Professional Coding Specialist

COS Pro 파이썬 1 급

25 강-26 강. 모의고사 6 차

1. 모의고사 6 차(1-10 번)

과정 소개

Cos Pro 1 급 파이썬 6 차 문제를 풀어보며 문제 유형을 익히고, 파이썬을 이용하여 알고리즘을 구현하기 위해 필요한 관련 지식을 익혀보도록 한다.

학습 목차

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
#다음과 같이 import를 사용할 수 있습니다.
#import math

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

#아래는 테스트케이스 출력을 해보기 위한 코드입니다.
n1 = 3
garden1 = [[0, 0, 0], [0, 1, 0], [0, 0, 0]]
ret1 = solution(n1, garden1)

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

n2 = 2
garden2 = [[1, 1], [1, 1]]
ret2 = solution(n2, garden2)

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

2) 문제 개요

- 제시된 과제를 해결하기 위해 solution( )에 프로그램 코드를 작성하는 문제.
- n x n 격자 모양의 정원에 개화하지 않은 꽃과 개화한 꽃을 심은 상황.
- 하루가 지나면 꽃이 피어 있던 영역의 상, 하, 좌, 우에 있는 피지 않았던 꽃도 개화함.
- 정원 크기 n 과 현재 정원에 꽃이 피어 있는 상태를 표현하는 2 차원 리스트 garden 이 주어졌을 때, 꽃이 모두 피는 데 걸리는 일자 수를 계산하는 문제.

3) 정답

- 주요 아이디어 정리 : garden = [[0, 0, 0], [0, 1, 0], [0, 0, 0]] 인 경우.

첫 날 개화 현황

0	0	0
0	1	0
0	0	0

- [1, 1] 영역의 꽃이 피어 있음.

1 일 경과 후 개화 현황

0	1	0
1	1	1
0	1	0

- [1, 1]을 기준으로 상/하/좌/우 의 위치인 [0, 1], [2, 1], [1, 0], [1, 2] 영역의 꽃이 추가로 개화.

2 일이 경과 후 개화 현황

1	1	1
1	1	1
1	1	1

- [0, 1], [2, 1], [1, 0], [1, 2] 4 개 영역을 기준으로 상/하/좌/우 의 영역에 추가로 꽃이 개화함으로써 전체 정원의 꽃이 모두 개화.

➔ 첫 날 개화한 영역을 이용하여 1 일 경과 후에 개화하는 영역을 구하고, 1 일 경과 후 새로 개화한 영역을 이용하여 2 일 경과 후에 개화하는 영역을 구함.

➔ 새로 개화한 영역만을 이용하여 꽃이 피지 않은 영역 중 다음 날 개화하는 좌표와 소요 일자를 구하기 위해서는 큐(Queue) 를 사용하는 것이 효과적.

- 정답 코드

```
1   import queue
2
3   def solution(n, garden):
4       answer = 0
5
6   ①   q = queue.Queue()
7       dx = [ -1, 1, 0, 0 ]
8   ②   dy = [ 0, 0, -1, 1 ]
9
10      for i in range(n) :
11          for j in range(n) :
12  ③           if garden[i][j] == 1 :
13                  q.put((i, j, 0))
14
15  ④   while q.empty() == False :
16  ⑤       x, y, day = q.get()
17
18  ⑥       for i in range(4) :
19              next_x = x + dx[i]
20  ⑦           next_y = y + dy[i]
21  ⑧           next_day = day + 1
22
23  ⑨           if (0 <= next_x and next_x < n and 0 <= next_y and next_y < n)
24                      and (garden[next_x][next_y] == 0) :
25  ⑩               garden[next_x][next_y] = 1
26                  answer = next_day
27  ⑪               q.put((next_x, next_y, next_day))
28
29      return answer
30
31  #아래는 테스트케이스 출력을 해보기 위한 코드입니다.
32  n1 = 3
33  garden1 = [[0, 0, 0], [0, 1, 0], [0, 0, 0]]
34  ret1 = solution(n1, garden1)
35
36  #[실행] 버튼을 누르면 출력 값을 볼 수 있습니다.
37  print("solution 함수의 반환 값은 ",ret1, "입니다.")
38
39  n2 = 2
40  garden2 = [[1, 1], [1, 1]]
41  ret2 = solution(n2, garden2)
42
43  #[실행] 버튼을 누르면 출력 값을 볼 수 있습니다.
44  print("solution 함수의 반환 값은", ret2, "입니다.")
```

①. queue 모듈의 Queue( ) 를 사용하여 FIFO 방식의 큐(Queue) 를 생성.

②. 선택한 영역의 상/하/좌/우 인덱스 지정을 위한 x 좌표 변화값을 dx 리스트로 저장하고, dx 의 각 항목에 대응하는 y 인덱스 변화값을 dy 리스트에 순서대로 저장.

③. 매개변수 garden 리스트에서 항목값이 개화한 영역을 나타내는 값인 1 인 항목을 찾아서 큐에 추가.

- 큐에 추가하는 항목 = (현 위치의 x 좌표를 나타내는 인덱스, 현 위치의 y 좌표를 나타내는 인덱스, 현재 위치에 개화하는데 소요된 일자)로 구성된 튜플(tuple).
- q 에 대한 put( ) 메소드를 이용하여 (i , j , 0) 로 된 항목을 큐의 초기 항목으로 추가
  - ◆ i : 현 위치의 x 좌표를 나타내는 인덱스 값.
  - ◆ Y : 현 위치의 y 좌표를 나타내는 인덱스 값.
  - ◆ 0 : - 시작하는 상태이므로 소요 일자가 없기 때문에 0 으로 지정.

④. while 문에서 새로 개화한 영역을 저장하는 큐 q 에 대한 empty( ) 메소드를 실행하여 q 가 비어 있지 않는 동안 반복하도록 지정.

⑤. q 에 대한 get( ) 메소드를 사용하여 q 에 저장되어 있는 첫 번째 항목을 삭제하고 그 삭제된 튜플 항목에 있는 개별 값을 각각 x, y, day 에 저장.

⑥. for 문을 이용하여 현재 개화한 영역의 상/하/좌/우 4 개의 영역을 차례대로 가져오기 위해 4 번 반복하도록 지정.

