면 answer 가 갖고 있는 문자열의 앞뒤 순서를 뒤집을 수 있음.

6. 문제 6

7) 문제 코드

```python
def power(base, exponent):
    val = 1
    for i in range(exponent):
        val *= base
    return val

def solution(k):
    answer = []
    bound = power(10, k)
    for i in range(bound // 10, bound):
        current = i
        calculated = 0
        while current != 0:
            @@@
            @@@
        if calculated == i:
            answer.append(i)
    return answer

# 아래는 테스트케이스 출력을 해보기 위한 코드입니다.
k = 3
ret = solution(k)

# [실행] 버튼을 누르면 출력 값을 볼 수 있습니다.
print("solution 함수의 반환 값은", ret, "입니다.")
```

8) 문제 개요

- 제시된 과제를 바르게 수행하기 위하여 비어 있는 코드를 완성하는 문제.
- k 자리 수가 주어졌을 때, 그 수의 각 자릿수를 k 제곱한 합이 원래 수와 같으면 그 수를 자아도취 수라 정의 (예시 : $153 = 1^3 + 5^3 + 3^3 = 1 + 125 + 27 = 153$)
- k 자리 자아도취 수를 모두 찾아 리스트에 저장하는 프로그램에서 빈 곳을 채우는 문제.

9) 정답

```
1   def power(base, exponent):
2       val = 1
3   ①   for i in range(exponent):
4           val *= base
5       return val
6
7   def solution(k):
8       answer = []
9   ②   bound = power(10, k)
10  ③   for i in range(bound // 10, bound):
11          current = i
12  ④       calculated = 0
13          while current != 0:
14  ⑤           calculated += (current%10) ** k
15  ⑥           current = current // 10
16  ⑦       if calculated == i:
17              answer.append(i)
18      return answer
19
20  # 아래는 테스트케이스 출력을 해보기 위한 코드입니다.
21  k = 3
22  ret = solution(k)
23
24  # [실행] 버튼을 누르면 출력 값을 볼 수 있습니다.
25  print("solution 함수의 반환 값은", ret, "입니다.")
```

① for 문을 이용하여 매개변수 base 를 exponent 만큼 거듭제곱한 결과를 val 에 저장

- for 문을 사용하지 않고 base ** exponent 를 사용해도 동일.

② 위에서 정의한 power( ) 함수를 사용하여 10 을 k 제곱한 결과를 bound 에 저장.

③ for 문을 이용하여 (bound // 10) 부터 (bound - 1)를 i 로 하나씩 받아와서 current 변수를 i 로 초기화.

- k=3 이라면 10^3 // 10 = 100 부터 10^3 – 1 = 999 까지의 수 즉 세 자리 수를 i 로 전달.

④ i 로 받아온 수의 각 자리 숫자에 대해 k 제곱한 값을 합산하기 위해 사용하는 변수 calculated 를 0 으로 초기화.

⑤ current 를 10 으로 나눈 나머지를 이용하여 일의 자리부터 순서대로 current 의 각 자리 숫자를 가져와 k 제곱한 것을 calculated 에 합산.

⑥ current 를 10 으로 나눈 몫을 current 에 다시 저장하여 앞에서 k 제곱에 사용한 자리 숫자를 current 에서 삭제.

⑦ for 문을 이용해 받아온 k 자리 수 i 와 i 의 각 자리 숫자를 k 제곱한 것을 더한 값이 같으면 answer 리스트에 추가.

.

7. 문제 7

4) 문제 코드

```python
class Unit:
    def __init__(self):
        self.HP = 1000
    def under_attack(self, damage):
        pass

class Monster@@@:
    def __init__(self, attack_point):
        super().__init__()
        self.attack_point = attack_point
    @@@:
        self.HP -= damage
    @@@:
        return self.attack_point

class Warrior@@@:
    def __init__(self, attack_point):
        super().__init__()
        self.attack_point = attack_point
    @@@:
        self.HP -= damage
    @@@:
        return self.attack_point

class Healer@@@:
    def __init__(self, healing_point):
        super().__init__()
        self.healing_point = healing_point
    @@@:
        self.HP -= damage
    @@@:
        unit.HP += self.healing_point

def solution(monster_attack_point, warrior_attack_point, healing_point):
    monster = Monster(monster_attack_point)
    warrior = Warrior(warrior_attack_point)
    healer = Healer(healing_point)

    # 전사가 몬스터를 한 번 공격
    monster.under_attack(warrior.attack())
    # 몬스터가 전사를 한 번 공격
    warrior.under_attack(monster.attack())
    # 몬스터가 힐러를 한 번 공격
    healer.under_attack(monster.attack())
    # 힐러가 전사의 체력을 한 번 회복
    healer.healing(warrior)
    # 힐러가 몬스터의 체력을 한 번 회복
    healer.healing(monster)
```

```
49
50      answer = [monster.HP, warrior.HP, healer.HP]
51      return answer
52
53  # 아래는 테스트케이스 출력을 해보기 위한 코드입니다.
54  monster_attack_point = 100
55  warrior_attack_point = 90
56  healing_point = 30
57  ret = solution(monster_attack_point, warrior_attack_point, healing_point)
58
59  # [실행] 버튼을 누르면 출력 값을 볼 수 있습니다.
60  print("solution 함수의 반환 값은", ret, "입니다.")
```

5) 문제 개요

- Unit 클래스를 상속받는 Monster, Warrior, Healer 클래스를 정의하는 문제.

