

04

조건문과 반복문

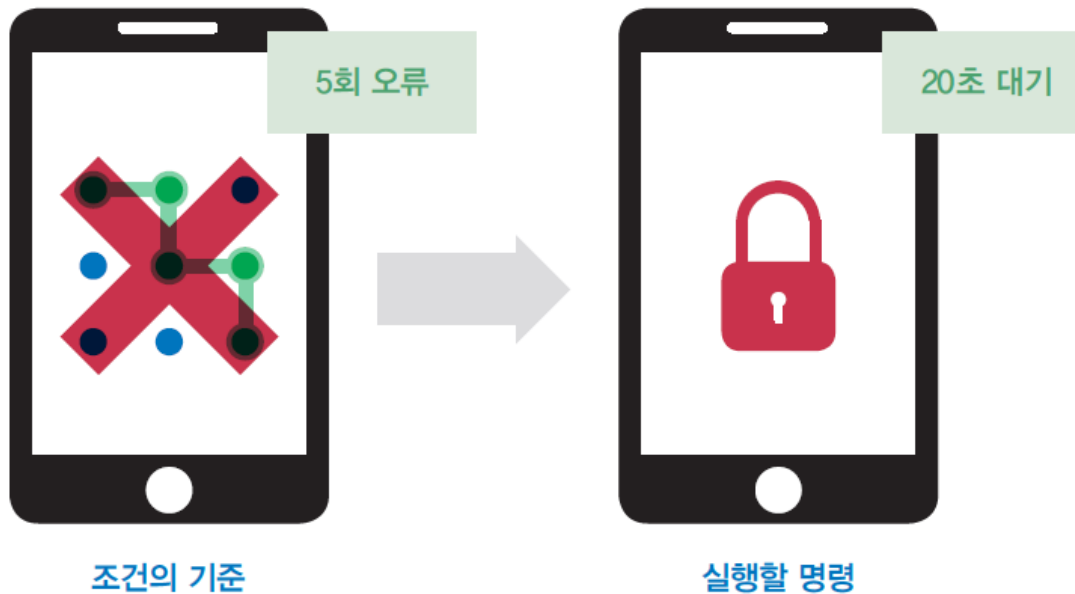
# 목차

1. 조건문
2. 반복문
3. 조건문과 반복문 실습
4. 코드의 오류를 처리하는 방법

# 01. 조건문

## ■ 조건문의 개념

- **조건문(conditional statement):** 조건에 따라 특정 동작을 하도록 하는 프로그래밍 명령어.
- 파이썬에서는 조건문을 사용하기 위해 if, else, elif 등의 명령 키워드를 사용. (switch 없음)
- 스마트폰 잠금 해제 패턴이 5회 틀리면, 20초 동안 대기 상태로 만들어라.



[ 조건문의 예시: 스마트폰 잠금 해제 패턴 ]

# 01. 조건문

## ■ if -else문

- if-else문의 기본 문법은 다음과 같다.

```
if <조건>:                # if를 쓰고 조건 삽입 후 ':' 입력
    <수행 명령 1-1>        # 들여쓰기 후, 수행 명령 입력
    <수행 명령 1-2>        # 같은 조건에서 실행일 경우 들여쓰기 유지
else:                    # 조건이 불일치할 경우 수행할 명령
    <수행 명령 2-1>        # 조건 불일치 시 수행할 명령 입력
    <수행 명령 2-2>        # 조건 불일치 시 수행할 명령 들여쓰기 유지
```

- ① if 뒤에는 참과 거짓을 판단할 수 있는 조건문이 들어가야 하고, 조건문이 끝나면 반드시 콜론(:)을 붙여야 한다.
- ② 들여쓰기를 사용하여 해당 조건이 참일 경우 수행할 명령을 작성한다.
- ③ if의 조건이 거짓일 경우 else문이 수행된다. else문은 생략해도 상관없다. 만약 조건에 해당하지 않는 경우에 따로 처리해야 한다면 else문을 넣으면 된다.

# 01. 조건문

## ■ if -else문

```
1 print("Tell me your age?")
2 myage = int(input())           # 나이를 입력받아 myage 변수에 할당
3 if myage < 30:                 # myage가 30 미만일 때
4     print("Welcome to the Club.")
5 else:                         # myage가 30 이상일 때
6     print("Oh! No. You are not accepted.")
```



```
Tell me your age?           ← 입력 대기
20                          ← 사용자 입력
Welcome to the Club.        ← 출력
```

# 01. 조건문

## ■ 조건의 판단 : 비교 연산자

- 비교 연산자(또는 조건 연산자): 어떤 것이 큰지 작은지 같은지를 비교하는 것으로, 그 결과는 참(True)이나 거짓(False)이 된다.

비교 연산자	비교 상태	설명
$x < y$	~보다 작음	x가 y보다 작은지 검사
$x > y$	~보다 큼	x가 y보다 큰지 검사
$x == y$	같음	x와 y의 값이 같은지 검사
$x \text{ is } y$	같음(메모리 주소)	x와 y의 메모리 주소가 같은지 검사
$x != y$	같지 않음	x와 y의 값이 같지 않은지 검사
$x \text{ is not } y$	같지 않음(메모리 주소)	x와 y의 메모리 주소가 같지 않은지 검사
$x \geq y$	크거나 같음	x가 y보다 크거나 같은지 검사
$x \leq y$	작거나 같음	x가 y보다 작거나 같은지 검사

# 01. 조건문

## ■ 조건의 판단 : True와 False의 치환

- 컴퓨터는 기본적으로 이진수만 처리할 수 있으며, True는 1로, False는 0으로 처리.
- 아래 코드를 실행하면 True가 출력된다. 그 이유는 앞서 설명한 것처럼 컴퓨터는 존재하면 True, 존재하지 않으면 False로 처리하기 때문이다.

```
>>> if 1: print("True")  
... else: print("False")
```

- 아래 코드를 실행하면 True가 출력된다. 먼저  $3 > 5$ 는 False이고 False는 결국 0으로 치환된다. 그래서 이것을 다시 치환하면  $(0) < 10$ 이 되고, 이 값은 참이므로 True가 반환된다.

```
>>> (3 > 5) < 10
```

# 01. 조건문

## ■ 조건의 판단 : 논리 연산자

- 논리 연산자는 and · or · not문을 사용해 조건을 확장할 수 있다.