⑦. 큐에서 가져온 영역의 x 좌표에 x 좌표 변화값을 더하여 next_x 에 저장하고, y 좌표에 y 좌표 변화값을 더하여 next_y 에 저장. → next_x, next_y 는 새로 개화하는 영역의 좌표.

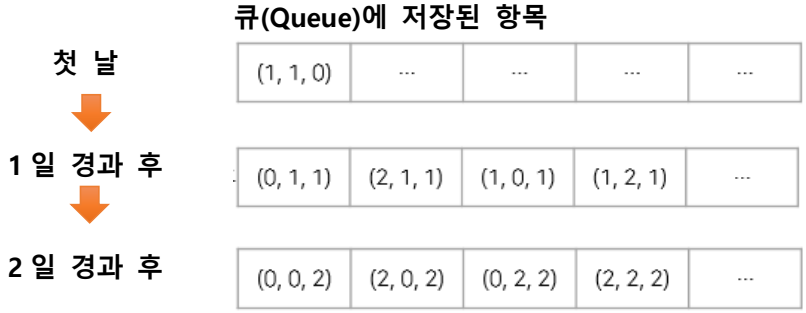
⑧. 새로 개화하는 데 소요된 일자는, day(큐에서 가져온 소요 일자) + 1 로 계산하여 next_day 에 저장.

⑨. next_x 와 next_y 의 값이 garden 리스트의 인덱스 범위를 벗어나지 않고, 해당하는 위치의 항목값이 = 0 인지 확인.

⑩. garden 리스트에서 새로 구한 영역의 항목값을 1 로 할당하여 개화한 상태로 변경하고 리턴 값을 저장하는 변수 answer 에 새로 개화하는데 소요된 일자인 next_day 를 할당.

⑪. 새로 개화한 영역에 대한 정보를 (x 좌표, y 좌표, 소요 일자) 형태의 튜플로 구성하여 큐에 새로운 항목으로 추가.

※ garden = [[0, 0, 0], [0, 1, 0], [0, 0, 0]] 인 경우 큐(Queue)의 변화



2. 문제 2

1) 문제 코드

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

def solution(K, words):
    #여기에 코드를 작성해주세요.
    answer = 0
    return answer

#아래는 테스트케이스 출력을 해보기 위한 코드입니다.
K = 10
words = ["nice", "happy", "hello", "world", "hi"]
ret = solution(10, words)

#[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.
- 한 줄에 K 자를 적을 수 있는 메모장에 주어진 단어들을 한 칸씩 띄워 적어야 하고, 한 줄에 적지 못하는 단어는 그 다음 줄부터 새로 적어야 함.
- 위의 조건에서 주어진 단어를 모두 적었을 때 생성되는 줄 수를 리턴하는 프로그램을 작성.

3) 정답

- 주요 아이디어 정리 : 한 줄에 10 글자를 적을 수 있고, 주어진 단어들이 [“nice” , “happy”, “hello”, “world”, “hi”] 인 경우

Line No.1	“nice_happy”	10 칸이 모두 채워졌으므로 세 번째 단어는 다음 줄에 작성.
Line No.2	“hello____”	남은 칸이 5 칸인데 네 번째 단어를 적기 위해 필요한 공간은 공란 1 칸 + world 의 글자 수 5 = 6 칸이므로 네 번째 단어는 다음 줄에 작성.
Line No.3	“world_hi__”	네 번째 단어와 다섯 번째 단어를 세 번째 줄에 모두 작성.

- 정답 코드

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

def solution(K, words):
    #여기에 코드를 작성해주세요.
    answer = 1
    remain = K

    for w in words:
        if remain >= len(w):
            remain -= (len(w)+1)
        else:
            answer += 1
            remain = K
            remain -= (len(w)+1)
    return answer

#아래는 테스트케이스 출력을 해보기 위한 코드입니다.
K = 10
words = ["nice", "happy", "hello", "world", "hi"]
ret = solution(10, words)

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

① 작성된 줄 수를 저장하는 변수 answer 를 1 로 초기화 하고, 남은 칸 수를 저장하는 변수 remain 을 한 줄에 적을 수 있는 칸 수 K 로 초기화.

② for 문을 이용하여 매개 변수 words 에 있는 단어를 하나씩 w 로 받음.

③ 남은 칸 수가 가져온 단어의 길이와 같거나 크면 현재 작성 중인 줄에 한 칸을 띄우고 단어를 적을 수 있으므로 남은 칸 수 값을 갖고 있는 remain 에서 (단어 길이 + 공란 1 칸)에 대한 값을 차감.

④ 남은 칸 수가 가져온 단어의 길이보다 작으면 새로운 줄에 단어를 적어야 하므로 작성된 줄 수를 갖는 answer 를 1 만큼 증가시키고 남은 칸수를 저장하는 변수 remain 을 K 로 재할당.

⑤ 남은 칸 수 값을 갖는 remain 에서 (단어 길이 + 공란 1 칸)에 대한 값을 차감.

3. 문제 3

1) 문제 코드

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

def solution(arr, K):
    #여기에 코드를 작성해주세요.
    answer = 0
    return answer

#아래는 테스트케이스 출력을 해보기 위한 코드입니다.
arr = [9, 11, 9, 6, 4, 19]
K = 4
ret = solution(arr, K)

#[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.
- 매개변수 arr 리스트에서 K 개의 항목을 선택하여 부분 리스트를 만드는데, 그 부분 리스트에서의 최댓값과 최솟값의 차이가 최소가 되도록 선택해서 그 최솟값을 리턴 해야 함.

3) 정답

- 주요 아이디어 정리
  - 부분 리스트에서 최대/최소 항목값의 차이가 가장 작은 것을 찾기 위해 원본 리스트의 항목 크기가 비슷한 값들이 모이도록 sort( ) 메소드를 사용해서 정렬.
  - 정렬된 원본 리스트에 대해서 K 개만큼 슬라이싱 한 것을 부분 리스트로 만들고 부분 리스트의 최대/최소 값의 차이를 비교.