6) 정답

```
1   class Unit:
2       def __init__(self):
3           self.HP = 1000
4       def under_attack(self, damage):
5           pass
6
7   ① class Monster(Unit):
8       def __init__(self, attack_point):
9           super().__init__()
10          self.attack_point = attack_point
11      def under_attack(self, damage):
12          self.HP -= damage
13      def attack(self):
14          return self.attack_point
15
16  ② class Warrior(Unit):
17      def __init__(self, attack_point):
18          super().__init__()
19          self.attack_point = attack_point
20      def under_attack(self, damage):
21          self.HP -= damage
22      def attack(self):
23          return self.attack_point
24
25  ③ class Healer(Unit):
26      def __init__(self, healing_point):
27          super().__init__()
28          self.healing_point = healing_point
29      def under_attack(self, damage):
30          self.HP -= damage
31      def healing(self, unit):
32          unit.HP += self.healing_point
33
34  def solution(monster_attack_point, warrior_attack_point, healing_point):
35      monster = Monster(monster_attack_point)
36      warrior = Warrior(warrior_attack_point)
37      healer = Healer(healing_point)
```

```
38
39      # 전사가 몬스터를 한 번 공격
40      monster.under_attack(warrior.attack())
41      # 몬스터가 전사를 한 번 공격
42      warrior.under_attack(monster.attack())
43      # 몬스터가 힐러를 한 번 공격
44      healer.under_attack(monster.attack())
45      # 힐러가 전사의 체력을 한 번 회복
46      healer.healing(warrior)
47      # 힐러가 몬스터의 체력을 한 번 회복
48      healer.healing(monster)
49
50      answer = [monster.HP, warrior.HP, healer.HP]
51      return answer
52
53  # 아래는 테스트케이스 출력을 해보기 위한 코드입니다.
54  monster_attack_point = 100
55  warrior_attack_point = 90
56  healing_point = 30
57  ret = solution(monster_attack_point, warrior_attack_point, healing_point)
58
59  # [실행] 버튼을 누르면 출력 값을 볼 수 있습니다.
60  print("solution 함수의 반환 값은", ret, "입니다.")
```

① Unit 클래스를 상속받은 Monster 클래스를 정의
- 생성자 메소드 __init__( ) 에서 부모 클래스의 생성자를 실행하고, 멤버변수 attack_point 를 매개변수로 전달받는 값으로 초기화.
- Unit 클래스에 있는 under_attack( ) 메소드를 오버라이드 : damage 만큼 멤버 변수 HP 를 감소.
- attack( ) 메소드를 정의 : 멤버 변수인 attack_point 를 return.

② Unit 클래스를 상속받은 Warrior 클래스를 정의
- 생성자 메소드 __init__( ) 에서 부모 클래스의 생성자를 실행하고, 멤버변수 attack_point 를 매개변수로 전달받는 값으로 초기화.
- Unit 클래스에 있는 under_attack( ) 메소드를 오버라이드 : damage 만큼 멤버 변수 HP 를 감소.
- attack( ) 메소드를 정의 : 멤버 변수인 attack_point 를 return.

③ Unit 클래스를 상속받은 Healer 클래스를 정의
- Unit 클래스에 있는 under_attack( ) 메소드를 오버라이드 : damage 만큼 멤버 변수 HP 를 감소.
- 매개변수로 unit 를 갖는 healing( ) 메소드를 정의 : 매개변수로 받은 객체 unit 의 멤버 변수 HP 를 healing_point 만큼 증가.

8. 문제 8

4) 문제 코드

```python
def solution(card, n):
    # 여기에 코드를 작성해주세요.
    answer = 0
    return answer

# 아래는 테스트케이스 출력을 해보기 위한 코드입니다.
card1 = [1, 2, 1, 3]
n1 = 1312
ret1 = solution(card1, n1)

# [실행] 버튼을 누르면 출력 값을 볼 수 있습니다.
print("solution 함수의 반환 값은 ", ret1, " 입니다.")

# 아래는 테스트케이스 출력을 해보기 위한 코드입니다.
card2 = [1, 1, 1, 2]
n2 = 1122
ret2 = solution(card2, n2)

# [실행] 버튼을 누르면 출력 값을 볼 수 있습니다.
print("solution 함수의 반환 값은 ", ret2, " 입니다.")
```

5) 문제 개요

- 제시된 과제를 해결하기 위해 solution( )에 프로그램 코드를 작성하는 문제.
- 전달된 숫자 카드를 모두 사용하여 만들 수 있는 수를 리스트로 저장한 후 solution( ) 함수의 매개변수 n 에 있는 수가 리스트에 있는 수 중에서 몇 번째로 작은 수인지 return 하는 solution 함수를 작성.