연산자	설명	예시
and	두 값이 모두 참일 경우 True, 그렇지 않을 경우 False	(7 > 5) and (10 > 5)는 True (7 > 5) and (10 < 5)는 False
or	두 값 중 하나만 참일 경우 True, 두 값 모두 거짓일 경우 False	(7 < 5) or (10 > 5)는 True (7 < 5) or (10 < 5)는 False
not	값을 역으로 반환하여 판단	not (7 < 5)는 True not (7 > 5)는 False

[논리 연산자]



# 01. 조건문

## ■ 조건의 판단 : 논리 연산자

- and는 둘 다 참이어야 True, or는 둘 중 하나만 참이어도 True, not은 참이면 False이고 거짓이면 True를 출력한다.

```
>>> a = 8
>>> b = 5
>>> a == 8 and b == 4
False
>>> a > 7 or b > 7
True
>>> not (a > 7)
False
```

# 01. 조건문

## ■ if-elif-else문

- 중첩 if문을 간단히 표현하려면 if-elif-else문을 사용한다.
- 다음 같은 점수판이 있다고 가정하자.

점수(score)	학점(grade)
98	
37	
16	
86	
71	
63	

[점수판]

# 01. 조건문

## ■ if-elif-else문

- 점수에 맞는 학점을 주기 위해 [코드 4-2]와 같이 코드를 입력하면, 어떤 학점으로 계산될까?

코드 4-2 grade.py

```
1 score = int(input("Enter your score: "))
2
3 if score >= 90:
4     grade = 'A'
5 if score >= 80:
6     grade = 'B'
7 if score >= 70:
8     grade = 'C'
9 if score >= 60:
10    grade = 'D'
11 if score < 60:
12    grade = 'F'
13
14 print(grade)
```

Enter your score: 98

← 사용자 점수 입력

D

← 잘못된 값 출력

# 01. 조건문

## ■ if-elif-else문

- [코드 4-2]의 문제를 해결하기 위해서는 여러 개의 조건을 하나의 if문에서 검토할 수 있도록 elif를 사용한 if-elif-else문으로 작성해야 한다. elif는 else if의 줄임말로, if문과 같은 방법으로 조건문을 표현할 수 있다.

코드 4-3 if-elif-else.py

```
1 score = int(input("Enter your score: "))
2
3 if score >= 90: grade = 'A'           # 90 이상일 경우 A
4 elif score >= 80: grade = 'B'        # 80 이상일 경우 B
5 elif score >= 70: grade = 'C'        # 70 이상일 경우 C
6 elif score >= 60: grade = 'D'        # 60 이상일 경우 D
7 else: grade = 'F'                    # 모든 조건에 만족하지 못할 경우 F
8
9 print(grade)
```

Enter your score: 98

A

← 사용자 점수 입력

← 올바른 값 출력

## 02. Lab: 어떤 종류의 학생인지 맞추기

### ■ 실습 내용

- 조건문을 이용하여 '어떤 종류의 학생인지 맞추는 프로그램'을 만들어 보자.
- 이 프로그램을 작성하는 규칙은 다음과 같다.

- 나이는 (2020 - 태어난 연도 + 1)로 계산
- 26세 이하 20세 이상이면 '대학생'
- 20세 미만 17세 이상이면 '고등학생'
- 17세 미만 14세 이상이면 '중학생'
- 14세 미만 8세 이상이면 '초등학생'
- 그 외의 경우는 '학생이 아닙니다.' 출력

당신이 태어난 연도를 입력하세요.

1982

학생이 아닙니다.

← 입력 대기

← 자신이 태어난 연도 입력

← 어떤 종류의 학생인지 출력

## 01. 조건문

여기서  잠깐! 논리 연산자 없이 비교 연산자를 사용할 경우

```
>>> 1 <= 2 < 10
True
>>> 1 <= 100 < 10
False
```

- 이 코드가 순서대로 실행되면 먼저 `1 <= 100`을 해석하니 `True`로 반환되고, `True`는 1과 같으므로 `1 < 10`도 `True`로 반환되어야 한다. 하지만 파이썬에서는 `False`가 나온다.
- 파이썬은 사람에게 친숙하게 프로그래밍을 하겠다는 철학을 가지고 있으므로 이러한 처리가 가능하도록 지원한다. 사람처럼 그냥 조건을 작성하면 파이썬 인터프리터가 다 해결해 준다. 문제는 다른 언어들이 지원해 주지 않을 경우가 있으므로, 이렇게 적어 주는 것은 좋은 코드가 아니라는 걸 기억해야 한다.

## 02. 반복문

### ■ 반복문의 개념

- **반복문(loop)** : 말 그대로 문장을 반복해 만드는 것으로, 정해진 동작을 반복적으로 수행할 때 내리는 명령어이다.
- 반복문은 모든 프로그램에서 핵심적으로 사용된다. 반복문은 반복 시작 조건, 종료 조건, 수행 명령으로 구성되어 있으며, 들여쓰기와 블록(block)으로 구분한다.
- 파이썬의 반복문은 for와 while 등의 명령 키워드를 사용한다.

## 02. 반복문

### ■ for문

- **for문:** 기본적인 반복문으로, 반복 범위를 지정하여 반복을 수행한다.
- for문으로 반복문을 만들 때는 먼저 for를 입력하고 반복되는 범위를 지정해야 한다.
- 범위를 지정하는 방법에는 두 가지가 있다. 첫 번째 리스트를 사용하는 것이고 두 번째 방법은 [변수 자체를 출력하는 방법이다.



