

Day 6 - Control Statement (Loop)

while 문:

```
i = 1
while i <= 5:
    print('Hello Python')
    i += 1
```

⇒ Hello Python
Hello Python
Hello Python
Hello Python
Hello Python

- 1 ~ 5까지는 True
- Hello Python 출력 * 5

ex) 1부터 10까지의 합:

```
[6] # 1부터 10까지의 총합
i = 1
sum = 0

while i <= 10:
    sum += i
    i += 1

print(f'1부터 10까지의 합: {sum}')
```

⇒ 1부터 10까지의 합: 55

ex) 구구단:

```
# 원하는 구구단을 입력받아 해당 단의 구구단을 출력
dan = int(input('원하는 단을 입력하세요: '))
print(f'{dan} 단')

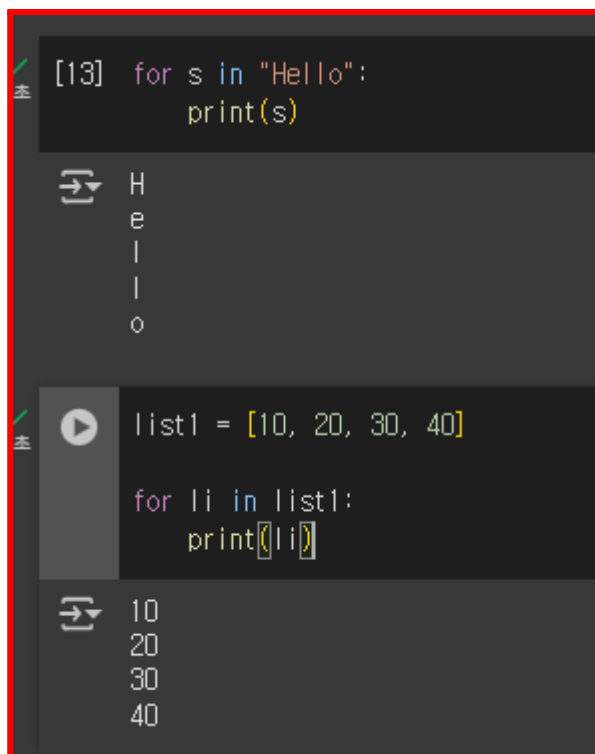
i = 1
while i <= 9:
    print(f'{dan} * {i} = {dan * i}')
    i += 1
```

⇒ 원하는 단을 입력하세요: 3
3 단
3 * 1 = 3
3 * 2 = 6
3 * 3 = 9
3 * 4 = 12
3 * 5 = 15
3 * 6 = 18
3 * 7 = 21
3 * 8 = 24
3 * 9 = 27

For문:

```
# 요소: 반복할 시퀀스의 각 항목이 for 문의 실행 도중에 할당되는 변수
for 요소 in 시퀀스:
    # 요소에 대한 작업 수행
```

- 지정된 범위 내에서 반복할때 사용
- 시퀀스(리스트, 튜플, 문자열 등)의 각 항목에 대해 반복 작업을 수행하는 반복문
 - **시퀀스 : 반복할 수 있는 객체**
- 요소는 시퀀스에 있는 요소를 하나씩 꺼내서 저장해준다.
- 순서가 있는 객체(모두)는 무조건 반복할 수 있다(iterable 하다).
 - 무조건 index가 있다 (순서를 뽑을 수 있다)



```
[13] for s in "Hello":
      print(s)

H
e
l
l
o

list1 = [10, 20, 30, 40]

for li in list1:
    print(li)

10
20
30
40
```

- 이런식으로 반복해서 요소를 하나씩 꺼내온다.

range() 함수:

```
range([start], stop, [step])
```

- start (선택 사항): 순차적인 범위의 시작 값을 지정합니다. 기본값은 0입니다.
- stop: 순차적인 범위의 끝 값을 지정합니다. 생성된 시퀀스는 stop 값 직전까지의 정수를 포함합니다.
- step (선택 사항): 순차적인 값을 증가시키는 간격을 지정합니다. 기본값은 1입니다.

ex)

```
num = range(0, 10, 1)
# print(num)
for i in num:
    print(i, end=' ')
```

0, 1, 2, 3, 4, 5, 6, 7, 8, 9

range(0, 10)

- 이런식으로 0 부터 9까지 찍힌다.