6) 정답

```python
def func_a(card):
①    card_count = [0] * 10
②    for card_i in card:
③        card_count[card_i] += 1
    return card_count

num_list = []
def func_b(level, max_level, num, current_count, max_count):
    if level == max_level:
④        num_list.append(num)
        return

⑤    for i in range(1, 10):
⑥        if current_count[i] < max_count[i]:
⑦            current_count[i] += 1
⑧            func_b(level + 1, max_level, num * 10 + i, current_count, max_count)
⑨            current_count[i] -= 1

def func_c(list, n):
    if n in list:
⑩        return list.index(n) + 1
⑪    return -1

def solution(card, n):
⑫    num_list.clear()
⑬    card_count = func_a(card)
⑭    func_b(0, len(card), 0, [0] * 10, card_count)
⑮    answer = func_c(num_list, n)
    return answer

# 아래는 테스트케이스 출력을 해보기 위한 코드입니다.
card1 = [1, 2, 1, 3]
n1 = 1312
ret1 = solution(card1, n1)

# [실행] 버튼을 누르면 출력 값을 볼 수 있습니다.
print("solution 함수의 반환 값은 ", ret1, " 입니다.")

# 아래는 테스트케이스 출력을 해보기 위한 코드입니다.
card2 = [1, 1, 1, 2]
n2 = 1122
ret2 = solution(card2, n2)

# [실행] 버튼을 누르면 출력 값을 볼 수 있습니다.
print("solution 함수의 반환 값은 ", ret2, " 입니다.")
```

❖ func_a( ) 는 card 리스트에 있는 숫자를 읽어서 각 숫자의 개수를 card_count 에 집계하여 리턴하는 함수

③. 10 개의 0 을 항목으로 갖는 card_count 리스트 생성.

④. for 문을 이용하여 card 에 있는 숫자를 하나씩 card_i 로 받음.

⑤. card_i 와 같은 값을 인덱스로 갖는 card_count 리스트의 항목을 1 만큼 증가.

ex) card = [1, 2, 1, 3] 인 경우 card_count 에 집계되는 값

인덱스	[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
항목	0	2	1	1	0	0	0	0	0	0

➔ card 에 1 이 2 개 존재하므로 card_count 의 1 번 인덱스의 항목은 2.

2 가 1 개 존재하므로 card_count 의 2 번 인덱스의 항목은 1.
3 이 1 개 존재하므로 card_count 의 3 번 인덱스의 항목은 1.

❖ func_b( ) 는 max_count 에 집계되어 있는 숫자 카드별 총 개수들을 바탕으로 하여 숫자 카드 모두를 사용한 수를 생성한 후 전역변수 num_list 에 생성한 수를 추가.

매개변수 level	현재 생성한 수 num 을 만들기 위해 사용한 총 숫자 카드 개수.
매개변수 max_level	숫자 카드 총 개수.
매개변수 num	현재 생성한 수.
매개변수 current_count	num 을 만들기 위해 각 숫자 카드별 사용한 개수.
매개변수 max_count	각 숫자 카드별 총 개수.

⑥. 현재 생성한 num 을 만들기 위해 사용한 총 카드 개수가 숫자 카드 총 개수와 동일하면 전역 변수로 선언된 num_list 에 num 을 추가.
⑦. for 문을 이용하여 숫자 카드 1 부터 숫자 카드 9 까지의 숫자를 i 로 받음.
⑧. 현재 생성한 수(num)를 만들기 위해 숫자 카드 i 를 사용한 개수(current_count[i])가 숫자 카드 i 의 총 개수 보다 작은 지 확인.
⑨. 숫자 카드 i 를 사용한 개수를 1 만큼 증가. :
⑩. 수를 만들기 위해 재귀 함수 호출 : 숫자 카드 i 를 추가하여 새로운 수를 생성해서 호출함.
- 첫 번째 인수 : 수를 만들기 위해 사용한 전체 카드 개수 (level) + 1
- 두 번째 인수 : 숫자 카드 총 개수 (max_level)
- 세 번째 인수 : 새로 생성한 수 = 현재 생성한 수 (num) * 10 + i
- 네 번째 인수 : 새로 생성한 수를 만들기 위해 각 숫자 카드별 사용한 개수 (current_count)
- 다섯 번째 인수 : 각 숫자 카드별 총 개수 (max_count)

⑪. 숫자 카드 i 를 사용하여 재귀 호출한 함수가 종료되면 숫자 카드 i 를 사용한 개수(current_count[i])를 다시 1 만큼 감소.

