세상의 속도를
따라잡고 싶다면

점프 투 파이썬

박응용 지음(위키독스 운영자)

이지스퍼블리싱

03

프로그램의 구조를 쌓는다! 제어문

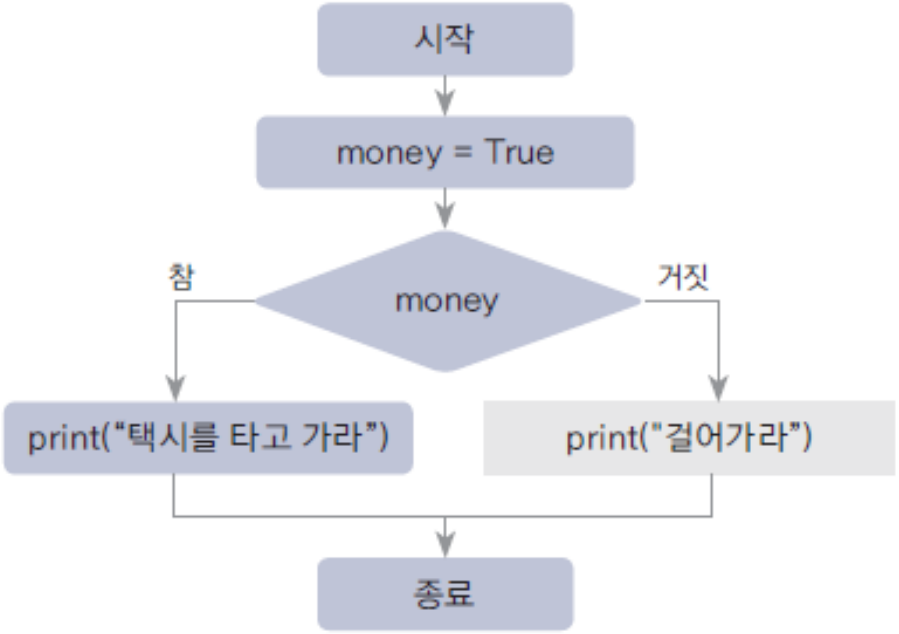
03-1 if 문

if 문은 왜 필요할까?

- 주어진 조건을 판단한 후 그 상황에 맞게 처리해야 할 경우

'돈이 있으면 택시를 타고 가고, 돈이 없으면 걸어간다.'

```
>>> money = True
>>> if money:
...     print("택시를 타고 가라")
... else:
...     print("걸어가라")
...
택시를 타고 가라
```