- 정답 코드

```
1   #다음과 같이 import를 사용할 수 있습니다.
2   #import math
3
4   def solution(arr, K):
5       #여기에 코드를 작성해주세요.
6       answer = 0
7   ①   arr.sort()
8   ②   min_dif = 10001
9       for i in range(len(arr)+1-K):
10  ③       sub_arr = arr[i:i+K]
11
12  ④       min_dif = min(min_dif,sub_arr[K-1]-sub_arr[0])
13      answer = min_dif
14
15      return answer
16
17  #아래는 테스트케이스 출력을 해보기 위한 코드입니다.
18  arr = [9, 11, 9, 6, 4, 19]
19  K = 4
20  ret = solution(arr, K)
21
22  #[실행] 버튼을 누르면 출력 값을 볼 수 있습니다.
23  print("solution 함수의 반환 값은", ret, "입니다.")
```

① 매개변수로 받은 리스트 arr 에 대해 sort( ) 메소드를 사용하여 비슷한 크기의 값들이 인접하도록 정렬.

② (최댓값 – 최솟값) 계산에 대한 최솟값을 저장하는 변수로 mid_dif 를 사용하고 초기값을 arr 항목이 갖는 최댓값보다 1 큰 값이 10001 로 지정.

③ for 문을 이용하여 arr 의 0 번 인덱스 항목부터 시작해서 K 개를 슬라이싱하여 부분 리스트인 sub_arr 을 생성.

④ 기존에 산출한 mid_dif 와 새로 생성한 부분 리스트의 최댓값과 최솟값의 차이를 비교하여 더 적은 값을 mid_dif 에 저장.

- arr 이 정렬되어 있는 상태이므로 sub_arr[0] 은 선택한 K 의 항목 중 최솟값이고, sub_arr[K-1] 은 선택한 K 의 항목 중 최댓값.

4. 문제 4

1) 문제 코드

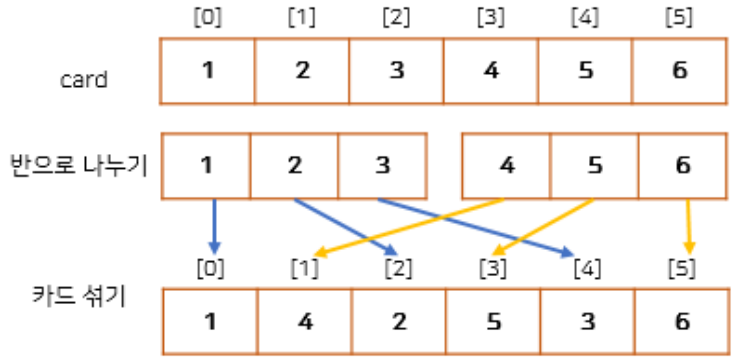
```python
def solution(n, mix, k):
    answer = 0
    card = [_ for _ in range(1, n+1)]
    while mix != 0:
        mix = mix - 1
        card_a, card_b = [0 for _ in range(n//2)], [0 for _ in range(n//2)]
        for i in range(0, n):
            if i < n//2:
                card_a[i] = card[i]
            else:
                card_b[i] = card[i]
        for i in range(0, n):
            if i % 2 == 0:
                card[i] = card_a[i//2]
            else:
                card[i] = card_b[i//2]
    answer = card[k-1]
    return answer

#아래는 테스트케이스 출력을 해보기 위한 코드입니다. 아래에는 잘못된 부분이 없으니 위의 코드만 수
n = 6
mix = 3
k = 3
ret = solution(n, mix, k)

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

2) 문제 개요

- 제시된 과제를 해결하기 위해 작성된 프로그램에서 잘못된 부분을 찾아 수정하는 문제.
- 1 부터 n 까지의 숫자를 가진 카드 뭉치를 이등분한 후 두 개의 뭉치에서 한 개씩 번갈아 가져와 섞었을 때 아래에서부터 k 번째에 있는 카드를 알아내기 위해 작성한 프로그램에서 잘못된 부분을 찾아 한 줄만 수정하는 문제.
- 1 부터 n 까지의 숫자를 리스트에 차례대로 저장한 후 문제에서 제시한 방법대로 숫자를 리스트에 재배열하는 패턴이 제시된 문제 코드로 제대로 구현되었는지 확인해야 함.

3) 정답

- 주요 아이디어 정리



➔ 두 개로 나눈 카드 뭉치에서 앞쪽 카드 뭉치의 카드들은 짝수 인덱스에 할당하고, 뒤쪽 카드 뭉치의 카드들은 홀수 인덱스에 할당하여 카드를 섞도록 구현.

- 정답 코드

```
def solution(n, mix, k):
    answer = 0
①   card = [_ for _ in range(1, n+1)]
    while mix != 0:
②       mix = mix - 1
③       card_a, card_b = [0 for _ in range(n//2)], [0 for _ in range(n//2)]
        for i in range(0, n):
④           if i < n//2:
                card_a[i] = card[i]
            else:
⑤               card_b[i-n//2] = card[i]
        for i in range(0, n):
⑥           if i % 2 == 0:
                card[i] = card_a[i//2]
            else:
⑦               card[i] = card_b[i//2]
⑧   answer = card[k-1]
    return answer

#아래는 테스트케이스 출력을 해보기 위한 코드입니다. 아래에는 잘못된 부분이 없으니 위의 코
n = 6
mix = 3
k = 3
ret = solution(n, mix, k)

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

①. 1 부터 n 까지의 수를 항목으로 갖는 card 리스트를 생성.

②. 카드 섞는 횟수를 1 만큼 차감.

③. card 를 절반으로 나누어 저장할 리스트 card_a, card_b 를 만들어서 항목값을 0 으로 초기화.

④. i 값이 card 의 항목 개수를 반으로 나눈 몫보다 작으면, card[i]를 card_a [i]로 저장.

⑤. i 값이 card 의 항목 개수를 반으로 나눈 몫보다 크면, (i - card 의 항목 개수를 반으로 나눈 몫)을 card_b 리스트의 인덱스로 지정하면서 card [i] 를 차례대로 저장. 즉 card 리스트의 절반 이후의 항목들이 card_b 의 0 번째 항목부터 차례대로 복사됨. 만일 문제

코드처럼 card_b[i] = card[i] 를 실행하면 card_b 리스트의 인덱스 범위를 벗어나게 되므로 오류 발생.

⑥. card 에 재배치할 항목의 인덱스가 짝수일 때는 card_a 의 해당하는 항목을 가져와서 저장.

