점프 투 파이썬

박응용 지음(위키독스 운영자)

이지스퍼블리싱

03

프로그램의 구조를 쌓는다! 제어문

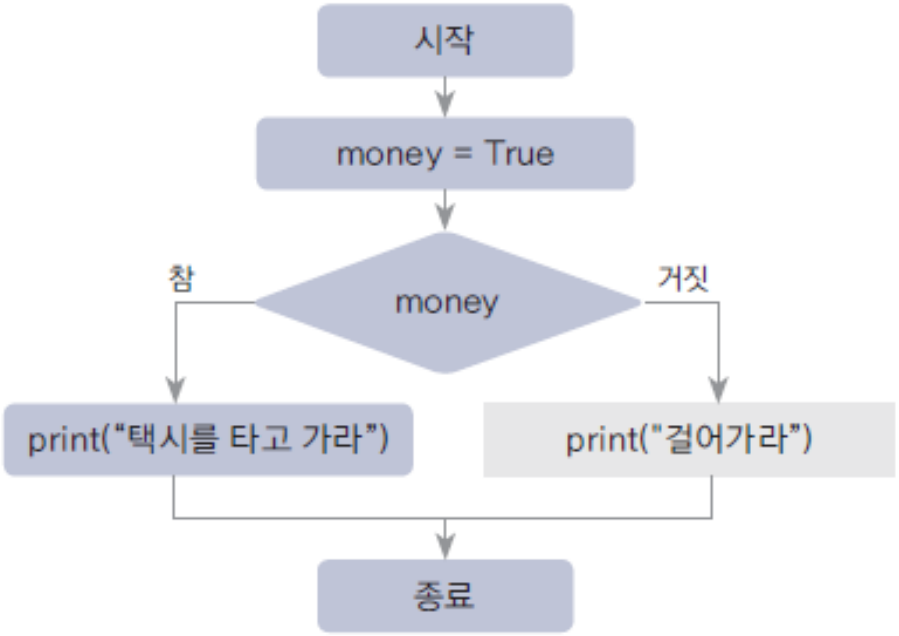
03-1 if 문

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
```

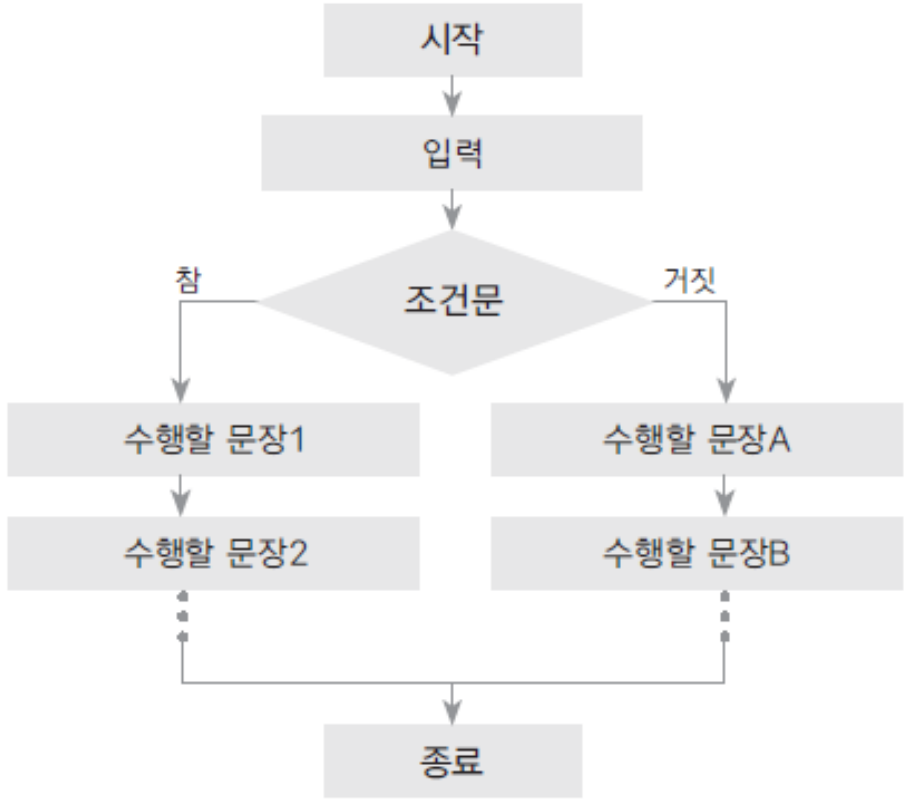


03-1 if 문

▪ 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

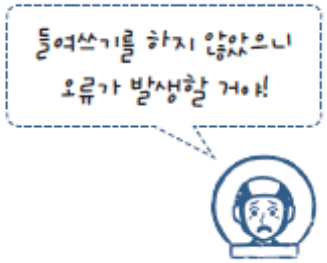
▪ 들여쓰기 방법 알아보기

- 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
```