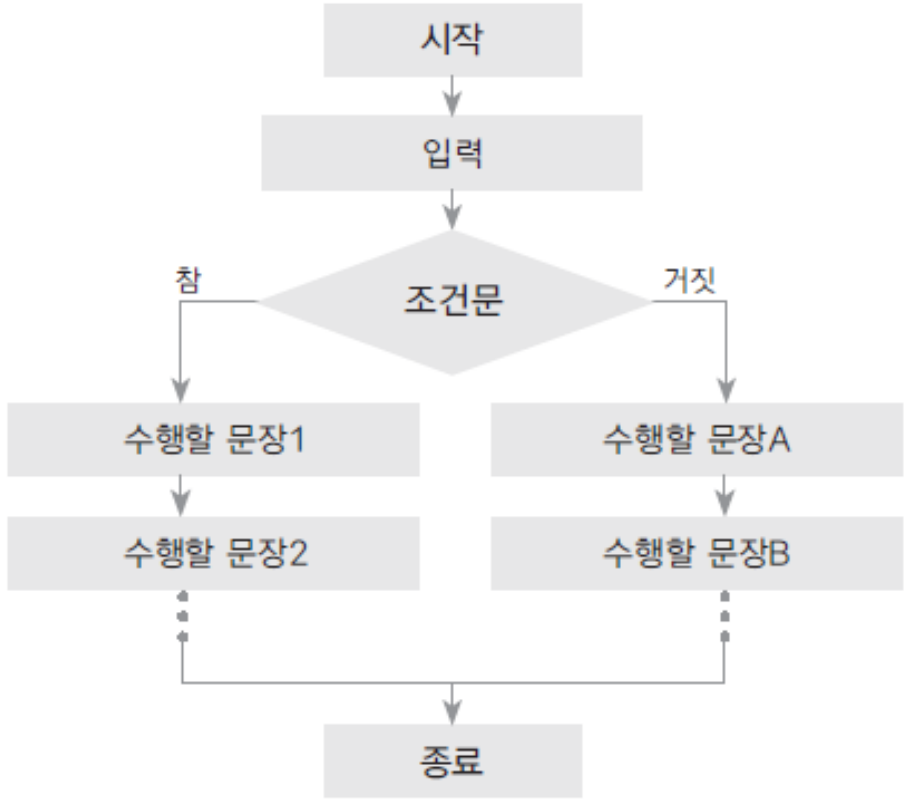


03-1 if 문

▪ if 문의 기본 구조

- if와 else를 사용한 조건문의 기본 구조

```
if 조건문:
    수행할_문장1
    수행할_문장2
        ⋮
else:
    수행할_문장A
    수행할_문장B
        ⋮
```



- 조건문이 참이면 if 블록 수행
- 조건문이 거짓이면 else 블록 수행

03-1 if 문

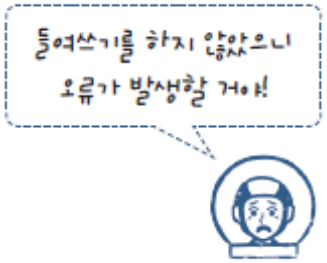
▪ 들여쓰기 방법 알아보기

- if 문을 만들 때는 if 조건문 바로 다음 문장부터 모든 문장에 들여쓰기(indentation)

```
if 조건문:
    수행할_문장1
    수행할_문장2
    수행할_문장3
```

- 들여쓰기를 무시하는 경우 오류 발생

```
if 조건문:
    수행할_문장1
수행할_문장2
    수행할_문장3
```




```
if 조건문:
    수행할_문장1
    수행할_문장2
        수행할_문장3
```

03-1 if 문

조건문이란 무엇인가?

- if 조건문에서 '조건문'이란 참과 거짓을 판단하는 문장

```
>>> money = True
>>> if money:
```

- 비교 연산자

비교 연산자	설명
x < y	x가 y보다 작다.
x > y	x가 y보다 크다.
x == y	x와 y가 같다.
x != y	x와 y가 같지 않다.
x >= y	x가 y보다 크거나 같다.
x <= y	x가 y보다 작거나 같다.

```
>>> x = 3
>>> y = 2
>>> x > y  <-- 3 > 2
True
```

```
>>> x < y  <-- 3 < 2
False
```

```
>>> x == y  <-- 3 == 2
False
```

```
>>> x != y  <-- 3 != 2
True
```

03-1 if 문

조건문이란 무엇인가?

- 비교 연산자
  - if 조건문에 비교 연산자를 사용하는 예시

> 만약 3000원 이상의 돈을 가지고 있으면 택시를 타고 가고, 그렇지 않으면 걸어가라.

```
>>> money = 2000  <-- 2,000원을 가지고 있다고 설정
>>> if money >= 3000:
...     print("택시를 타고 가라")
... else:
...     print("걸어가라")
...
걸어가라
```

03-1 if 문

▪ 조건문이란 무엇인가?

▪ and, or, not

연산자	설명
x or y	x와 y 둘 중 하나만 참이어도 참이다.
x and y	x와 y 모두 참이어야 참이다.
not x	x가 거짓이면 참이다.

▪ or 연산자의 사용법

돈이 3000원 이상 있거나 카드가 있다면 택시를 타고 가고, 그렇지 않으면 걸어가라.

```
>>> money = 2000  <-- 2,000원을 가지고 있다고 설정
>>> card = True  <-- 카드를 가지고 있다고 설정
>>> if money >= 3000 or card:
...     print("택시를 타고 가라")
... else:
...     print("걸어가라")
...
택시를 타고 가라
```

03-1 if 문

조건문이란 무엇인가?

in, not in

in	not in
x in 리스트	x not in 리스트
x in 튜플	x not in 튜플
x in 문자열	x not in 문자열

```
>>> 1 in [1, 2, 3] <— 1이 [1, 2, 3] 안에 있는가?
True
>>> 1 not in [1, 2, 3] <— 1이 [1, 2, 3] 안에 없는가?
False
```

```
>>> 'a' in ('a', 'b', 'c')
True
>>> 'j' not in 'python'
True
```

03-1 if 문

▪ 다양한 조건을 판단하는 elif

- if와 else만으로는
  조건 판단에 어려움이 있음

> 주머니에 돈이 있으면 택시를 타고,
> 주머니에 돈은 없지만 카드가 있으면 택시를 타고,
> 돈도 없고 카드도 없으면 걸어 가라.