⑦. card 에 재배치할 항목의 인덱스가 홀수일 때는 card_b 의 해당하는 항목을 가져와서 저장.

⑧. k 번째 수를 찾아 리턴하기 위해 card[k-1]를 answer 에 저장.

5. 문제 5

1) 문제 코드

```python
def solution(board):
    coins = [[0 for c in range(4)] for r in range(4)]
    for i in range(4):
        for j in range(4):
            if i == 0 and j == 0:
                coins[i][j] = board[i][j]
            elif i == 0 and j != 0:
                coins[i][j] = board[i][j] + coins[i][j-1]
            elif i != 0 and j == 0:
                coins[i][j] = board[i][j] + coins[i-1][j]
            else:
                coins[i][j] = board[i][j] + max(coins[i][j], coins[i-1][j-1])
    answer = coins[3][3]
    return answer

#아래는 테스트케이스 출력을 해보기 위한 코드입니다. 아래에는 잘못된 부분이 없으면 위의 코드만
board = [[6, 7, 1, 2], [3, 5, 3, 9], [6, 4, 5, 2], [7, 3, 2, 6]]
ret = solution(board)

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

2) 문제 개요

- 제시된 과제를 해결하기 위해 작성된 프로그램에서 잘못된 부분을 찾아 수정하는 문제.
- 오른쪽과 아래로만 이동할 수 있는 말을 사용해서 4 x 4 격자 보드의 가장 왼쪽 위에서 가장 오른쪽 아래로 말을 이동시키며 지나가는 구역의 코인을 누적 합산.
- 말이 보드의 종료 지점에 도착했을 때 최대로 얻을 수 있는 누적 코인을 계산하는 프로그램에서 잘못된 부분을 찾아 한 줄만 수정하는 문제.

3) 정답

- 주요 아이디어 정리

< 보드 구역별 코인 >

< 보드 구역별 최대 누적 코인 >

6	7	1	2
3	5	3	9
6	4	5	2
7	3	2	6

① 첫 번째 행의 각 구역별 최대 누적 코인은 현재 구역의 바로 왼쪽 구역의 누적 코인 값 + 현재 구역 코인으로 계산.

6	13	14	16

② 첫 번째 열의 각 구역별 최대 누적 코인은 바로 윗줄 구역의 누적 코인 값 + 현재 구역 코인으로 계산.

6	13	14	16
9			
15			
22			

③ 보드의 나머지 구역별 최대 누적 코인은 현재 구역의 바로 왼쪽 구역의 누적 코인과 현재 구역의 바로 윗줄 구역의 누적 코인 중 큰 값 + 현재 구역 코인으로 계산.(윗줄의 누적 코인이 더 크기 때문에 : 13 + 5 = 18)

6	13	14	16
9	18		
15			
22			

- 정답 코드

```
def solution(board):
    coins = [[0 for c in range(4)] for r in range(4)]
    for i in range(4):
        for j in range(4):
            if i == 0 and j == 0:
                coins[i][j] = board[i][j]
            elif i == 0 and j != 0:
                coins[i][j] = board[i][j] + coins[i][j-1]
            elif i != 0 and j == 0:
                coins[i][j] = board[i][j] + coins[i-1][j]
            else:
                coins[i][j] = board[i][j] + max(coins[i-1][j], coins[i][j-1])
    answer = coins[3][3]
    return answer

#아래는 테스트케이스 출력을 해보기 위한 코드입니다. 아래에는 잘못된 부분이 없으면 위의 코드만
board = [[6, 7, 1, 2], [3, 5, 3, 9], [6, 4, 5, 2], [7, 3, 2, 6]]
ret = solution(board)

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

①. 각 구역의 누적 코인을 저장하는 4 x 4 크기의 2 차원 리스트를 생성하여 0 으로 초기화.

②. 현재 구역의 행과 열이 모두 0 이면 보드의 시작점이므로 현재 구역의 코인만 누적 코인 값으로 할당.

③. 현재 구역의 행 = 0 and 열 != 0 이면 시작점을 제외한 보드의 첫 번째 행에 위치한 구역이기 때문에 현재 구역의 이전 구역은 왼쪽 구역만 존재. → (현재 구역의 코인 값 + 바로 왼쪽 구역의 누적 코인)을 현재 구역의 누적 코인 값으로 할당.

④. 현재 구역의 행 != 0 and 열 = 0 이면 시작점을 제외한 보드의 첫 번째 열에 위치한 구역이기 때문에 현재 구역의 이전 구역은 위쪽 구역만 존재. → (현재 구역의 코인 값 + 바로 윗줄 구역의 누적 코인)을 현재 구역의 누적 코인 값으로 할당.

⑤. 나머지 구역의 누적 코인 값은 { 현재 구역의 코인 값 + max(바로 윗줄 구역의 누적 코인, 바로 왼쪽 구역의 누적 코인) } 으로 할당. 문제 코드는 계산하지도 않은 현 구역의 누적 코인 값과 현 구역의 왼쪽 대각선 위쪽 방향 구역 중 큰 값을 더하도록 되어 있어서 원하는 값을 얻을 수 없음.

⑥. 도착점에 저장되어 있는 누적 코인 값은 answer 에 저장.