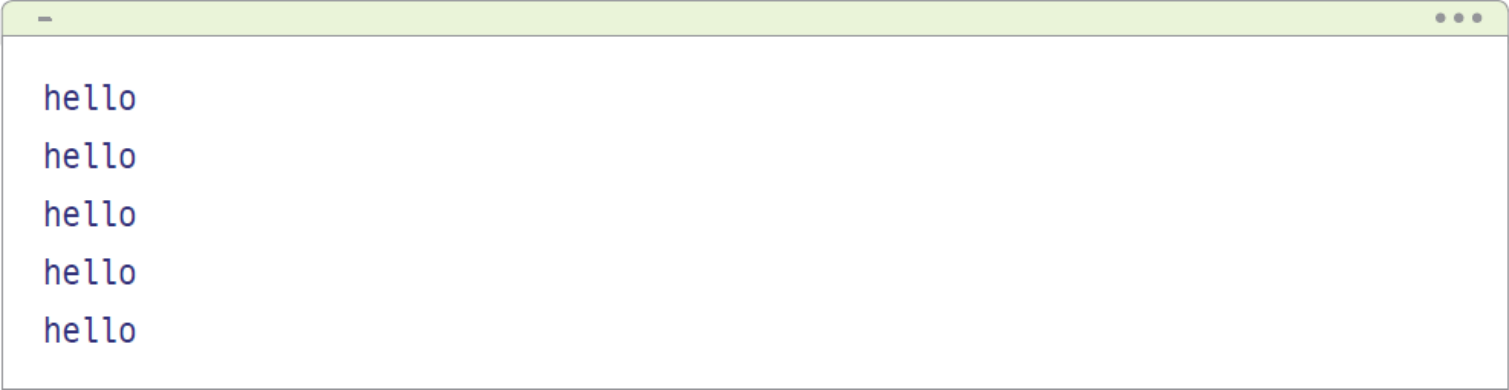
## 02. 반복문

### ■ for문

- ① 리스트를 사용해서 반복되는 범위를 지정하는 방법

코드 4-5 for1.py

```
1 for loopier in [1, 2, 3, 4, 5]:  
2     print("hello")
```



```
hello  
hello  
hello  
hello  
hello
```

## 03. 반복문

### ■ for문

- ② 변수 자체를 출력하여 반복되는 범위를 지정하는 방법

코드 4-6 for2.py

```
1 for loop in [1, 2, 3, 4, 5]:  
2     print(loop)
```

```
1  
2  
3  
4  
5
```

## 02. 반복문

### ■ for문

- 만약 100번 반복해야 한다면 코드를 어떻게 작성해야 할까? 리스트를 가지고 1부터 100까지 모든 값을 적기에는 너무 오래 걸린다. 이럴 때는 '**range**'라는 키워드를 사용한다.

코드 4-7 for3.py

```
1 for loop in range(100):  
2     print("hello")
```



```
hello  
:  
:  
hello
```

← 100번 반복  
← 생략

## 02. 반복문

### ■ for문 : [코드 4-7] 해석

- range 문법의 기본 구조

```
for 변수 in range(시작 번호, 마지막 번호, 증가값)
```

- range는 마지막 번호의 마지막 숫자 바로 앞까지 리스트를 만든다. 즉, range(1, 5)라고 하면 [1, 2, 3, 4]의 리스트를 만들고, range(0, 5)라고 하면 [0, 1, 2, 3, 4]의 리스트를 만든다.
- 앞의 시작 번호와 증가값은 생략 가능, 생략했을 경우 시작 번호는 0을, 증가값은 1을 사용.

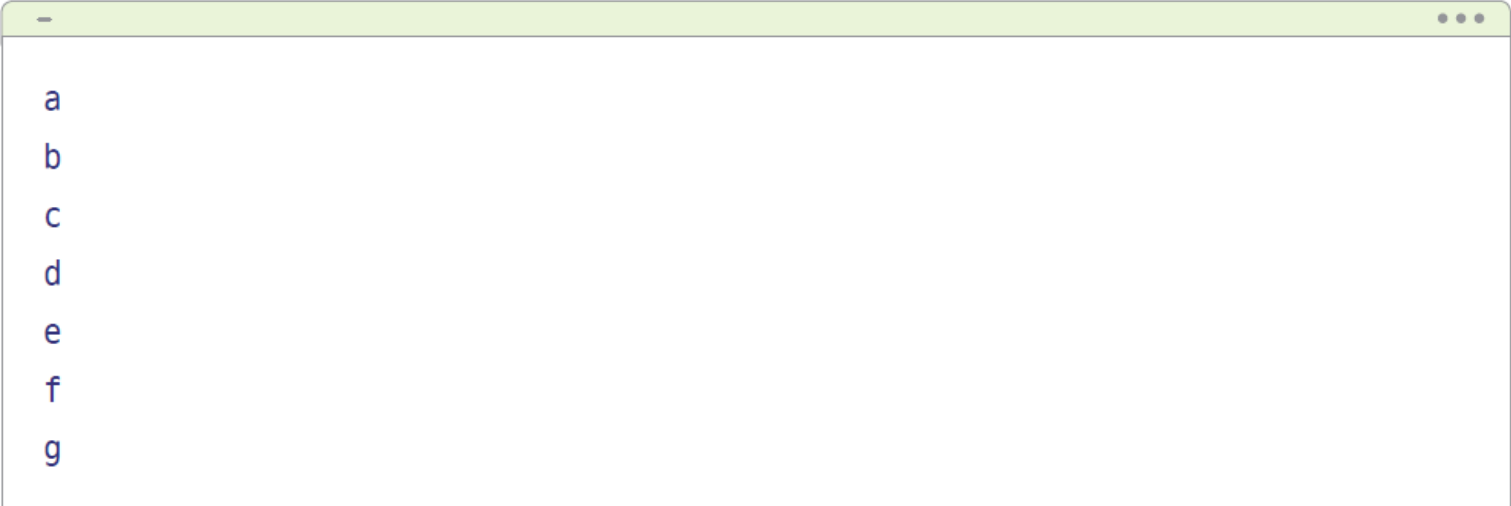
## 02. 반복문

### ■ for문

- 문자열도 리스트와 같은 연속적인 데이터를 표현하므로 각 문자를 변수 i에 할당하여 화면에 출력한다.

코드 4-8 for4.py

```
1 for i in 'abcdefg':  
2     print(i)
```



```
a  
b  
c  
d  
e  
f  
g
```

## 02. 반복문

### ■ for문

- 숫자를 화면에 출력하듯이 문자열로 이루어진 리스트의 값들도 사용할 수 있다.