- 조건 판단하는 부분
  1) 주머니에 돈이 있는지 판단
  2) 주머니에 돈이 없으면,
     주머니에 카드가 있는지 판단

```
>>> pocket = ['paper', 'cellphone']  <-- 주머니 안에 종이, 휴대전화가 있다.
>>> card = True  <-- 카드를 가지고 있다.
>>> if 'money' in pocket:
...     print("택시를 타고 가라")
... else:
...     if card:
...         print("택시를 타고 가라")
...     else:
...         print("걸어가라")
...
택시를 타고 가라
```

- if와 else만으로는 이해하기 어렵고 산만한 느낌

03-1 if 문

다양한 조건을 판단하는 elif

- elif를 사용한다면?

```
>>> pocket = ['paper', 'cellphone']
>>> card = True
>>> if 'money' in pocket: <-- 주머니에 돈이 있으면
...     print("택시를 타고 가라")
... elif card: <-- 주머니에 돈이 없고 카드가 있으면
...     print("택시를 타고 가라")
... else: <-- 주머니에 돈이 없고 카드도 없으면
...     print("걸어가라")
...
택시를 타고 가라
```

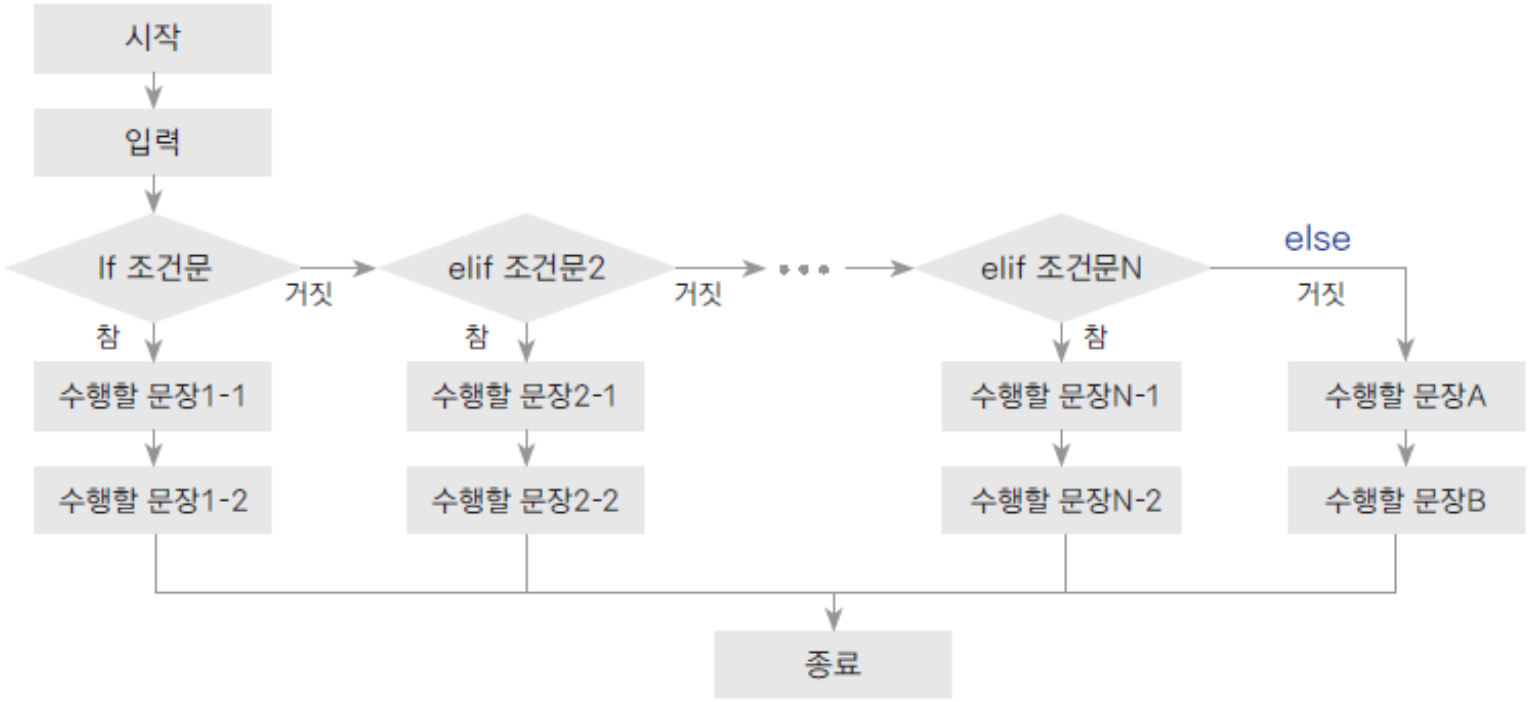
- elif는 이전 조건문이 거짓일 때 수행됨

```
if 조건문:
    수행할_문장1
    수행할_문장2
    ...
elif 조건문:
    수행할_문장1
    수행할_문장2
    ...
elif 조건문:
    수행할_문장1
    수행할_문장2
    ...
(...생략...)
else:
   수행할_문장1
   수행할_문장2
   ...
```