6. 문제 6

1) 문제 코드

```
def solution(grid):
    answer = 0
    for i in range(4):
        for j in range(4):
            for k in range(j + 1, 4, 2):
                answer = max(answer, max(grid[i][j] + grid[j][k], grid[j][k] + grid[k][i]))
    return answer

#아래는 테스트케이스 출력을 해보기 위한 코드입니다. 아래에는 잘못된 부분이 없으니 위의 코드만 수정하세요.
grid = [[1, 4, 16, 1], [20, 5, 15, 8], [6, 13, 36, 14], [20, 7, 19, 15]]
ret = solution(grid)

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

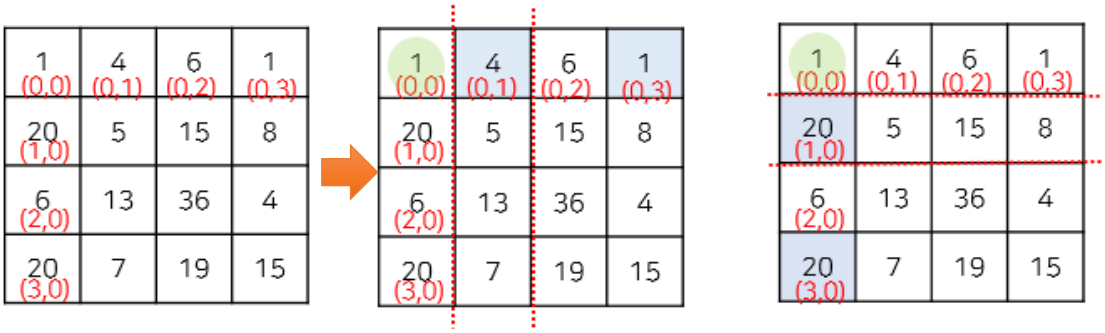
2) 문제 개요

- 제시된 과제를 해결하기 위해 작성된 프로그램에서 잘못된 부분을 찾아 수정하는 문제.
- 4 x 4 격자로 구성된 정사각형 종이를 가로축 혹은 세로축에 평행한 격자 선을 따라 접었을 때, 만나는 수의 합이 최대가 되는 것을 리턴하는 프로그램에서 잘못된 곳 한 줄만 찾아 수정.

3) 정답

- 주요 아이디어 정리

< 4x4 종이에 적힌 수 >　　< 종이를 접었을 때, (0,0) 좌표에 있는 1 이 만나는 수 >



➔ 종이를 세로축에 평행하게 접으면 (0, 1)에 있는 4 와 (0, 3)에 있는 1 을 만날 수 있음.

➔ 종이를 가로축에 평행하게 접으면 (1, 0)에 있는 20 와 (3, 0)에 있는 20 을 만날 수 있음.

∴ (row, column) 위치에 있는 수가 4 x 4 종이를 접었을 때 만나는 수

- 종이를 **세로**로 접을 경우 : (row, column+1)과 (row, column+3) 에 있는 수
- 종이를 **가로**로 접을 경우 : (row+1, column)과 (row+3, column) 에 있는 수

(단, 종이의 범위는 벗어나지 않아야 함.)

- 정답 코드

```
1   def solution(grid):
2       answer = 0
3   ①   for i in range(4):
4           for j in range(4):
5   ②           for k in range(j + 1, 4, 2):
6   ③               answer = max(answer, max(grid[i][j] + grid[i][k], grid[j][i] + grid[k][i]))
7       return answer
8
9   #아래는 테스트케이스 출력을 해보기 위한 코드입니다. 아래에는 잘못된 부분이 없으니 위의 코드만 수정하세요.
10  grid = [[1, 4, 16, 1], [20, 5, 15, 8], [6, 13, 36, 14], [20, 7, 19, 15]]
11  ret = solution(grid)
12
13  #[실행] 버튼을 누르면 출력 값을 볼 수 있습니다.
14  print("solution 함수의 반환 값은", ret, "입니다.")
```

① 중첩 for 문을 이용하여 0 부터 3 까지의 수를 i 와 j 에 각각 받아서 좌표를 구성하는 수로 이용.

② for 문을 이용하여 j+1 부터 2 만큼 증가하여 4 보다 작은 수를 k 로 받음.

③ 기준 좌표 (i, j) 에 대해서 세로로 접어서 만나는 항목인 (i, k)을 더한 값, 기준 좌표 (j, i) 에 대해서 가로로 접어서 만나는 항목인 (k, i)을 더한 값 중 큰 값을 구한 후, 기존에 구해 놓은 최댓값과 비교하여 최댓값을 다시 지정.

7. 문제 7

1) 문제 코드

```python
def solution(K, numbers, up_down):
    left = 1
    right = K
    for num, word in zip(numbers, up_down):
        if word == "UP":
            left = @@@
        elif word == "DOWN":
            right = @@@
        elif word == "RIGHT":
            return 1
    return @@@

#아래는 테스트케이스 출력을 해보기 위한 코드입니다.
K1 = 10
numbers1 = [4, 9, 6]
up_down1 = ["UP", "DOWN", "UP"]
ret1 = solution(K1, numbers1, up_down1)

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

K2 = 10
numbers2 = [2, 1, 6]
up_down2 = ["UP", "UP", "DOWN"]
ret2 = solution(K2, numbers2, up_down2)

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

K3 = 100
numbers3 = [97, 98]
up_down3 = ["UP", "RIGHT"]
ret3 = solution(K3, numbers3, up_down3)

#[실행] 버튼을 누르면 출력 값을 볼 수 있습니다.
print("solution 함수의 반환 값은", ret3, "입니다.")
```

2) 문제 개요

- 제시된 과제를 해결하기 위해 작성된 프로그램에서 빈 곳을 채우는 문제.
- 출제자가 생각한 수를 참가자가 맞추는 게임을 구현하는 프로그램으로 참가자가 제시한 수에 대해서 출제자가 "UP" / "DOWN" / "RIGHT" 을 대답하고 그 대답을 기초로 정답이 될 수 있는 수의 개수를 리턴 하는 프로그램에서 빈 곳에 알맞은 코드를 채워 넣는 문제.
- 출제자가 생각하는 수의 범위를 1 ~ K 로 시작하여 참가자가 제시하는 수에 대한 출제자의 답을 보고 수의 범위를 좁혀가며 정답의 개수를 구함.

3) 정답

- 주요 아이디어 정리

✓ 참가자가 제시한 수에 대한 출제자의 답이 'UP' 인 경우, 추측하는 수 범위의 최소 한계값을 참가자가 제시한 수와 현재 최소 한계값 중 큰 것으로 변경.

✓ 참가자가 제시한 수에 대한 출제자의 답이 'DOWN' 인 경우, 추측하는 수 범위의 최대 한계값을 참가자가 제시한 수와 현재 최대 한계값 중 작은 것으로 변경.

- 정답 코드

```
def solution(K, numbers, up_down):
    left = 1
    right = K
①   for num, word in zip(numbers, up_down):
②       if word == "UP":
            left = max(left,num)
③       elif word == "DOWN":
            right = min(right, num)
④       elif word == "RIGHT":
            return 1
⑤   return right-left-1

#아래는 테스트케이스 출력을 해보기 위한 코드입니다.
K1 = 10
numbers1 = [4, 9, 6]
up_down1 = ["UP", "DOWN", "UP"]
ret1 = solution(K1, numbers1, up_down1)