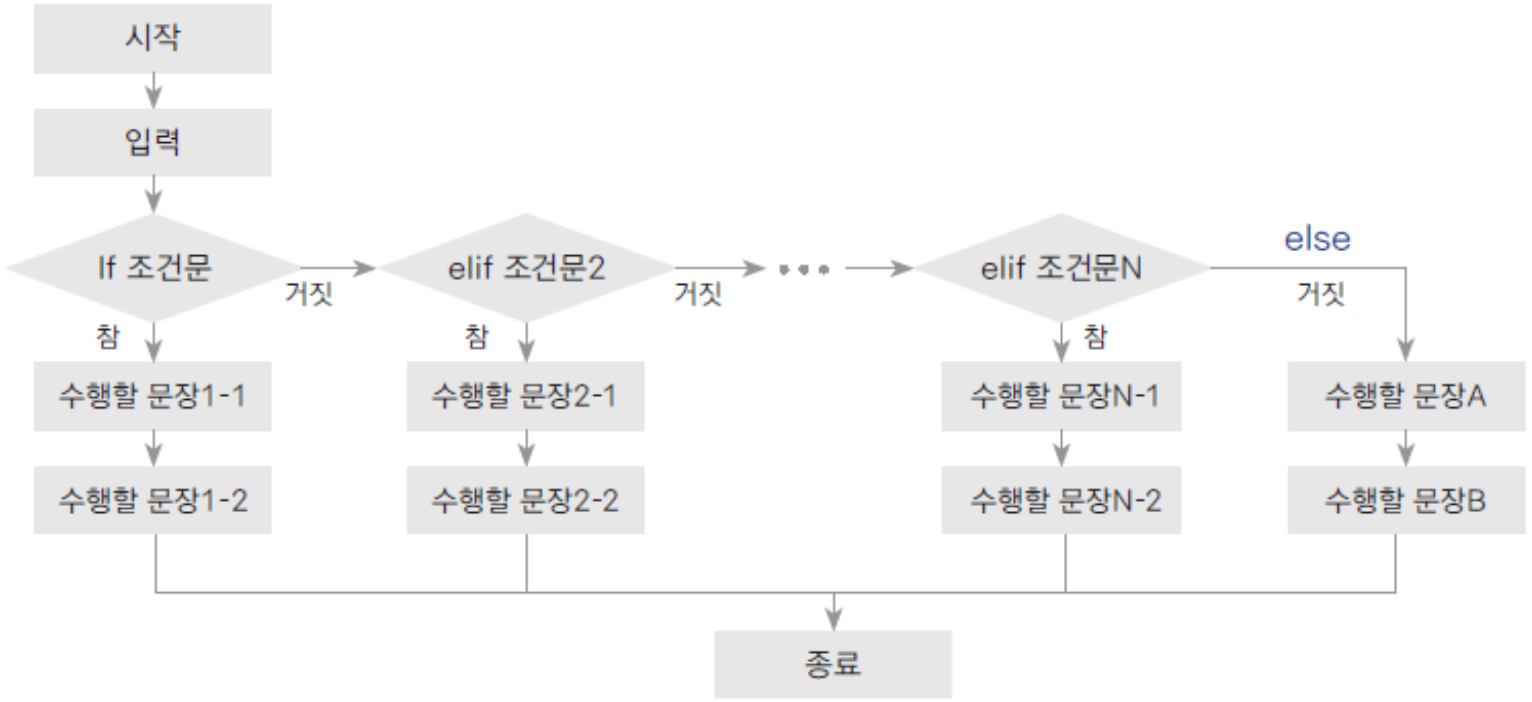
- elif는 이전 조건문이 거짓일 때 수행됨

```
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
```