코드 4-9 for5.py

```
1 for i in ['americano', 'latte', 'frappuccino']:
2     print(i)
```

```
americano
latte
frappuccino
```

## 02. 반복문

### ■ for문

- range 구문을 이용하여 1부터 9까지 2씩 증가시키는 for문을 [코드 4-10]에서 확인해 보자.

코드 4-10 for6.py

```
1 for i in range(1, 10, 2):      # 1부터 9까지 2씩 증가시키면서 반복문 수행
2     print(i)
```



```
1
3
5
7
9
```

## 02. 반복문

### ■ for문

- range 구문을 사용하여 10부터 2까지 1씩 감소시키는 반복문은 [코드 4-11]과 같다.

코드 4-11 for7.py

```
1 for i in range(10, 1, -1):    # 10부터 2까지 1씩 감소시키면서 반복문 수행
2     print(i)
```



```
10
9
8
7
6
5
4
3
2
```



## 02. 반복문

### ■ while문

- **while문** : 어떤 조건이 만족하는 동안 명령 블록을 수행하고, 해당 조건이 거짓일 경우 반복 명령문을 더는 수행하지 않는 구문이다.

코드 4-12 while.py

```
1 i = 1           # i 변수에 1 할당
2 while i < 10:    # i가 10 미만인지 판단
3     print(i)     # 조건을 만족할 때 i 출력
4     i += 1       # i에 1을 더하는 것을 반복하다가 i가 10이 되면 반복 종료
```



```
1
2
3
4
5
6
7
8
9
```

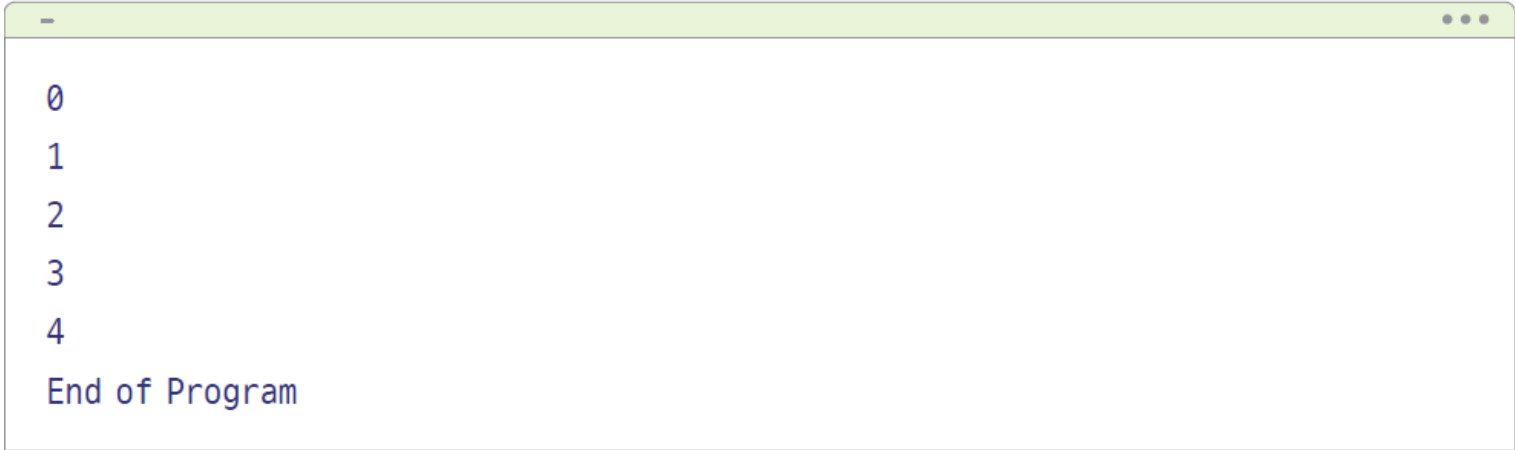
## 02. 반복문

### ■ 반복문의 제어 : **break**문

- **break**문 : 반복문에서 논리적으로 반복을 종료하는 방법이다.

코드 4-13 break.py

```
1 for i in range(10):  
2     if i == 5: break          # i가 5가 되면 반복 종료  
3     print(i)  
4 print("End of Program")      # 반복 종료 후 'End of Program' 출력
```



```
0  
1  
2  
3  
4  
End of Program
```

## 02. 반복문

### ■ 반복문의 제어 : **continue**문

- **continue**문 : 특정 조건에서 남은 명령을 건너뛰고 다음 반복문을 수행한다.

코드 4-14 continue.py

```
1 for i in range(10):  
2     if i == 5: continue      # i가 5가 되면 i를 출력하지 않음  
3     print(i)  
4 print("End of Program")      # 반복 종료 후 'End of Program' 출력
```

```
0  
1  
2  
3  
4  
6  
7  
8  
9  
End of Program
```