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

K2 = 10
numbers2 = [2, 1, 6]
up_down2 = ["UP", "UP", "DOWN"]
ret2 = solution(K2, numbers2, up_down2)

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

K3 = 100
numbers3 = [97, 98]
up_down3 = ["UP", "RIGHT"]
ret3 = solution(K3, numbers3, up_down3)

#[실행] 버튼을 누르면 출력 값을 볼 수 있습니다.
print("solution 함수의 반환 값은", ret3, "입니다.")
```

① zip( ) 함수를 이용하여 numbers 리스트와 up_down 리스트에서 같은 인덱스를 갖는 항목을 쌍으로 가져와 num, word 로 받음.

② word 의 값이 "UP" 이면 최소 한계값을 num 과 현재의 최소 한계값 중 큰 값으로 변경.

③ word 의 값이 "DOWN" 이면 최대 한계값을 num 과 현재의 최대 한계값 중 작은 값으로 변경.

④ word 의 값이 "RIGHT" 이면 num 이 정답이므로 1 을 리턴.

⑤ 정답이 될 수 있는 수의 개수로 최대 한계값 – 최소 한계값 – 1 을 계산하여 리턴.
(ex: 정답이 될 수 있는 범위가 1 보다 크고 4 보다 작은 경우 정답이 될 수 있는 수의 개수는 4 – 1 – 1 = 2 개.)

8. 문제 8

1) 문제 코드

```python
INC = 0
DEC = 1

def func_a(arr):
    length = len(arr)
    ret = [0 for _ in range(length)]
    ret[0] = 1
    for i in range(1, length):
        if arr[i] != arr[i-1]:
            ret[i] = ret[i-1] + 1
        else:
            ret[i] = 2
    return ret

def func_b(arr):
    global INC, DEC
    length = len(arr)
    ret = [0 for _ in range(length)]
    ret[0] = -1
    for i in range(1, length):
        if arr[i] > arr[i-1]:
            ret[i] = INC
        elif arr[i] < arr[i-1]:
            ret[i] = DEC
    return ret

def func_c(arr):
    ret = max(arr)
    if ret == 2:
        return 0
    return ret

def solution(S):
    check = func_@@@(@@@)
    dp = func_@@@(@@@)
    answer = func_@@@(@@@)
    return answer

# 아래는 테스트케이스 출력을 해보기 위한 코드입니다.
S1 = [2, 5, 7, 3, 4, 6, 1, 8, 9]
ret1 = solution(S1)
# [실행] 버튼을 누르면 출력 값을 볼 수 있습니다.
print("solution 함수의 반환 값은", ret1, "입니다.")

S2 = [4, 3, 2, 1, 10, 6, 9, 7, 8]
ret2 = solution(S2)
# [실행] 버튼을 누르면 출력 값을 볼 수 있습니다.
print("solution 함수의 반환 값은", ret2, "입니다.")
```

2) 문제 개요

- 문제 코드 안에 작성된 함수를 파악한 후, 제시된 과제를 해결하기 위한 알고리즘 대로 알맞은 함수를 호출하도록 코드를 완성하는 문제.
- 주어진 수열에서 지그재그 수열로 구성된 구간의 최대 길이를 구하여 return 하기 위해 알맞은 함수와 인수를 적는 문제.
- 지그재그 수열이란 첫 번째 원소부터 인접한 원소의 차이가 증가 → 감소 → 증가 → 감소 … 혹은 감소 → 증가 → 감소 → 증가 … 순으로 나타나는 수열을 뜻함.

3) 정답

- 주요 아이디어 정리

✓ 주어진 리스트의 항목을 이전 항목과 비교하여 증가/감소했는지 나타내는 리스트를 생성.

✓ 생성한 리스트에서 각 항목별로 증가/감소 상태를 번갈아 나타내는 길이를 계산하여 또 다른 리스트에 저장해서 활용.

- 정답 코드

```
INC = 0
DEC = 1

def func_a(arr):
    length = len(arr)
①   ret = [0 for _ in range(length)]
    ret[0] = 1
    for i in range(1, length):
②       if arr[i] != arr[i-1]:
③           ret[i] = ret[i-1] + 1
        else:
            ret[i] = 2
    return ret

def func_b(arr):
    global INC, DEC
    length = len(arr)
④   ret = [0 for _ in range(length)]
    ret[0] = -1
    for i in range(1, length):
⑤       if arr[i] > arr[i-1]:
            ret[i] = INC
⑥       elif arr[i] < arr[i-1]:
            ret[i] = DEC
    return ret

def func_c(arr):
⑦   ret = max(arr)
⑧   if ret == 2:
        return 0
    return ret