03-1 if 문

▪ 다양한 조건을 판단하는 elif

- elif는 개수에 제한 없이 사용 가능



03-1 if 문

▪ 조건부 표현식

- score가 60 이상일 경우 message에 문자열 "success", 아닐 경우에 문자열 "failure" 대입하는 코드

```
if score >= 60:
    message = "success"
else:
    message = "failure"
```

- 파이썬의 조건부 표현식(conditional expression) 사용

```
message = "success" if score >= 60 else "failure"
```

- 조건부 표현식

```
변수 = 조건문이_참인_경우의_값 if 조건문 else 조건문이_거짓인_경우의_값
```

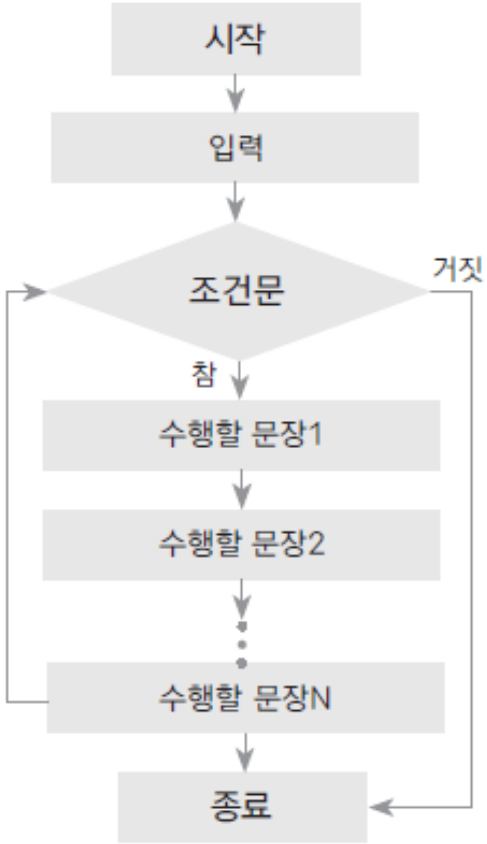
03-2 while 문

while 문의 기본 구조

- 반복해서 문장을 수행해야 할 경우 while 문 사용
- 반복문이라고도 부름

```
while 조건문:
    수행할_문장1
    수행할_문장2
    수행할_문장3
    ...
```

- while 문은 조건문이 참인 동안에
  while 문에 속한 문장이 반복해서 수행됨



03-2 while 문

while 문의 기본 구조

- 예제) '열 번 찍어 안 넘어가는 나무 없다'는 속담 구현
  - while 문의 조건문은 treeHit < 10
  - treeHit이 10보다 작은 동안 while 문에 포함된 문장 반복 수행

```
>>> treeHit = 0  <-- 나무를 찍은 횟수
>>> while treeHit < 10:  <-- 나무를 찍은 횟수가 10보다 작은 동안 반복
...     treeHit = treeHit + 1  <-- 나무를 찍은 횟수 1씩 증가
...     print("나무를 %d번 찍었습니다." % treeHit)
...     if treeHit == 10:  <-- 나무를 열 번 찍으면
...         print("나무 넘어갑니다.")
...
나무를 1번 찍었습니다.
나무를 2번 찍었습니다.
나무를 3번 찍었습니다.
나무를 4번 찍었습니다.
나무를 5번 찍었습니다.
나무를 6번 찍었습니다.
나무를 7번 찍었습니다.
나무를 8번 찍었습니다.
나무를 9번 찍었습니다.
나무를 10번 찍었습니다.
나무 넘어갑니다.
```