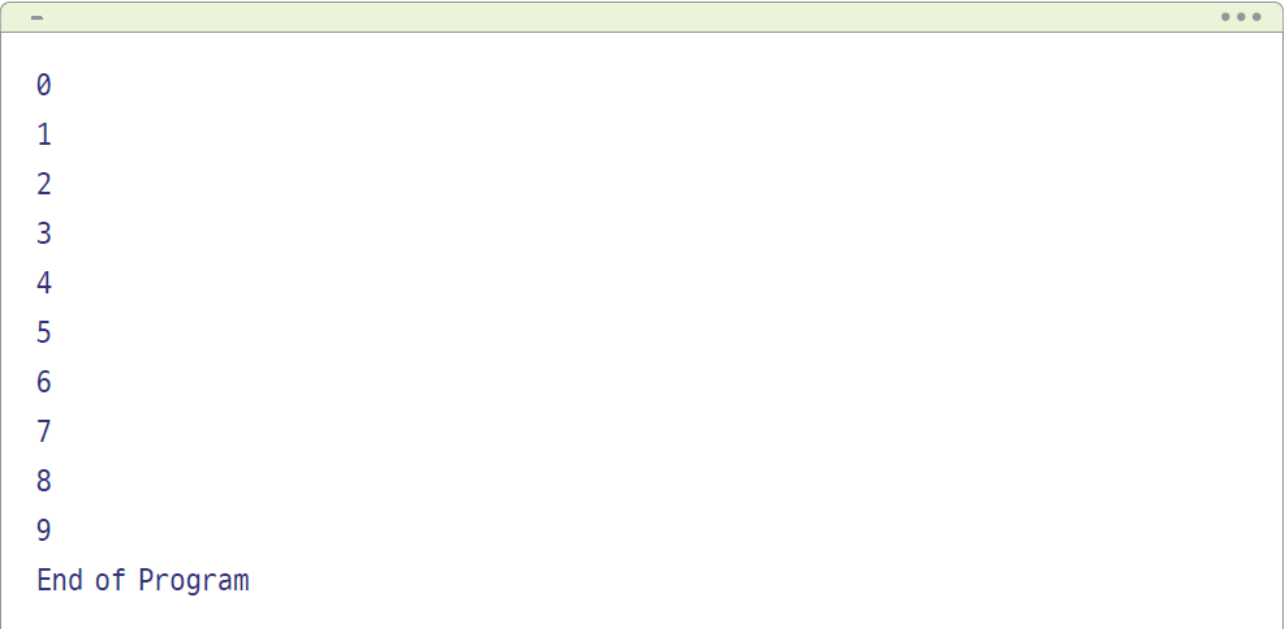
## 02. 반복문

### ■ 반복문의 제어 : **else문**

- **else문** : 어떤 조건이 완전히 끝났을 때 한 번 더 실행해 주는 역할을 한다.

코드 4-15 else.py

```
1 for i in range(10):  
2     print(i)  
3 else:  
4     print("End of Program")
```



```
0  
1  
2  
3  
4  
5  
6  
7  
8  
9  
End of Program
```

## 03. 조건문과 반복문 실습

### ■ 실습 내용 : 구구단 계산기

- 이번 Lab에서는 앞에서 배운 반복문을 이용하여 구구단 계산기를 만들어 보자.
- 구구단 계산기의 규칙은 다음과 같다.

- 프로그램이 시작되면 '구구단 몇 단을 계산할까?'가 출력된다.
- 사용자는 계산하고 싶은 구구단 숫자를 입력한다.
- 프로그램은 '구구단 n단을 계산한다.'라는 메시지와 함께 구구단의 결과를 출력한다.

## 03. 조건문과 반복문 실습

### ■ 실행 결과

구구단 몇 단을 계산할까?

5

구구단 5단을 계산한다.

5 × 1 = 5

5 × 2 = 10

⋮

5 × 8 = 40

5 × 9 = 45

```
1 print("구구단 몇 단을 계산할까?")
2 user_input = input()
3 print("구구단", user_input, "단을 계산한다.")
4 int_input = int(user_input)
5 for i in range(1, 10):
6     result = int_input * i
7     print(user_input, "x", i, "=", result)
```

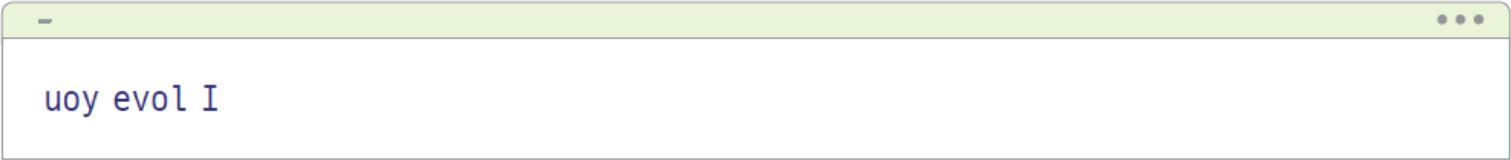
## 03. 조건문과 반복문 실습

### ■ 문자열 역순 출력

- **reverse\_sentence** : 입력된 문자열을 역순으로 출력하는 변수이다.

코드 4-17 reverse\_sentence.py

```
1 sentence = "I love you"
2 reverse_sentence = ' '
3 for char in sentence:
4     reverse_sentence = char + reverse_sentence
5 print(reverse_sentence)
```

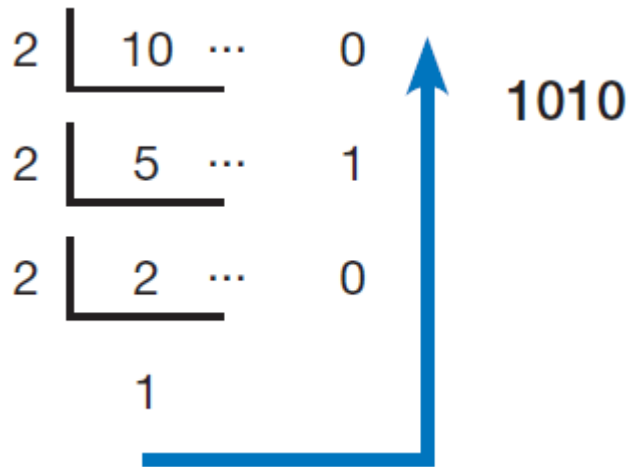


uoy evol I

### 03. 조건문과 반복문 실습

#### ■ 십진수를 이진수로 변환

- 십진수 숫자를 2로 계속 나눈 후, 그 나머지를 역순으로 취하면 이진수가 된다.