def solution(S):
⑨   check = func_b(S)
⑩   dp = func_a(check)
⑪   answer = func_c(dp)
    return answer
```

❖ func_a( ) : 리스트의 각 항목을 마지막으로 하는 지그재그 수열 중 가장 긴 구간의 길이를 각 항목별로 구하여 리스트로 저장해서 리턴하는 함수.

①. 매개변수 arr 길이와 동일한 리스트 ret 를 생성하고 항목값을 모두 0 으로 초기화.

②. arr 의 i 번 인덱스 항목을 그 이전 항목과 비교했을 때 같지 않으면

- arr 리스트의 항목들을 INC(증가) 혹은 DEC(감소) 를 값으로 가짐.
- 현재 항목과 이전 항목의 값이 다르다는 것은 증가와 감소가 번갈아 나타나는 것을 의미.

- 이런 경우는 지그재그 수열이 되는 구간이므로 ret[i – 1] + 1 을 ret[i] 의 값으로 저장. (리스트의 이전 항목을 이용하는 다이나믹 프로그래밍 방법)

③. arr 의 i 번 인덱스 항목을 그 이전 항목과 비교했을 때 같으면 ret[i]의 값을 2 로 저장.

- 만일 arr 에서 이전 항목과 현재 항목이 모두 INC(증가) 를 갖는 경우, 다음 항목이 DEC(감소)이 오게 되면 다음 항목에 대한 지그재그 수열 구간의 길이가 3 으로 구해져야 하기 때문.

  ex) S = [1, 2, 5, 3] 인 경우

  증감을 나타내는 리스트 arr = [-1, INC, INC, DEC] 으로 집계됨.

  arr[1] == arr[2] == INC 이고, arr[3] == DEC 이므로

  ret[2] = 2 , ret[3] = 3 가 되어 지그재그 수열 구간인 [2, 5, 3] 에 대한 길이 3 을 구하게 됨.

❖ func_b( ) : 리스트에 저장되어 있는 각 항목의 숫자가 이전 항목보다 증가/감소했는지 나타내는 리스트를 생성하여 리턴하는 함수.

④. arr 길이와 동일한 길이의 리스트 ret 를 생성하여 항목값을 0 으로 초기화.

⑤. arr[i]이 이전 항목보다 크면 ret[i]에 INC 에 저장되어 있는 값을 할당.

⑥. arr[i]이 이전 항목보다 작으면 ret[i]에 DEC 에 저장되어 있는 값을 할당.

❖ func_c( ) : 리스트에 저장되어 지그재그 구간의 길이 중 가장 큰 값을 리턴하는 함수.

⑦. arr 에 저장되어 있는 항목값 중 최댓값을 구하여 ret 에 저장.

⑧. ret 에 저장된 값이 2 이면 0 을 리턴.

❖ solution( )

⑨. func_b( ) 를 이용하여 S 에 저장되어 있는 항목들을 이전 항목과 비교한 결과를 check 에 저장.

⑩. func_a( ) 를 이용하여 check 에 저장되어 있는 증감 현황에 대해서 각 항목별 지그재그 수열의 최대 길이를 구하여 dp 에 저장.

⑪. func_c( ) 를 이용하여 dp 에 저장되어 있는 최댓값을 구함.

9. 문제 9

1) 문제 코드

```python
def func_a(stack):
    return stack.pop()

def func_b(stack1, stack2):
    while not func_@@@(@@@):
        item = func_@@@(@@@)
        stack2.append(item)

def func_c(stack):
    return (len(stack) == 0)

def solution(stack1, stack2):
    if func_@@@(@@@):
        func_@@@(@@@)

    answer = func_@@@(@@@)
    return answer

#아래는 테스트케이스 출력을 해보기 위한 코드입니다.
stack1_1 = [1,2]
stack2_1 = [3,4]
ret1 = solution(stack1_1, stack2_1)
#[실행] 버튼을 누르면 출력 값을 볼 수 있습니다.
print("solution 함수의 반환 값은", ret1, "입니다.")

stack1_2 = [1,2,3]
stack2_2 = []
ret2 = solution(stack1_2, stack2_2)
#[실행] 버튼을 누르면 출력 값을 볼 수 있습니다.
print("solution 함수의 반환 값은", ret2, "입니다.")
```

2) 문제 개요

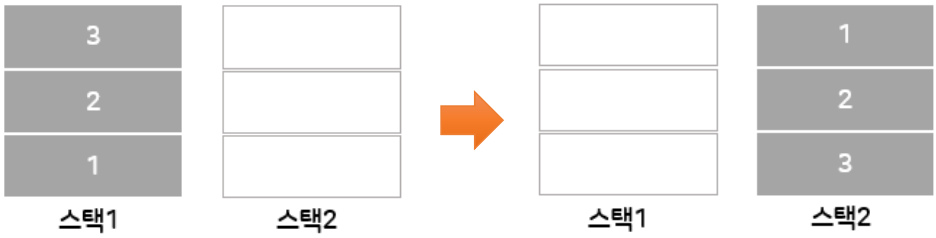
- 문제 코드 안에 작성된 함수를 파악한 후, 제시된 과제를 해결하기 위한 알고리즘 대로 알맞은 함수를 호출하도록 코드를 완성하는 문제.
- LIFO 방식으로 자료를 관리하는 스택(Stack) 두 개를 이용하여 FIFO 방식으로 자료를 관리하는 큐(Queue)의 pop(dequeue)함수를 구현하는 프로그램. .
- 리스트 두 개를 두 개의 스택으로 사용하여 Queue 의 pop( ) 함수와 같이 작동하도록 알맞은 함수와 인수를 적는 문제.

3) 정답

- 주요 아이디어 정리

✓ 스택 1 과 스택 2 가 있는데, 스택 2 가 비어 있으면 스택 1 의 모든 항목을 pop( ) 하여 스택 2 로 push( )를 수행한 후에 스택 2 에서 pop( ) 을 실행.

✓ 스택 2 가 이미 값을 가지고 있다면, 이미 스택 1 에 있는 값을 스택 2 로 옮긴 것이므로 스택 2 에서 pop( ) 을 수행..



➔ 스택 1 의 모든 항목을 pop( )해서 스택 2 로 push( ) 를 하면 스택 2 에 역순으로 저장.

- 정답 코드

```
1   def func_a(stack):
2   ①    return stack.pop()
3
4   def func_b(stack1, stack2):
5   ②    while not func_c(stack1):
6   ③        item = func_a(stack1)
7   ④        stack2.append(item)
8
9   def func_c(stack):
10  ⑤    return (len(stack) == 0)
11
12  def solution(stack1, stack2):
13  ⑥    if func_c(stack2):
14  ⑦        func_b(stack1, stack2)
15
16  ⑧    answer = func_a(stack2)
17       return answer
18
19  #아래는 테스트케이스 출력을 해보기 위한 코드입니다.
20  stack1_1 = [1,2]
21  stack2_1 = [3,4]
22  ret1 = solution(stack1_1, stack2_1)
23
24  #[실행] 버튼을 누르면 출력 값을 볼 수 있습니다.
25  print("solution 함수의 반환 값은", ret1, "입니다.")
26
27  stack1_2 = [1,2,3]
28  stack2_2 = []
29  ret2 = solution(stack1_2, stack2_2)
30
31  #[실행] 버튼을 누르면 출력 값을 볼 수 있습니다.
32  print("solution 함수의 반환 값은", ret2, "입니다.")
```

❖ func_a( ) : stack 리스트에서 마지막 항목값을 리턴.

①. 매개변수 stack 리스트에 대해서 pop( ) 메소드를 실행하여 stack 에 저장된 마지막 항목값을 stack 에서 지우고 삭제한 마지막 항목을 리턴.

❖ func_b( ) : stack1 리스트에 있는 항목을 모두 가져와 stack2 리스트에 역순으로 추가.

②. func_c( ) 를 이용하여 stack1 의 길이가 0 인지 확인하여 0 이 아닌 동안 실행.

③. func_a( ) 를 이용하여 stack1 의 마지막 항목을 가져와서 item 에 할당.

④. item 에 있는 값을 stack2 리스트의 마지막 항목으로 추가.

❖ func_c( ) : stack 리스트의 길이가 0 인지 확인.

⑤. stack 리스트의 길이가 0 과 같은 지 논리연산을 수행하여 그 결과를 리턴.

❖ solution( )

⑥. func_c( ) 를 이용하여 stack2 리스트의 길이가 0 인지 확인.

⑦. func_b( ) 를 이용하여 stack1 리스트의 항목을 stack2 리스트에 역순으로 추가.

⑧. func_a( ) 를 이용하여 stack2 리스트의 마지막에 있는 항목값을 answer 로 받음.

10. 문제 10

1) 문제 코드