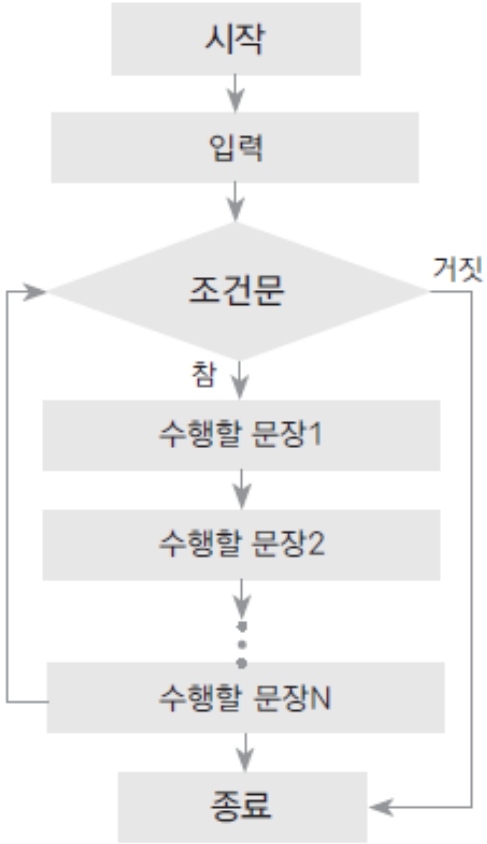
03-2 while 문

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

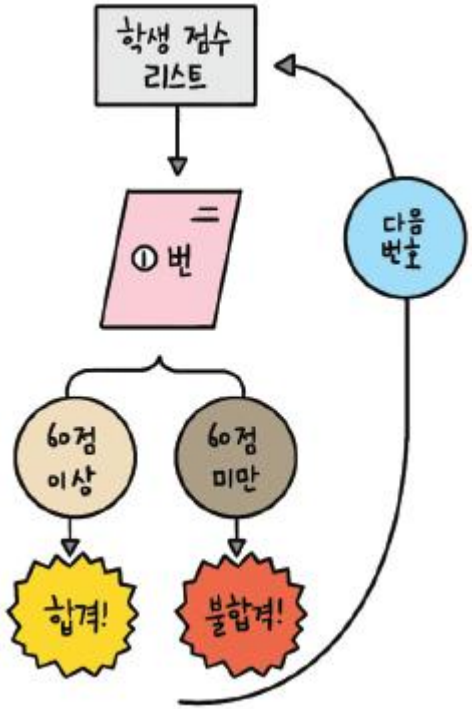
▪ 예제를 통해 for 문 이해하기

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]
```

04

파이썬의 입출력

04-1 함수

04-1 함수

▪ 함수란 무엇인가?

- 우리는 믹서에 과일을 넣고, 믹서를 사용해서 과일을 갈아 과일 주스를 만듦

- 믹서에 넣는 과일 = 입력
- 과일주스 = 출력(결괏값)
- 믹서 = ?

믹서는 과일을 입력받아 주스를 출력하는 함수와 같다.

04-1 함수

- **함수를 사용하는 이유는 무엇일까?**
  - 반복되는 부분이 있을 경우 '반복적으로 사용되는 가치 있는 부분'을 한 뭉치로 묶어 '어떤 입력값을 주었을 때 어떤 결괏값을 리턴해 준다'라는 식의 함수로 작성하는 것이 현명함
  - 프로그램의 흐름을 파악하기 좋고 오류 발생 지점도 찾기 쉬움

04-1 함수

▪ 파이썬 함수의 구조

- def : 함수를 만들 때 사용하는 예약어
- 함수 이름은 임의로 생성 가능
- 매개변수는 함수에 입력으로 전달되는 값을 받는 변수
- return : 함수의 결괏값(리턴값)을 돌려주는 명령어