[십진수를 이진수로 변환하는 방법]



## 03. 조건문과 반복문 실습

### ■ 십진수를 이진수로 변환

- 십진수를 이진수로 변환하는 코드

코드 4-18 decimal.py

```
1 decimal = 10
2 result = ''
3 while (decimal > 0):
4     remainder = decimal % 2
5     decimal = decimal // 2
6     result = str(remainder) + result
7 print(result)
```

1010

## 03. 조건문과 반복문 실습

### ■ 문제 해결

코드 4-19 guess\_number.py

```
1 import random                                # 난수 발생 함수 호출
2 guess_number = random.randint(1, 100)        # 1~100 사이 정수 난수 발생
3 print("숫자를 맞춰 보세요. (1 ~ 100)")
4 users_input = int(input())                   # 사용자 입력을 받음
5 while (users_input is not guess_number):     # 사용자 입력과 난수가 같은지 판단
6     if users_input > guess_number:           # 사용자 입력이 클 경우
7         print("숫자가 너무 큼니다.")
8     else:                                    # 사용자 입력이 작을 경우
9         print("숫자가 너무 작습니다.")
10    users_input = int(input())                # 다시 사용자 입력을 받음
11 else:
12    print("정답입니다.", "입력한 숫자는", users_input, "입니다.") # 종료 조건
```

## 03. 조건문과 반복문 실습

### ■ 실습 내용 : 연속적인 구구단 계산기

- 지금까지 배운 반복문과 조건문을 토대로 연속적인 구구단 계산기 프로그램을 만들어 보자.
- 이 프로그램의 규칙은 다음과 같다.

- 프로그램이 시작되면 '구구단 몇 단을 계산할까요(1~9)?'가 출력된다.
- 사용자는 계산하고 싶은 구구단 숫자를 입력한다.
- 프로그램은 '구구단 n단을 계산합니다.'라는 메시지와 함께 구구단의 결과를 출력한다.
- 기존 문제와 달리, 이번에는 프로그램이 계속 실행되다가 종료 조건에 해당하는 숫자(여기에서는 0)를 입력하면 종료된다.

## 03. 조건문과 반복문 실습

### ■ 실행 결과

구구단 몇 단을 계산할까요(1~9)?

3

구구단 3단을 계산합니다.

$$3 \times 1 = 3$$

$$3 \times 2 = 6$$

$$3 \times 3 = 9$$

$$3 \times 4 = 12$$

$$3 \times 5 = 15$$

$$3 \times 6 = 18$$

$$3 \times 7 = 21$$

$$3 \times 8 = 24$$

$$3 \times 9 = 27$$

구구단 몇 단을 계산할까요(1~9)?

5

구구단 5단을 계산합니다.

## 03. 조건문과 반복문 실습

### ■ 실행 결과

$$5 \times 1 = 5$$

$$5 \times 2 = 10$$

$$5 \times 3 = 15$$

$$5 \times 4 = 20$$

$$5 \times 5 = 25$$

$$5 \times 6 = 30$$

$$5 \times 7 = 35$$

$$5 \times 8 = 40$$

$$5 \times 9 = 45$$

구구단 몇 단을 계산할까요(1~9)?

0

구구단 게임을 종료합니다.

## 03. 조건문과 반복문 실습

### ■ 문제 해결

코드 4-20 calculator2.py

```
1 print("구구단 몇 단을 계산할까요(1~9)?")
2 x = 1
3 while (x is not 0):
4     x = int(input())
5     if x == 0: break
6     if not(1 <= x <= 9):
7         print("잘못 입력했습니다", "1부터 9 사이 숫자를 입력하세요.")
8         continue
9     else:
10        print("구구단 " + str(x) + "단을 계산합니다.")
11        for i in range(1,10):
12            print(str(x) + " x " + str(i) + " = " + str(x*i))
13        print("구구단 몇 단을 계산할까요(1~9)?")
14 print("구구단 게임을 종료합니다.")
```

## 03. 조건문과 반복문 실습

### ■ 실습 내용 : 평균 구하기

- 이번 Lab에서는 지금까지 배운 반복문과 조건문을 토대로 연속적인 평균 구하기 프로그램을 만들어 보자.

학생	A	B	C	D	E
국어 점수	49	80	20	100	80
수학 점수	43	60	85	30	90
영어 점수	49	82	48	50	100

- 이 프로그램의 규칙은 다음과 같다.

- 이차원 리스트이므로 각 행을 호출하고 각 열에 있는 값을 더해 학생별 평균을 구한다.
- for문 2개를 사용한다.

- 실행 결과

```
[47.0, 74.0, 51.0, 60.0, 90.0]
```

## 03. 조건문과 반복문 실습

### ■ 문제 해결

코드 4-21 average.py

```
1 kor_score = [49, 80, 20, 100, 80]
2 math_score = [43, 60, 85, 30, 90]
3 eng_score = [49, 82, 48, 50, 100]
4 midterm_score = [kor_score, math_score, eng_score]
5
6 student_score = [0, 0, 0, 0, 0]
7 i = 0
8 for subject in midterm_score:
9     for score in subject:
10         student_score[i] += score        # 학생마다 개별로 교과 점수를 저장
11         i += 1                          # 학생 인덱스 구분
12     i = 0                                # 과목이 바뀔 때 학생 인덱스 초기화
13 else:
14     a, b, c, d, e = student_score        # 학생별 점수를 언패킹
15     student_average = [a/3, b/3, c/3, d/3, e/3]
16     print(student_average)
```



### 03. 조건문과 반복문 실습

#### ■ 문제 해결

	A	B	C	D	E
국어 점수	49	80	20	100	80
수학 점수	43	60	85	30	90
영어 점수	49	82	48	50	100

student\_score[0]

student\_score[1]

student\_score[2]

student\_score[3]

student\_score[4]

[student\_score[i]]

## 04. 코드의 오류를 처리하는 방법

---

### ■ 버그와 디버그

- **버그(bug)** : 프로그래밍에서의 오류
- **디버그(debug)** : 오류를 수정하는 과정
- **디버깅(debugging)** : 코드에서 오류를 만났을 때, 프로그램의 잘못을 찾아내고 고치는 것

## 04. 코드의 오류를 처리하는 방법

---

### ■ 오류의 종류와 해결 방법 : 문법적 오류

- 문법적 오류는 코딩했을 때, 인터프리터가 해석을 못 해 코드 자체를 실행시키지 못하는 오류이다. 문법적 오류는 비교적 쉬운 유형의 오류이며, 대표적으로 들여쓰기 오류와 오타자로 인한 오류가 있다.