❖ func_c( ) 는 list 에서 n 을 항목값으로 갖는 인덱스를 찾아 리턴하는 함수. .
⑫. n 이 list 의 항목값으로 존재하면 index( ) 메소드를 사용하여 list 에서 항목값이 n 인 인덱스를 찾은 후 그 인덱스에 + 1 을 하여 return. (순서는 첫 번째부터 시작하지만 인덱스는 0 부터 시작하기 때문)
⑬. n 이 list 에 존재하지 않으면 -1 을 리턴.

❖ solution( ) 은 card 와 n 을 매개변수로 받아 card 에 있는 숫자 카드를 이용해서 수를 만들고 n 이 생성한 수들 중에 몇 번째 수인지 리턴하는 함수. .
⑭. 생성한 수를 저장하는 전역 변수 num_list 를 초기화.

⑮. func_a( ) 를 이용하여 card 에 있는 각 숫자 카드별 총 개수를 card_count 에 집계.

16. func_b( ) 를 이용하여 card 에 있는 숫자 카드로 구성된 수를 만들어 num_list 에 저장.
    - 첫 번째 인수 : 0 - 처음 시작하기 때문에 사용한 숫자 카드가 없음.
    - 두 번째 인수 : len(card) – 숫자 카드 총 개수
    - 세 번째 인수 : 0 – 처음 시작하기 때문에 생성한 수가 없음.
    - 네 번째 인수 : 10 개의 0 을 갖는 리스트 – 생성한 수가 없기 때문에 숫자 카드 1 부터 9 까지 사용한 카드가 없음.
    - 다섯 번째 인수 : card_count – 각 숫자 카드별 총 개수

17. func_c( ) 를 이용하여 매개변수 n 의 값이 숫자 카드로 만든 수 중 몇 번째인지 확인.

7) 정답 코드의 func_b( ) 함수의 실행 process

: card = [1, 2, 1, 3] 를 사용하여 수를 만드는 경우

```
num_list = []
def func_b(level, max_level, num, current_count, max_count):
    if level == max_level:
        num_list.append(num)
        return

    for i in range(1, 10):
        if current_count[i] < max_count[i]:
            current_count[i] += 1
            func_b(level + 1, max_level, num * 10 + i, current_count, max_count)
            current_count[i] -= 1
```

① solution( ) 에서 func_b( ) 호출 : ,

```
func_b(0, 4, 0, current_count, max_count)
```

- 매개 변수에 저장하는 값

level = 0 (사용한 숫자 카드 개수 : 아직 생성한 수가 없기 때문에 0)

max_level = 4 (card 의 항목 개수)

num = 0

current_count

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
0	0	0	0	0	0	0	0	0	0

(생성한 수가 없기 때문에 사용한 숫자 카드가 없음)

max_count

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
0	2	1	1	0	0	0	0	0	0

(card 의 각 항목값 별 개수)

- func_b( ) 실행
    - level < max_level 이므로 for 문을 실행.
    - for 문에 의해서 1 부터 9 까지의 숫자를 i 로 받음.

- current_count[1] < max_count[1] 이므로
  - current_count[1] 을 1 만큼 증가시킨 후 func_b( )를 재귀 호출.
  - 재귀 호출이 종료되면 숫자 카드 1 을 이용한 작업이 종료되었으므로 current_count[1]을 1 만큼 감소.

② func_b( ) 첫 번째 재귀 호출 :

◆ 매개 변수에 저장하는 값

level	= 1 (num 을 만드는데 사용한 숫자 카드 개수)
max_level	= 4 (card 의 항목 개수)
num	= 0 * 10 + 1 = 1 (숫자 카드 1 한 개를 사용하여 만든 수)

current_count

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
0	1	0	0	0	0	0	0	0	0

(num 을 만드는 데 사용한 숫자 카드 1 을 1 개 사용했기 때문)

max_count

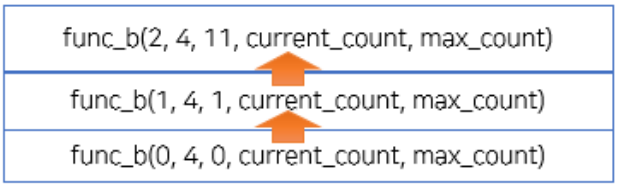
[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
0	2	1	1	0	0	0	0	0	0

(card 의 각 항목 값 별 개수)

◆ func_b( ) 실행

- level < max_level 이므로 for 문을 실행.
- for 문에 의해서 1 부터 9 까지의 숫자를 i 로 받음.
- current_count[1] < max_count[1] 이므로
  - current_count[1] 을 1 만큼 증가시킨 후 func_b( )를 재귀 호출.
  - 재귀 호출이 종료되면 숫자 카드 1 을 이용한 작업이 종료되었으므로 current_count[1]을 1 만큼 감소.