03-2 while 문

while 문의 기본 구조

- 예제) '열 번 찍어 안 넘어가는 나무 없다'는 속담 구현
  - while 문이 반복되는 과정

treeHit	조건문	조건 판단	수행하는 문장	while 문
0	0 < 10	참	나무를 1번 찍었습니다.	반복
1	1 < 10	참	나무를 2번 찍었습니다.	반복
2	2 < 10	참	나무를 3번 찍었습니다.	반복
3	3 < 10	참	나무를 4번 찍었습니다.	반복
4	4 < 10	참	나무를 5번 찍었습니다.	반복
5	5 < 10	참	나무를 6번 찍었습니다.	반복
6	6 < 10	참	나무를 7번 찍었습니다.	반복
7	7 < 10	참	나무를 8번 찍었습니다.	반복
8	8 < 10	참	나무를 9번 찍었습니다.	반복
9	9 < 10	참	나무를 10번 찍었습니다. 나무 넘어갑니다.	반복
10	10 < 10	거짓		종료

03-2 while 문

while 문 만들기

- 예제) 여러 가지 선택지 중 하나를 선택해 입력받기

```
>>> prompt = """
... 1. Add
... 2. Del
... 3. List
... 4. Quit
...
... Enter number: """
```

- number 변수에 0 대입하기

```
>>> number = 0  <- 번호를 입력받을 변수
>>> while number != 4:  <- 입력받은 번호가 4가 아닌 동안 반복
...     print(prompt)
...     number = int(input())
...
1. Add
2. Del
3. List
4. Quit

Enter number:
```

변수 prompt 출력

03-2 while 문

while 문 만들기

- 예제) 여러 가지 선택지 중 하나를 선택해 입력받기
  - number가 4가 아닌 동안 prompt를 출력

```
Enter number:
1  <-- 1 입력

1. Add
2. Del
3. List
4. Quit
```

4를 입력하지 않으면 계속 prompt의 값 출력

  - 사용자가 4를 입력하면 조건문이 거짓이 되어 while 문을 빠져나감

```
Enter number:
4    <-- 4 입력
>>>  <-- while 문 종료
```

03-2 while 문

while 문 강제로 빠져나가기

- 강제로 while 문을 빠져나가야 할 때 break 문 사용


```
>>> coffee = 10  <-- 자판기에 커피가 10개 있다.
>>> money = 300  <-- 자판기에 넣을 돈은 300원이다.
>>> while money:
...     print("돈을 받았으니 커피를 줍니다.")
...     coffee = coffee - 1  <-- while문을 한 번 돌 때마다 커피가 1개씩 줄어든다.
...     print("남은 커피의 양은 %d개입니다." % coffee)
...     if coffee == 0:
...         print("커피가 다 떨어졌습니다. 판매를 중지합니다.")
...         break
...
```

- money가 300으로 고정되어 있어, while문의 조건문은 항상 참 → 무한 루프
- break 문 호출 시 while 문 종료

03-2 while 문

while 문의 맨 처음으로 돌아가기

- while 문을 빠져나가지 않고 while 문의 맨 처음(조건문)으로 다시 돌아가야 할 때 사용
- 1부터 10까지의 숫자 중 홀수만 출력하는 예시
  - 조건문이 참이 되는 경우 → a가 짝수
  - continue 문장 수행 시 while 문의 맨 처음, 즉 조건문 a < 10으로 돌아감
  - 따라서 a가 짝수이면 print(a)는 수행되지 않음

```
>>> a = 0
>>> while a < 10:
...     a = a + 1
...     if a % 2 == 0: continue  <-- a를 2로 나누었을 때 나머지가 0이면 맨 처음으로 돌아간다.
...     print(a)
...
1
3
5
7
9
```