```
def 함수_이름(매개변수):
    수행할_문장1
    수행할_문장2
    ...
```

```
def add(a, b):
    return a + b
```

- 함수의 풀이

이 함수의 이름은 add이고 입력으로 2개의 값을 받으며 리턴값(출력값)은 2개의 입력값을 더한 값이다.

04-1 함수

파이썬 함수의 구조

- 예) add 함수
  - add 함수 만들기

```
>>> def add(a, b):
...     return a + b
...
>>>
```

  - add 함수 사용하기

```
>>> a = 3
>>> b = 4
>>> c = add(a, b)  <-- add(3, 4)의 리턴값을 c에 대입
>>> print(c)
7
```

04-1 함수

매개변수와 인수

- **매개변수**와 인수는 혼용해서 사용되는 헷갈리는 용어로 잘 구분하는 것이 중요!

- **매개변수(parameter)**
  - 함수에 입력으로 전달된 값을 받는 변수

- **인수(arguments)**
  - 함수를 호출할 때 전달받는 입력값

```
def add(a, b): ← a, b는 매개변수
    return a + b

print(add(3, 4)) ← 3, 4는 인수
```

04-1 함수

▪ 입력값과 리턴값에 따른 함수의 형태

- 함수는 들어온 입력값을 받은 후 어떤 처리를 하여 적절한 값을 리턴해 줌

- 함수의 형태는 입력값과 리턴값의 존재 유무에 따라 4가지 유형으로 나뉨

04-1 함수

▪ 입력값과 리턴값에 따른 함수의 형태

- 일반적인 함수
  - 입력값이 있고 리턴값이 있는 함수
  - 일반 함수의 전형적인 예
    - add 함수는 2개의 입력값을 받아 서로 더한 결괏값을 리턴함

```
def 함수_이름(매개변수):
    수행할_문장
    ...
    return 리턴값
```

```
>>> def add(a, b):
...     result = a + b
...     return result  <-- a + b의 결괏값 리턴
```

```
>>> a = add(3, 4)
>>> print(a)
7
```

```
리턴값을_받을_변수 = 함수_이름(입력_인수1, 입력_인수2, ...)
```

04-1 함수

입력값과 리턴값에 따른 함수의 형태

- 입력값이 없는 함수
  - 입력값이 없는 함수도 존재함

  - say 함수는 매개변수 부분을 나타내는 함수 이름 뒤의 괄호 안이 비어있음
  - 함수 사용 시 say( )처럼 괄호 안에 아무 값도 넣지 않아야 함

```
>>> def say():
...     return 'Hi'
```

```
>>> a = say()
>>> print(a)
Hi
```

```
리턴값을_받을_변수 = 함수_이름()
```

04-1 함수

입력값과 리턴값에 따른 함수의 형태

리턴값이 없는 함수

- 리턴값이 없는 함수는 호출해도 리턴되는 값이 없음

```
>>> def add(a, b):
...     print("%d, %d의 합은 %d입니다." % (a, b, a + b))
```

```
>>> add(3, 4)
3, 4의 합은 7입니다.
```

- 사용 방법

```
함수_이름(입력_인수1, 입력_인수2, ...)
```

04-1 함수

▪ 입력값과 리턴값에 따른 함수의 형태

- 리턴값이 없는 함수
  - 리턴값이 진짜 없을까?
    - 리턴받을 값을 a 변수에 대입하여 출력하여 확인
    - None이란 거짓을 나타내는 자료형으로, 리턴값이 없을 때 쓰임

```
>>> a = add(3, 4) <— add 함수의 리턴값을 a에 대입
3, 4의 합은 7입니다.
>>> print(a) <— a 값 출력
None
```

04-1 함수

입력값과 리턴값에 따른 함수의 형태

- 입력값도, 리턴값도 없는 함수
  - 입력 인수를 받는 매개변수도 없고 return 문도 없는, 즉 입력값도 리턴값도 없는 함수