③ func_b( ) 두 번째 재귀 호출 :

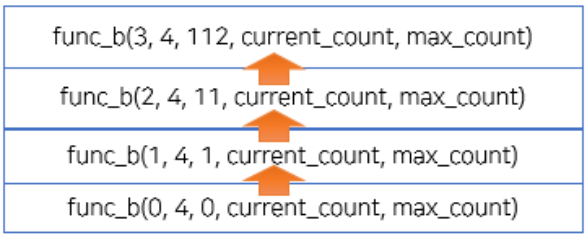


- 매개 변수에 저장되어 있는 값

level	= 2 (num 을 만드는데 사용한 숫자 카드 개수)
max_level	= 4 (card 의 항목 개수)
num	= 1 * 10 + 1 = 11 (숫자 카드 1 두 개를 사용하여 만든 수)
current_count	[0] [1] [2] [3] [4] [5] [6] [7] [8] [9] 0 2 0 0 0 0 0 0 0 0 (num 을 만드는 데 사용한 숫자 카드 1 을 2 개 사용했기 때문)
max_count	[0] [1] [2] [3] [4] [5] [6] [7] [8] [9] 0 2 1 1 0 0 0 0 0 0 (card 의 각 항목값 별 개수)

- func_b( ) 실행
  - level < max_level 이므로 for 문을 실행.
  - for 문에 의해서 1 부터 9 까지의 숫자를 i 로 받음.
  - i 가 1 일 경우는 current_count[i] == max_count[i] 이지만, current_count[2] < max_count[2] 이므로
    - current_count[2] 을 1 만큼 증가시킨 후 func_b( )를 재귀 호출.
    - 재귀호출이 종료되면 숫자 카드 2 을 이용한 작업이 종료되었으므로 current_count[2] 을 1 만큼 감소.

④ func_b( ) 세 번째 재귀 호출 :



- 매개 변수에 저장되어 있는 값

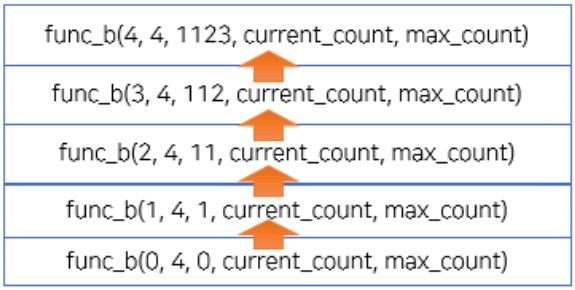
level	= 3 (num 을 만드는데 사용한 숫자 카드 개수)
max_level	= 4 (card 의 항목 개수)
num	= 11 * 10 + 2 = 112 (숫자 카드 1 두 개와 숫자 카드 2 한 개를 사용하여 만든 수)
current_count	[0] [1] [2] [3] [4] [5] [6] [7] [8] [9] 0 2 1 0 0 0 0 0 0 0 (num 을 만드는데 사용한 숫자 카드 1 을 2 개, 숫자 카드 2 를 1 개 사용했기 때문)
max_count	[0] [1] [2] [3] [4] [5] [6] [7] [8] [9]

	0	2	1	1	0	0	0	0	0	0

(card 의 각 항목값 별 개수)

- func_b( ) 실행
  - level < max_level 이므로 for 문을 실행.
  - for 문에 의해서 1 부터 9 까지의 숫자를 i 로 받음.
  - i 가 1, 2 일 경우는 current_count[i] == max_count[i] 이지만, current_count[3] < max_count[3] 이므로
    - current_count[3] 을 1 만큼 증가시킨 후 func_b( )를 재귀 호출.
    - 재귀 호출이 종료되면 숫자 카드 3 을 이용한 작업이 종료되었으므로 current_count[3]을 1 만큼 감소.

⑤ func_b( ) 네 번째 재귀 호출 :



- 매개 변수에 저장되어 있는 값

level	= 4 (num 을 만드는데 사용한 숫자 카드 개수)
max_level	= 4 (card 의 항목 개수)
num	= 112* 10 + 3 = 1123 (숫자 카드 1 두 개, 숫자 카드 2 한 개, 숫자 카드 3 한 개를 이용하여 만든 수)

current_count

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
0	2	1	1	0	0	0	0	0	0

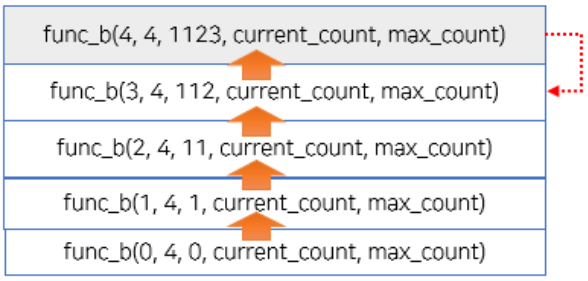
(num 을 만드는데 사용한 숫자 카드 1 을 2 개, 숫자 카드 2 를 1 개, 숫자 카드 3 을 1 개 사용했기 때문)

max_count

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
0	2	1	1	0	0	0	0	0	0

(card 의 각 항목값 별 개수)

- func_b( ) 의 실행
  - level == max_level 이므로 num 을 num_list 에 추가하고 종료하여 func_b( )를 세 번째로 재귀 호출한 곳으로 돌아감.