```python
class Job:
    def __init__(self):
        self.salary = 0

    def get_salary(self):
        return salary

class PartTimeJob@@@:
    def __init__(self, work_hour, pay_per_hour):
        super().__init__()
        self.work_hour = work_hour
        self.pay_per_hour = pay_per_hour

    @@@:
        self.salary = self.work_hour * self.pay_per_hour
        if self.work_hour >= 40:
            self.salary += (self.pay_per_hour * 8)
        return self.salary

class SalesJob@@@:
    def __init__(self, sales_result, pay_per_sale):
        super().__init__()
        self.sales_result = sales_result
        self.pay_per_sale = pay_per_sale

    @@@:
        if self.sales_result > 20:
            self.salary = self.sales_result * self.pay_per_sale * 3
        elif self.sales_result > 10:
            self.salary = self.sales_result * self.pay_per_sale * 2
        else:
            self.salary = self.sales_result * self.pay_per_sale
        return self.salary

def solution(part_time_jobs, sales_jobs):
    answer = 0
    for p in part_time_jobs:
        part_time_job = PartTimeJob(p[0], p[1])
        answer += part_time_job.get_salary()
    for s in sales_jobs:
        sale_job = SalesJob(s[0], s[1])
        answer += sale_job.get_salary()
    return answer

#아래는 테스트케이스 출력을 해보기 위한 코드입니다.
part_time_jobs = [[10, 5000], [43, 6800], [5, 12800]]
sales_jobs = [[3, 18000], [12, 8500]]
ret = solution(part_time_jobs, sales_jobs)

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

2) 문제 개요

- Job 클래스를 상속받도록 PartTimeJob 클래스와 SalesJob 클래스를 정의하는 문제.
- PartTimeJob 클래스와 SalesJob 클래스의 부모 클래스를 지정하고, 각각의 자식 클래스에서 부모 클래스인 Job 의 get_salary( ) 메소드를 재정의 하여 문제에서 제시된 대로 급여를 계산하도록 메소드 정의 부분의 빈 칸을 채워야 함.

3) 정답

```python
class Job:
    def __init__(self):
        self.salary = 0

    def get_salary(self):
        return salary

class PartTimeJob(Job):
    def __init__(self, work_hour, pay_per_hour):
        super().__init__()
        self.work_hour = work_hour
        self.pay_per_hour = pay_per_hour

    def get_salary(self):
        self.salary = self.work_hour * self.pay_per_hour
        if self.work_hour >= 40:
            self.salary += (self.pay_per_hour * 8)
        return self.salary

class SalesJob(Job):
    def __init__(self, sales_result, pay_per_sale):
        super().__init__()
        self.sales_result = sales_result
        self.pay_per_sale = pay_per_sale

    def get_salary(self):
        if self.sales_result > 20:
            self.salary = self.sales_result * self.pay_per_sale * 3
        elif self.sales_result > 10:
            self.salary = self.sales_result * self.pay_per_sale * 2
        else:
            self.salary = self.sales_result * self.pay_per_sale
        return self.salary

def solution(part_time_jobs, sales_jobs):
    answer = 0
    for p in part_time_jobs:
        part_time_job = PartTimeJob(p[0], p[1])
        answer += part_time_job.get_salary()
    for s in sales_jobs:
        sale_job = SalesJob(s[0], s[1])
        answer += sale_job.get_salary()
    return answer

#아래는 테스트케이스 출력을 해보기 위한 코드입니다.
part_time_jobs = [[10, 5000], [43, 6800], [5, 12800]]
sales_jobs = [[3, 18000], [12, 8500]]
ret = solution(part_time_jobs, sales_jobs)

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

❖ Job 클래스 정의

①. 생성자 메소드를 이용하여 멤버 변수 salary 를 생성하고 0 으로 초기화.

②. get_salary( ) 메소드는 멤버 변수 salary 에 저장되어 있는 값을 리턴.

❖ PartTimeJob 클래스 정의

③. PartTimeJob 클래스가 Job 클래스를 상속받도록 부모 클래스를 Job 으로 지정.

④. 부모 클래스의 생성자 메소드를 실행하여 멤버 변수 salary 를 생성하고 0 으로 초기화.

⑤. 매개 변수로 받은 값을 이용하여 멤버 변수 work_hour 와 멤버 변수 pay_per_hour 를 생성하고 초기화.

⑥. 부모 클래스 Job 에 있는 get_salary( ) 메소드를 오버라이드.

⑦. 멤버 변수 work_hour * payt_per_hour 를 계산한 값을 멤버 변수 salary 로 저장.

⑧. 멤버 변수 work_hour >= 40 이면 멤버 변수 pay_per_hour * 8 을 계산한 것을 멤버 변수 salary 에 추가로 더함.

❖ SalesJob 클래스 정의

⑨. SalesJob 클래스가 Job 클래스를 상속받도록 부모 클래스를 Job 으로 지정.

⑩. 부모 클래스의 생성자 메소드를 실행하여 멤버 변수 salary 를 생성하고 0 으로 초기화.

⑪. 매개 변수로 받은 값을 이용하여 멤버 변수 sales_result 와 멤버 변수 pay_per_sale 를 생성하고 초기화.

⑫. 부모 클래스 Job 에 있는 get_salary( ) 메소드를 오버라이드.