```
>>> def say():
...     print('Hi')
```

```
>>> say()
Hi
```

```
함수_이름()
```

04-1 함수

▪ 매개변수를 지정하여 호출하기

- 함수 호출 시 매개변수 지정 가능
  - 예) sub 함수

```
>>> def sub(a, b):
...     return a - b
```

  - 매개변수를 지정하여 사용

```
>>> result = sub(a=7, b=3) <— a에 7, b에 3을 전달
>>> print(result)
4
```

- 매개변수를 지정하면 매개변수 순서에 상관없이 사용할 수 있는 장점

```
>>> result = sub(b=5, a=3) <— b에 5, a에 3을 전달
>>> print(result)
-2
```

04-1 함수

- **입력값이 몇 개가 될지 모를 때는 어떻게 해야 할까?**
  - 파이썬에서의 해결 방법
    - 일반 함수 형태에서 괄호 안의 매개변수 부분이 *매개변수로 바뀜

```
def 함수_이름(*매개변수):
    수행할_문장
    ...
```

04-1 함수

- **입력값이 몇 개가 될지 모를 때는 어떻게 해야 할까?**
  - 여러 개의 입력값을 받는 함수 만들기
    - 매개변수 이름 앞에 *을 붙이면 입력값을 전부 모아 튜플로 만들어 줌

```
>>> def add_many(*args):
...     result = 0
...     for i in args:
...         result = result + i  <-- *args에 입력받은 모든 값을 더한다.
...     return result
```

```
>>> result = add_many(1, 2, 3) <-- add_many 함수의 리턴값을 result 변수에 대입
>>> print(result)
6
>>> result = add_many(1, 2, 3, 4, 5, 6, 7, 8, 9, 10)
>>> print(result)
55
```

04-1 함수

▪ 입력값이 몇 개가 될지 모를 때는 어떻게 해야 할까?

- 여러 개의 입력값을 받는 함수 만들기
  - *args 매개변수 앞에 choice 매개변수를 추가할 수도 있음

```
>>> def add_mul(choice, *args):
...     if choice == "add": <- 매개변수 choice에 "add"를 입력받았을 때
...         result = 0
...         for i in args:
...             result = result + i  <- args에 입력받은 모든 값을 더한다.
...     elif choice == "mul": <- 매개변수 choice에 "mul"을 입력받았을 때
...         result = 1
...         for i in args:
...             result = result * i  <- *args에 입력받은 모든 값을 곱한다.
...     return result
```

```
>>> result = add_mul('add', 1, 2, 3, 4, 5)
>>> print(result)
15
>>> result = add_mul('mul', 1, 2, 3, 4, 5)
>>> print(result)
120
```

04-1 함수

키워드 매개변수, kwargs

- 매개변수 앞에 별 2개(**)를 붙임

```
>>> def print_kwargs(**kwargs):
...     print(kwargs)
```

```
>>> print_kwargs(a=1)
{'a': 1}
>>> print_kwargs(name='foo', age=3)
{'age': 3, 'name': 'foo'}
```

04-1 함수

▪ 함수의 리턴값은 언제나 하나이다

- 2개의 입력 인수를 받아 리턴하는 함수

```
>>> def add_and_mul(a, b):
...     return a+b, a*b
```

```
>>> result = add_and_mul(3, 4)
```

```
result = (7, 12)
```

- 하나의 튜플 값을 2개의 값으로 분리하여 리턴

```
>>> result1, result2 = add_and_mul(3, 4)
```

- return 문을 2번 사용하면 리턴값은 하나뿐

```
>>> def add_and_mul(a, b):
...     return a + b
...     return a * b
```

```
>>> result = add_and_mul(2, 3)
>>> print(result)
5
```

04-1 함수

▪ 매개변수에 초깃값 미리 설정하기

- 매개변수에 초깃값을 미리 설정

```
def say_myself(name, age, man=True):
    print("나의 이름은 %s입니다." % name)
    print("나이는 %d살입니다." % age)
    if man:
        print("남자입니다.")
    else:
        print("여자입니다.")
```

  - man=True처럼 매개변수에 미리 값을 넣어 함수의 매개변수 초깃값을 설정

04-1 함수

▪ 매개변수에 초깃값 미리 설정하기

- 매개변수에 들어갈 값이 항상 변하는 것이 아니면, 초깃값을 미리 설정하는 것이 유용함
- say_myself 함수 사용법