Memory management in range:

```
for i in range(0, 10, 1):
    print(i, end=' ')
```

0 1 2 3 4 5 6 7 8 9

- i가 range(heap)을 가리킨다
- 블록 전체가 출력완료되었을 경우, range는 i가 더이상 가르키지 않음으로 메모리상에서 자동으로 지워진다.

enumerate() 함수

```
enumerate(iterable, [start=0])
```

- 반복문을 사용할때 인덱스 + 값을 가져오기 위해 사용
- 주로 for문과 사용
- 순서추적 + 값과 인덱스 함께 사용
- 튜플을 반환 (인덱스, 값)
- [start = 0] 은 옵션이라는 뜻이다.

ex)

```
for e in enumerate('hello', 0):  
    print(e)  
  
(0, 'h')  
(1, 'e')  
(2, 'l')  
(3, 'l')  
(4, 'o')
```

- (index, value) - Tuple

Iterable:

1. 이터러블(iterable)

이터러블은 for문처럼 반복문에서 사용할 수 있는 모든 객체를 의미합니다. 이터러블한 객체는 for item in 객체: 구조에서 사용 가능합니다. 모든 순서 있는 컬렉션은 이터러블입니다. 하지만 모든 이터러블이 순서 있는 것은 아닙니다. (set, dict.keys())는 반복은 되지만 인덱스가 없음)

- 모든 이터러블이 순서가 있는건 아니다 (set, dict.key())

Iterator:

2. 이터레이터(Iterator)

이터러블 객체는 iter() 함수를 사용해서 이터레이터로 바꿀 수 있습니다. 또한 next()를 사용해서 값을 하나씩 꺼낼 수 있습니다.

이터러블	iter()를 적용할 수 있는 객체	리스트, 튜플, 문자열 등
이터레이터	next()로 값을 꺼낼 수 있는 객체	iter(리스트)로 만든 것

- next() 로 다음 값을 꺼낼 수 있다.

enumerate() in list:

```
list1 = [10, 20, 30, 40]

for e in enumerate(list1):
    print(e)
```

```
(0, 10)
(1, 20)
(2, 30)
(3, 40)
```

Save both index and value in variable:

```
# i, v = (0, 10)
for i, v in enumerate(list1):
    print(f'인덱스:{i}, 값:{v}')
```

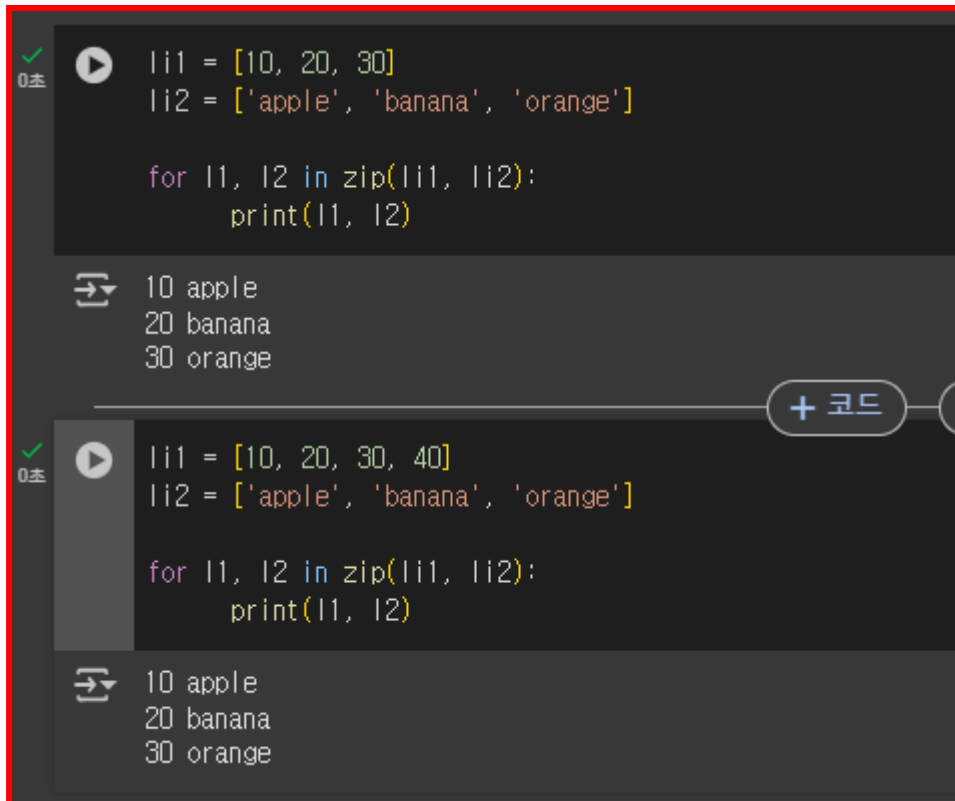
```
인덱스:0, 값:10
인덱스:1, 값:20
인덱스:2, 값:30
인덱스:3, 값:40
```