## 04. 코드의 오류를 처리하는 방법

### ■ 오류의 종류와 해결 방법 : 문법적 오류

- 들여쓰기 오류(indentation error)

코드 4-22 indentation.py

```
1 x = 2
2 y = 5
3 print(x + y)
```

```
D:\workspace\Ch04>python indentation.py
File "indentation.py", line 2
  y = 5
  ^
IndentationError: unexpected indent
```

[들여쓰기 오류 메시지]

## 04. 코드의 오류를 처리하는 방법

### ■ 오류의 종류와 해결 방법 : 문법적 오류

- 오타자로 인한 오류

코드 4-23 name.py

```
1 pront (x + y)      # Print가 아닌 Pront로 작성
2 korean = "ACE"
3 print(Korean)      # k는 소문자
```

```
D:\workspace\Ch04>python name.py
Traceback (most recent call last):
  File "name.py", line 1, in <module>
    pront (x + y) # Print가 아닌 Pront로 작성
NameError: name 'pront' is not defined
```

[오타자로 인한 오류 메시지]

## 04. 코드의 오류를 처리하는 방법

---

### ■ 오류의 종류와 해결 방법 : 논리적 오류

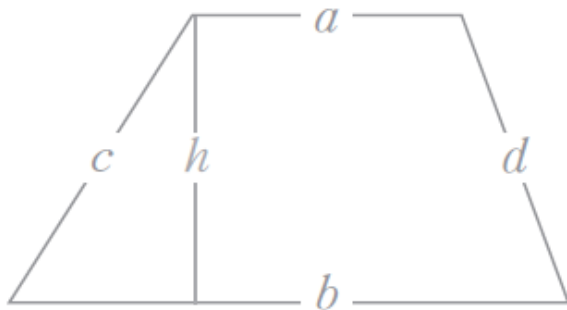
- 프로그램을 작성하다 보면 코드를 제대로 작성했다고 생각했음에도, 원하는 결과가 나오지 않는 경우가 종종 있다.
- 논리적 오류를 해결하는 방법은 다양한데, 당장 쉽게 사용할 수 있는 방법은 확인이 필요한 변수들에 `print( )` 함수를 사용하여 값을 확인하는 것이다.

## 04. 코드의 오류를 처리하는 방법

### ■ 오류의 종류와 해결 방법

- 사다리꼴 넓이를 구하는 프로그램을 작성하면서, 이 논리적 오류를 해결하는 연습을 해 보자.

$$A = \frac{a + b}{2} h \quad (a: \text{윗변}, b: \text{아랫변}, h: \text{높이})$$



[사다리꼴의 넓이 구하기]

## 04. 코드의 오류를 처리하는 방법

### ■ 오류의 종류와 해결 방법

- 사다리꼴의 넓이를 구하는 공식은 ' $\{(\text{밑변} + \text{윗변}) / 2\} * \text{높이}$ '이다. 이 수식에 있는 각각의 과정을 하나씩 함수로 만들어 변환하는 연습을 할 것이다. 첫 번째는 두 변수를 더하는 `addition()` 함수, 두 번째는 두 값을 곱하는 `multiplication()` 함수, 세 번째는 2로 나누는 `divided()` 함수이다.

```
1 def addition(x, y):  
2     return x + y  
3  
4 def multiplication(x, y):  
5     return x * y  
6  
7 def divided_by_2(x):  
8     return x / 2
```



## 04. 코드의 오류를 처리하는 방법

### ■ 오류의 종류와 해결 방법

- ① 함수가 잘 작동하는지 확인하는 방법 : 파이썬 셸에서 실행하기

```
>>> import trapezium_def as ta          # trapezium_def 파일을 ta라는 이름으로 부름
>>> ta.addition(10, 5)
15
>>> ta.multiplication(10, 5)
50
>>> ta.divided_by_2(50)
25.0
```

## 04. 코드의 오류를 처리하는 방법

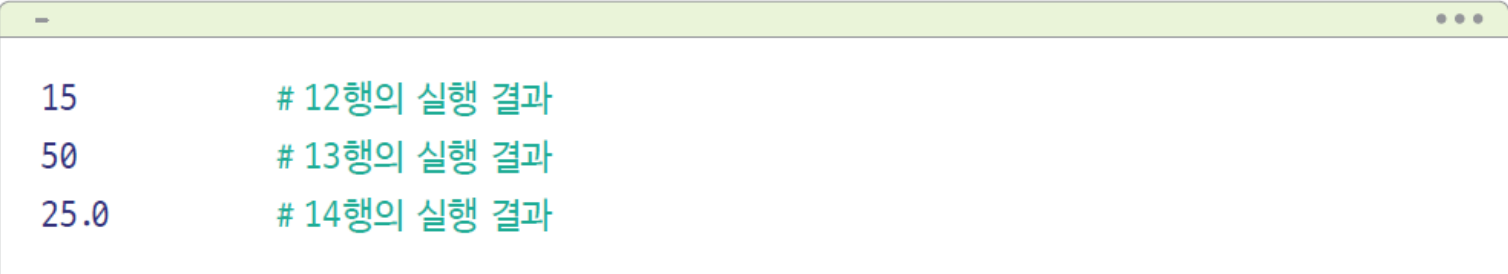
### ■ 오류의 종류와 해결 방법

- ② 함수가 잘 작동하는지 확인하는 방법 : 파일에서 체크할 수 있도록 if `_name_ == "_main_"`: 을 써 주는 구조로, if 문 안에 테스트할 코드를 작성하기

```
1 def addition(x, y):
2     return x + y
3
4 def multiplication(x, y):
5     return x * y
6
7 def divided_by_2(x):
8     return x / 2
9
10 # 파이썬 셸에서 호출할 경우 실행되지 않음
11 if __name__ == '__main__':
12     print(addition(10, 5))
13     print(multiplication(10, 5))
14     print(divided_by_2(50))
```

## 04. 코드의 오류를 처리하는 방법

### ■ 오류의 종류와 해결 방법



```
15          # 12행의 실행 결과
50          # 13행의 실행 결과
25.0        # 14행의 실행 결과
```

➡ if `__name__ == '__main__'`을 넣는 가장 큰 이유는 해당 파일을 파이썬 셸에서 불러올 import 때 함수 안에 들어 있지 않은 코드들이 작동되지 않게 하기 위해서이다. 만약 해당 구문 없이 `print()` 함수로 구문을 작성한다면, 해당 파일을 파이썬 셸에서 호출할 때 그 구문이 화면에 출력되는 것을 확인할 수 있다.