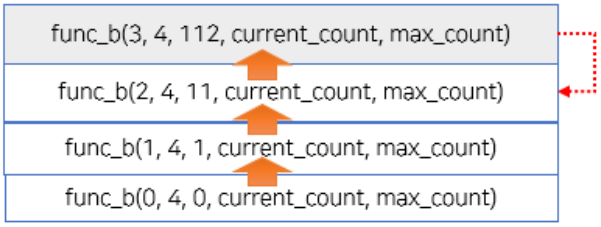
⑥ 네 번째 재귀 호출 → 세 번째 재귀 호출로 돌아간 후 func_b( ) 의 실행

- for 문에 의해 i=3 인 상태에서 실행되던 네 번째 재귀 호출을 종료한 후 current_count[3] 을 1 만큼 감소.

current_count

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
0	2	1	0	0	0	0	0	0	0

- i=4 부터 9 까지에 대해 current_count[i] == max_count[i] 이므로 나머지 반복에 대해서는 재귀 호출을 하지 않고 종료.
- 세 번째 재귀 호출 종료 후 func_b( ) 를 두 번째로 재귀 호출한 곳으로 돌아감.



⑦ 세 번째 재귀 호출 → 두 번째 재귀 호출로 돌아간 후 func_b( ) 실행

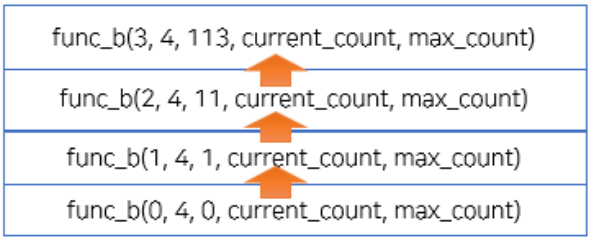
- for 문에 의해 i=2 인 상태에서 실행되던 세 번째 재귀 호출 종료 후 current_count[2] 을 1 만큼 감소.

current_count

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
0	2	0	0	0	0	0	0	0	0

- i=3 부터 9 까지 반복문을 실행하는데 current_count[3] < max_count[3] 이므로
  - current_count[3] 을 1 만큼 증가시킨 후 func_b( )를 재귀 호출.
  - 재귀 호출이 종료되면 숫자 카드 3 을 이용한 작업이 종료되었으므로 current_count[3]을 1 만큼 감소

⑧ func_b( ) 다섯 번째 재귀 호출 :



- 매개 변수에 저장되어 있는 값

level	= 3 (num 을 만드는데 사용한 숫자 카드 개수)
max_level	= 4 (card 의 항목 개수)
num	= 11 * 10 + 3 = 113 (숫자 카드 1 두 개와 숫자 카드 3 한 개를 사용하여 만든 수)
current_count	[0] [1] [2] [3] [4] [5] [6] [7] [8] [9] 0 2 0 1 0 0 0 0 0 0 (num 을 만드는데 사용한 숫자 카드 1 을 2 개, 숫자 카드 3 을 1 개 사용했기 때문)
max_count	[0] [1] [2] [3] [4] [5] [6] [7] [8] [9] 0 2 1 1 0 0 0 0 0 0 (card 의 각 항목값 별 개수)

- func_b( ) 실행
  - level < max_level 이므로 for 문을 실행.
  - for 문에 의해서 1 부터 9 까지의 숫자를 i 로 받음.
  - i 가 1 일 경우는 current_count[i] == max_count[i] 이지만, current_count[2] < max_count[2] 이므로
    - current_count[2] 을 1 만큼 증가시킨 후 func_b( )를 재귀 호출.
    - 재귀 호출이 종료되면 숫자카드 2 를 이용한 작업이 종료되었으므로 current_count[2]을 1 만큼 감소.

➔ 위와 같은 과정을 [1, 1, 2, 3] 을 이용하여 만들 수 있는 4 자리수를 모두 생성할 때까지 반복.