03-2 while 문

▪ 무한 루프

- 무한히 반복한다는 뜻의 무한 루프(endless loop)
- 파이썬에서의 무한 루프는 while 문으로 구현
  - while 문의 조건문이 True이므로 항상 참
    → while 문 안에 있는 문장들은 무한 수행

```
while True:
    수행할_문장1
    수행할_문장2
    ...
```

- 무한 루프 예

```
>>> while True:
...     print("Ctrl+C를 눌러야 while 문을 빠져나갈 수 있습니다.")
...
Ctrl+C를 눌러야 while 문을 빠져나갈 수 있습니다.
Ctrl+C를 눌러야 while 문을 빠져나갈 수 있습니다.
Ctrl+C를 눌러야 while 문을 빠져나갈 수 있습니다.
(...생략...)
```

03-3 for 문

for 문의 기본 구조

```
for 변수 in 리스트(또는 튜플, 문자열):
    수행할_문장1
    수행할_문장2
    ...
```

- 리스트나 튜플, 문자열의 첫 번째 요소부터 마지막 요소까지 차례로 변수에 대입되어 '수행할 문장1', '수행할 문장2' 등이 수행됨

03-3 for 문

▪ 예제를 통해 for 문 이해하기

1. 전형적인 for 문

```
>>> test_list = ['one', 'two', 'three']
>>> for i in test_list:  <-- one, two, three를 순서대로 i에 대입
...     print(i)
...
one
two
three
```

2. 다양한 for 문의 사용

```
>>> a = [(1, 2), (3, 4), (5, 6)]
>>> for (first, last) in a:
...     print(first + last)
...
3  <-- first: 1, last: 2
7  <-- first: 3, last: 4
11 <-- first: 5, last: 6
```

03-3 for 문

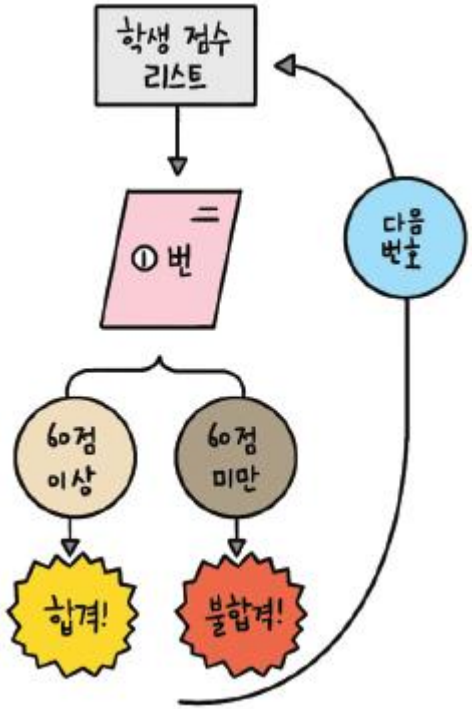
▪ 예제를 통해 for 문 이해하기

3. for 문의 응용

총 5명의 학생이 시험을 보았는데 시험 점수가 60점 이상이면 합격이고 그렇지 않으면 불합격이다. 합격인지, 불합격인지 결과를 보여 주시오.

```
marks = [90, 25, 67, 45, 80]          # 학생들의 시험 점수 리스트

number = 0                            # 학생에게 붙여 줄 번호
for mark in marks:                    # 90, 25, 67, 45, 80을 순서대로 mark에 대입
    number = number + 1
    if mark >= 60:
        print("%d번 학생은 합격입니다." % number)
    else:
        print("%d번 학생은 불합격입니다." % number)
```



03-3 for 문

for 문과 continue 문

- for 문 안의 문장을 수행하는 도중 continue 문을 만나면 for 문의 처음으로 돌아감
- 60점 이상인 사람에게는 축하 메시지를 보내고 나머지 사람에게는 아무런 메시지도 전하지 않는 프로그램

```
marks = [90, 25, 67, 45, 80]

number = 0
for mark in marks:
    number = number + 1
    if mark < 60:
        continue
    print("%d번 학생 축하합니다. 합격입니다. " % number)
```

03-3 for 문

for 문과 함께 자주 사용하는 range 함수

- 숫자 리스트를 자동으로 만들어주는 함수

```
>>> a = range(10)
>>> a
range(0, 10) <-- 0, 1, 2, 3, 4, 5, 6, 7, 8, 9
```

  - range(10)은 0부터 10 미만의 숫자를 포함하는 range 객체를 만들어 준다.