따라서 어떤 것을 테스트하기 위해서는 반드시 if `__name__ == '__main__'` 안에 코드를 넣는 것이 좋다.

## 04. 코드의 오류를 처리하는 방법

### ■ 오류의 종류와 해결 방법

- 실제 사다리꼴의 넓이 구하기 프로그램을 작성

```
1 def addition(x, y):
2     return x + y
3
4 def divided_by_2(x):
5     return x / 2
6
7 def main():
8     base_line = float(input("밑변의 길이는? "))
9     upper_edge = float(input("윗변의 길이는? "))
10    height = float(input("높이는? "))
11
12    print("넓이는:", divided_by_2(addition(base_line, upper_edge) * height))
13
14 if __name__ == '__main__':
15     main()
```

## 04. 코드의 오류를 처리하는 방법

### ■ 오류의 종류와 해결 방법

- 실제 사다리꼴의 넓이 구하기 프로그램을 작성



```
밑변의 길이는? 5
```

```
윗변의 길이는? 4
```

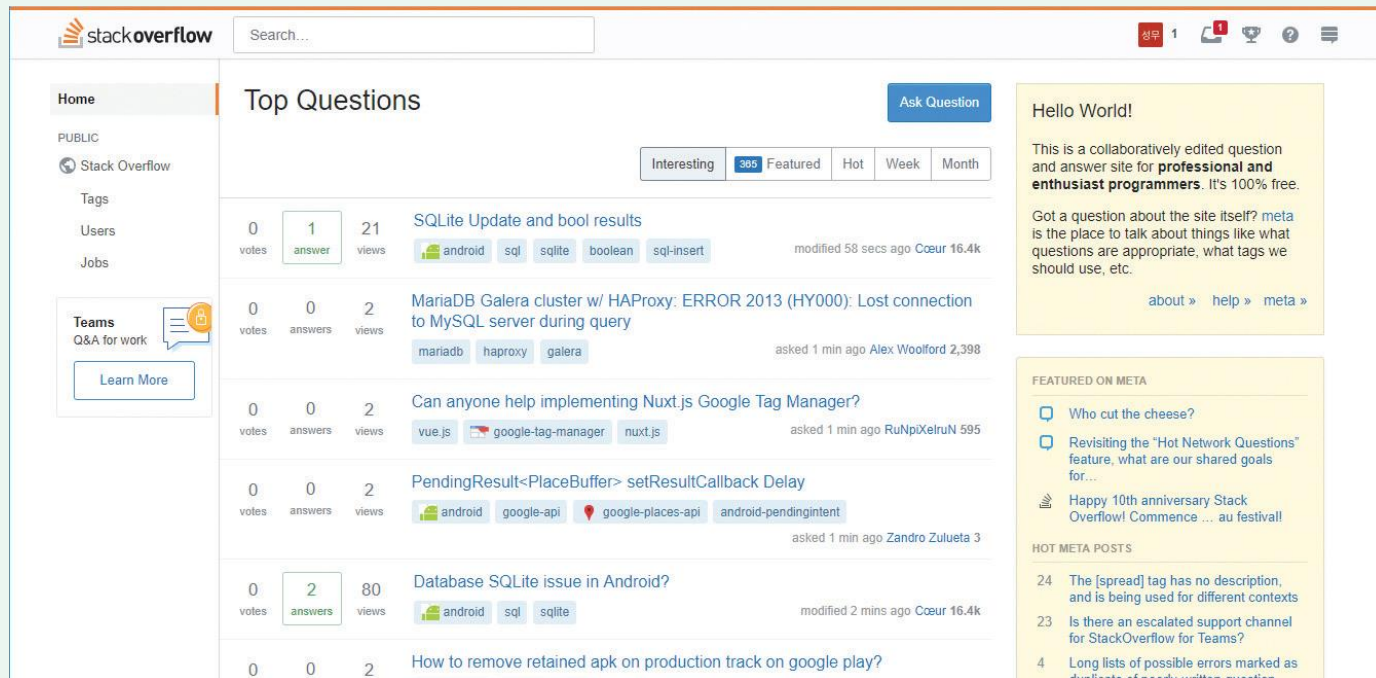
```
높이는? 3
```

```
넓이는: 13.5
```

## 04. 코드의 오류를 처리하는 방법

### 여기서 잠깐! 오류를 스스로 해결하기

- 많은 사람이 구글과 프로그래밍계의 지식인인 Stack Overflow를 통해 문제를 해결하고 있다. Stack Overflow는 전 세계 개발자들이 사용하는 Q&A 사이트이다.



The screenshot shows the Stack Overflow homepage. The top navigation bar includes the Stack Overflow logo, a search bar, and user avatars. The left sidebar contains links for Home, PUBLIC, Stack Overflow, Tags, Users, Jobs, Teams, and a Learn More button. The main content area displays 'Top Questions' with a list of questions, each showing votes, answers, views, tags, and the time since it was asked. The right sidebar features a 'Hello World!' message, a 'FEATURED ON META' section, and a 'HOT META POSTS' section.

votes	answers	views	question	tags	time
0	1	21	SQLite Update and bool results	android, sql, sqlite, boolean, sql-insert	modified 58 secs ago Cœur 16.4k
0	0	2	MariaDB Galera cluster w/ HAProxy: ERROR 2013 (HY000): Lost connection to MySQL server during query	mariadb, haproxy, galera	asked 1 min ago Alex Woolford 2,398
0	0	2	Can anyone help implementing Nuxt.js Google Tag Manager?	vue.js, google-tag-manager, nuxt.js	asked 1 min ago RuNpXelruN 595
0	0	2	PendingResult<PlaceBuffer> setResultCallback Delay	android, google-api, google-places-api, android-pendingintent	asked 1 min ago Zandro Zulueta 3
0	2	80	Database SQLite issue in Android?	android, sql, sqlite	modified 2 mins ago Cœur 16.4k
0	0	2	How to remove retained apk on production track on google play?		

[Stack Overflow(<https://stackoverflow.com>)]