9. 문제 9

5) 문제 코드

```python
# 다음과 같이 import를 사용할 수 있습니다.
# import math

def solution(hour, minute):
    answer = ''
    return answer

# 아래는 테스트케이스 출력을 해보기 위한 코드입니다.
hour = 3
minute = 0
ret = solution(hour, minute)

# [실행] 버튼을 누르면 출력 값을 볼 수 있습니다.
print("solution 함수의 반환 값은 ", ret, " 입니다.")
```

6) 문제 개요

- 제시된 과제를 해결하기 위해 solution( )에 프로그램 코드를 작성하는 문제.
- 매개변수 hour와 minute 를 이용하여 1 시간 동안 30 도, 1 분 동안 0.5 도 움직이는 시침과 1 시간 동안 360 도, 1 분 동안 6 도 움직이는 분침이 이루는 각도를 소수점 첫 번째 자리까지의 문자열로 return 하도록 코드를 작성하는 문제.

7) 정답

시침	1 시간 동안 30 도 이동, 1 분 동안 30/60=0.5 도 이동.
분침	1 분 동안 360/60 = 6 도 이동.

➔ 시침이 이동하는 각도 = hour * 30 + minute * 0.5

➔ 분침이 이동하는 각도 = minute * 6

```python
def solution(hour, minute):
    answer = ''
    h_hand = hour * 30 + minute * 0.5
①   m_hand = minute * 6
    arc = abs(m_hand - h_hand)
②   if arc > 180:
        arc = 360 - arc
③   answer = '{:.1f}'.format(arc)
    return answer

# 아래는 테스트케이스 출력을 해보기 위한 코드입니다.
hour = 3
minute = 0
ret = solution(hour, minute)

# [실행] 버튼을 누르면 출력 값을 볼 수 있습니다.
print("solution 함수의 반환 값은 ", ret, " 입니다.")
```

① 매개변수 hour 와 minute 를 이용하여 시침과 분침이 이동하는 각도를 각각 계산한 후, 두 바늘이 이루는 각도를 계산.

② 계산한 각도 값이 180 보다 크면 360 에서 각도 값을 빼서 각도 값으로 다시 할당.

③. 출력 서식을 지정하는 메소드인 format( )을 사용하여 계산한 각도 arc 를 소수점 아래 한 자리를 갖는 서식을 갖는 문자열로 변환하여 answer 에 저장.

10. 문제 10

1) 문제 코드

```
#다음과 같이 import를 사용할 수 있습니다.
#import math

def solution(a, b):
    # 여기에 코드를 작성해주세요.
    answer = 0
    return answer

#아래는 테스트케이스 출력을 해보기 위한 코드입니다.
a = 6
b = 30
ret = solution(a, b)

#[실행] 버튼을 누르면 출력 값을 볼 수 있습니다.
print("solution 함수의 반환 값은", ret, "입니다.")
```

2) 문제 개요

- 제시된 과제를 해결하기 위해 solution( )에 프로그램 코드를 작성하는 문제.
- 매개변수로 전달된 a, b 사이에 있는 수들 중에서 소수의 제곱수와 소수의 세제곱수를 찾아 그 개수를 return 하도록 코드를 작성하는 문제.

3) 소수 찾기

① 소수의 정의

- 1 과 자기 자신만으로 나누어 떨어지는 수

② 소수를 판별하는 방법 1

- 어떤 수를 2 부터 시작하여 자기 자신보다 하나 작은 수까지의 수로 나누어 떨어지는 수가 없으면 소수로 판별.
- 만일 나누어 떨어지는 수가 존재하면 소수가 아님.
- 2 부터 k 까지의 수 중 소수를 찾는 파이썬 프로그램

```
def solution(k):
    arr=[]

    for n in range(2, k+1):
        for i in range(2, n):
            if n % i == 0:
                break
        else:
            arr.append(n)
    return arr
```

- for 문을 이용하여 2 부터 k 까지의 수를 n 으로 받음.
- 소수를 판별하기 위해 2 부터 n-1 까지의 수를 i 로 받음.
- n 을 i 로 나눈 나머지가 0 이면 i 는 n 의 약수이고 n 은 소수가 아니므로, break 를 사용하여 소수 판별수 i 로 나누는 작업 종료.

- break 를 실행하지 않고 안쪽 for 가 종료된 경우 n 을 arr 에 추가.

➔ 숫자 k 가 커질수록 실행 시간이 커짐.
k=100000 인 경우 28 초 정도 소요됨.

③ 에라토스테네스의 체

- 입자의 크기가 다른 가루를 체로 거르는 것처럼 에라토스테네스의 체를 사용하여 특정 자연수 이하의 소수만 골라내는 방법.
- 원하는 범위만큼 리스트를 생성하여 초기값을 T 로 지정.

	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	...
	T	T	T	T	T	T	T	T	T	T	T	T	T	T	T	T	T	...

- 항목값이 T 인 최소 인덱스 2 에 대해서 인덱스가 2 보다 큰 2 의 배수인 항목들의 값을 모두 F 로 변경(소수 2 를 찾은 후 2 의 배수는 소수가 아닌 것을 나타냄.)