Zip 함수:

```
zip(iterable1, iterable2, ...)
```

- 반복 가능한(iterable) 객체를 한꺼번에 출력해주는 함수
- 여러 개의 리스트나 튜플을 병렬적으로 처리하고자 할때 사용

ex)



```
li1 = [10, 20, 30]
li2 = ['apple', 'banana', 'orange']

for l1, l2 in zip(li1, li2):
    print(l1, l2)
```

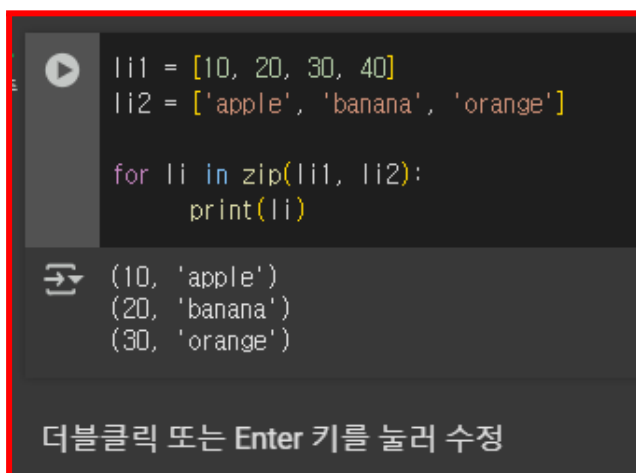
10 apple
20 banana
30 orange

```
li1 = [10, 20, 30, 40]
li2 = ['apple', 'banana', 'orange']

for l1, l2 in zip(li1, li2):
    print(l1, l2)
```

10 apple
20 banana
30 orange

- 두 리스트가 같은 length가 아니면 반복을 하지 않는다
- 따라서 밑에는 3바퀴까지밖에 돌지 않는다.



```
li1 = [10, 20, 30, 40]
li2 = ['apple', 'banana', 'orange']

for li in zip(li1, li2):
    print(li)
```

(10, 'apple')
(20, 'banana')
(30, 'orange')

더블클릭 또는 Enter 키를 눌러 수정

- enumerate 처럼 튜플로도 만들어줄 수 있다.

중첩 반복문:

```
for 외부_변수 in 외부_시퀀스:
    for 내부_변수 in 내부_시퀀스:
        # 내부 반복문 코드
    # 외부 반복문 코드
```

- 내부 반복문이 다 돌고 외부 반복문이 돈다.

ex)

```
for i in range(1, 4):
    print(f'😎 i: {i}')
    for j in range(1, 4):
        print(f'😍 i: {j}')
```

Handwritten annotations in the image: A purple bracket above the first loop's range(1, 4) is labeled '1 2 3'. To the right of the first loop's print statement is '1, 2, 3'. To the right of the second loop's print statement is '1, 2, 3'. A purple arrow points from the first loop's range to the second loop's range.

- output:

```
😎 i: 1
😍 i: 1
😍 i: 2
😍 i: 3
😎 i: 2
😍 i: 1
😍 i: 2
😍 i: 3
😎 i: 3
😍 i: 1
😍 i: 2
😍 i: 3
```