- range(a, b)
  - a: 시작 숫자
  - b: 끝 숫자 (반환 범위에 포함되지 않음)

```
>>> a = range(1, 11)
>>> a
range(1, 11) <-- 1, 2, 3, 4, 5, 6, 7, 8, 9, 10
```

03-3 for 문

for 문과 함께 자주 사용하는 range 함수

- range 함수의 예시
  - for와 range 함수를 사용하여 1부터 10까지 더하기

```
>>> add = 0
>>> for i in range(1, 11):
...     add = add + i
...
>>> print(add)
55
```

03-3 for 문

for 문과 함께 자주 사용하는 range 함수

- for와 range를 이용한 구구단
  - ①번 for문
    - 2부터 9까지의 숫자(range(2, 10))가 차례로 i에 대입됨
  - ②번 for문
    - 1부터 9까지의 숫자(range(1, 10))가 차례로 j에 대입됨
    - print(i*j) 수행

```
>>> for i in range(2, 10):          <- ①번 for 문
...     for j in range(1, 10):      <- ②번 for 문
...         print(i*j, end=" ")
...     print('')
...
2 4 6 8 10 12 14 16 18
3 6 9 12 15 18 21 24 27
4 8 12 16 20 24 28 32 36
5 10 15 20 25 30 35 40 45
6 12 18 24 30 36 42 48 54
7 14 21 28 35 42 49 56 63
8 16 24 32 40 48 56 64 72
9 18 27 36 45 54 63 72 81
```

03-3 for 문

for 문과 함께 자주 사용하는 range 함수

for와 range를 이용한 구구단

03-3 for 문

▪ 리스트 컴프리헨션(list comprehension) 사용하기

- 리스트 안에 for 문 포함하기
- 예제
  - a 리스트의 각 항목에 3을 곱한 결과를 result 리스트에 담기

```
>>> a = [1, 2, 3, 4]
>>> result = []
>>> for num in a:
...     result.append(num*3)
...
>>> print(result)
[3, 6, 9, 12]
```

- 리스트 컴프리헨션을 사용하도록 수정

```
>>> a = [1, 2, 3, 4]
>>> result = [num*3 for num in a]
>>> print(result)
[3, 6, 9, 12]
```

03-3 for 문

리스트 컴프리헨션(list comprehension) 사용하기

- 리스트 안에 for 문 포함하기

- 예제
  - 리스트 컴프리헨션 안에 'if 조건' 사용 가능
  - [1, 2, 3, 4] 중에서 짝수에만 3을 곱하여 담도록 수정

```
>>> a = [1, 2, 3, 4]
>>> result = [num*3 for num in a if num%2 == 0]
>>> print(result)
[6, 12]
```

03-3 for 문

▪ 리스트 컴프리헨션(list comprehension) 사용하기

▪ 리스트 컴프리헨션 문법

- ‘if 조건문’ 부분은 생략 가능

```
[표현식 for 항목 in 반복_가능_객체 if 조건문]
```

- for 문 여러 개 사용 가능

```
[표현식 for 항목1 in 반복_가능_객체1 if 조건문1
       for 항목2 in 반복_가능_객체2 if 조건문2
       ...
       for 항목n in 반복_가능_객체n if 조건문n]
```

03-3 for 문

▪ 리스트 컴프리헨션(list comprehension) 사용하기

- 구구단의 모든 결과를 리스트로 담는 리스트 컴프리헨션 사용 예제

```
>>> result = [x*y for x in range(2, 10)
...               for y in range(1, 10)]
>>> print(result)
[2, 4, 6, 8, 10, 12, 14, 16, 18, 3, 6, 9, 12, 15, 18, 21, 24, 27, 4, 8, 12, 16,
20, 24, 28, 32, 36, 5, 10, 15, 20, 25, 30, 35, 40, 45, 6, 12, 18, 24, 30, 36, 42
, 48, 54, 7, 14, 21, 28, 35, 42, 49, 56, 63, 8, 16, 24, 32, 40, 48, 56, 64, 72,
9, 18, 27, 36, 45, 54, 63, 72, 81]
```

감사합니다.