	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	...
	T	T	F	T	F	T	F	T	F	T	F	T	F	T	F	T	F	...

- 항목값이 T 인 최소 인덱스 3 에 대해서 인덱스가 3 보다 큰 3 의 배수인 항목들의 값을 모두 F 로 변경(소수 3 을 찾은 후 3 의 배수는 소수가 아닌 것을 나타냄.)

	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	...
	T	F	F	T	F	T	F	F	F	T	F	T	F	F	F	T	F	...

- 항목값이 T 로 남아있는 인덱스 중 가장 작은 수가 원하는 범위 n 의 제곱근을 넘을 때까지 항목값이 T 인 최소 인덱스를 찾아서 위와 같은 과정을 반복.
(∵ n 의 최대 약수가 $\sqrt{n}$)
- 파이썬 코드로 구현

```
1   import math
2   n = 10
3   arr=[True]*(n+1)
4   arr[0]=arr[1]=False
5
6   for i in range(2, int(math.sqrt(n))+1):
7       if arr[i] == True:
8           for j in range(i+i, n+1, i):
9               arr[j] = False
10
11  for t in range(2, n+1):
12      if arr[t] == True:
13          print(t, end=" ")
```

①. for 문을 이용하여 2 부터 $\sqrt{n}$ 의 정수부까지의 수를 가져와 i 로 받음.
②. i 보다 큰 i 의 배수를 인덱스로 갖는 항목을 False 로 지정하여 소수가 아님을 표시.

➔ 중첩 반복문을 사용하기 때문에 실행 시간이 많이 소요될 것으로 예상되지만 실제로 시간 복잡도는 O(n log log n) 만큼 걸림.

4) 정답

① 브루트 포스 방식

```
1  import math
2  def solution(a, b):
3      # 여기에 코드를 작성해주세요.
4      answer = 0
5
6  ①   for n in range(2, int(math.sqrt(b))+1):
7          for i in range(2,n):
8              if n%i == 0:    # 자기 자신 이외의 약수가 존재하면
9                  break       # 소수가 아님.
10         else:
11             #소수의 제곱수가 범위 안에 들어가면 집계
12 ②           if a<= n ** 2 <= b:
13                 answer += 1
14             # 소수의 세제곱수가 범위 안에 들어가면 집계
15 ③           if a<= n ** 3 <= b :
16                 answer += 1
17     return answer
18
19 #아래는 테스트케이스 출력을 해보기 위한 코드입니다.
20 a = 6
21 b = 30
22 ret = solution(a, b)
23
24 #[실행] 버튼을 누르면 출력 값을 볼 수 있습니다.
25 print("solution 함수의 반환 값은", ret, "입니다.")
```

①. 2 부터 $\sqrt{b}$ 의 정수값까지 가져오도록 math 모듈의 sqrt( ) 함수를 사용.

②. 소수로 판명된 n 의 제곱수가 a 이상이고 b 이하이면 answer 를 1 만큼 증가.

③. 소수로 판정된 n 의 세제곱수가 a 이상이고 b 이하이면 answer 를 1 만큼 증가.

② 에라토스테네스의 체를 이용한 방식

```
import math

def solutio(a, b):
    answer = 0
①   end = int(math.sqrt(b))
②   arr = [True] * (end + 1)
    for n in range(2, end+1):
        if arr[n] == True:
③           for j in range(n+n, end+1, n):
                arr[j] = False
    for i in range(2, len(arr)+1):
④       if arr[i] == True:
⑤           if a <= i**2 <= b:
                answer += 1
⑥           if a <= i **3 <= b:
                answer += 1

#아래는 테스트케이스 출력을 해보기 위한 코드입니다.
a =  6
b =  30
ret = solution(a, b)

#[실행] 버튼을 누르면 출력 값을 볼 수 있습니다.
print("solution 함수의 반환 값은", ret, "입니다.")
```

① 검색할 끝 범위를 $\sqrt{b}$ 의 정수값까지 가져오도록 math 모듈의 sqrt( ) 함수를 사용하여 end 에 저장.

② 인덱스가 0 부터 end 까지인 리스트 arr 를 생성하여 end+1 개의 항목을 모두 True 로 지정.

③ arr 의 n 번 인덱스 항목값이 True 이면 arr 의 인덱스가 n 의 배수인 항목값을 모두 False 로 변경.

④ arr 의 i 번 인덱스 항목값이 True 이면 인덱스가 소수라는 의미

⑤ 소수로 판명된 i 의 제곱수가 a 이상이고 b 이하이면 answer 를 1 만큼 증가.

⑥ 소수로 판정된 i 의 세제곱수가 a 이상이고 b 이하이면 answer 를 1 만큼 증가.

➔ 소수를 판별해야 하는 수의 범위가 커질수록 '에라토스테네스의 체' 방식을 사용하는 것이 더 효율적.