```
say_myself("박응용", 27)
```

```
say_myself("박응용", 27, True)
```

실행 결과

```
나의 이름은 박응용입니다.
나이는 27살입니다.
남자입니다.
```

```
say_myself("박응선", 27, False)
```

실행 결과

```
나의 이름은 박응선입니다.
나이는 27살입니다.
여자입니다.
```

- 입력값으로 ("박응용", 27)처럼 2개를 주면 name에는 "박응용"이 old에는 27이 대입됨
- man이라는 변수에는 입력값을 주지 않았지만 초깃값 True를 갖게 됨

04-1 함수

▪ 매개변수에 초깃값 미리 설정하기

- 주의할 점
  - 초기화하고 싶은 매개변수는 항상 뒤쪽에 놓아야 함

```
def say_myself(name, man=True, age):
    print("나의 이름은 %s입니다." % name)
    print("나이는 %d살입니다." % age)
    if man:
        print("남자입니다.")
    else:
        print("여자입니다.")
```

- 파이썬 인터프리터는 27을 man 매개변수와 age 매개변수 중 어느 곳에 대입해야 할지 판단이 어려워 오류 발생

```
say_myself("박응용", 27)
```

실행 결과

```
SyntaxError: non-default argument follows default argument
```

04-1 함수

▪ 함수 안에서 선언한 변수의 효력 범위

- 함수 안에서 사용할 변수의 이름을 함수 밖에서도 동일하게 사용한다면?

```
a = 1              # 함수 밖의 변수 a
def vartest(a):    # vartest 함수 선언
    a = a + 1

vartest(a)         # vartest 함수의 입력값으로 a를 대입
print(a)           # a 값 출력
```

- 함수를 실행해 보면, 결괏값은 1이 나옴
- 함수 안에서 사용하는 매개변수는 함수 안에서만 사용하는 '함수만의 변수'이기 때문
- 즉, 매개변수 a는 함수 안에서만 사용하는 변수일 뿐, 함수 밖의 변수 a와는 전혀 상관없음
- 따라서 vartest 함수는 매개변수 이름을 바꾸어도 이전 함수와 동일하게 동작함

```
def vartest(hello):
    hello = hello + 1
```

04-1 함수

▪ 함수 안에서 선언한 변수의 효력 범위

- 함수 안에서 선언한 매개변수는 함수 안에서만 사용될 뿐,
  함수 밖에서는 사용되지 않음

```
def vartest(a):
    a = a + 1

vartest(3)
print(a)
```

  - print(a)에서 사용한 a 변수는 어디에도 선언되지 않았기 때문

04-1 함수

▪ 함수 안에서 함수 밖의 변수를 변경하는 방법

1. return 사용하기

```
a = 1
def vartest(a):
    a = a + 1
    return a

a = vartest(a)    # vartest(a)의 리턴값을 함수 밖의 변수 a에 대입
print(a)
```

2. global 명령어 사용하기

```
a = 1
def vartest():
    global a
    a = a + 1

vartest()
print(a)
```

- 단, 함수는 독립적으로 존재하는 것이 좋기 때문에 이 방법은 피하는 것이 좋음

04-1 함수

lambda 예약어

- 함수를 생성할 때 사용하는 예약어
- 함수를 한 줄로 간결하게 만들 때 사용
- def와 동일한 역할
- 우리말로 '람다'
- def를 사용해야 할 정도로 복잡하지 않거나 def를 사용할 수 없는 곳에 주로 쓰임

```
함수_이름 = lambda 매개변수1, 매개변수2, ... : 매개변수를_이용한_표현식
```

04-1 함수

lambda 예약어

- 사용 예시

```
>>> add = lambda a, b: a + b
>>> result = add(3, 4)
>>> print(result)
7
```

  - add는 2개의 인수를 받아
    서로 더한 값을 리턴하는 lambda 함수

- def를 사용한 경우와 하는 일이 완전히 동일함

```
>>> def add(a, b):
...     return a + b
...
>>> result = add(3, 4)
>>> print(result)
7
```

04

파이썬의 입출력

04-2 사용자 입출력

▪ 사용자 입력 활용하기

▪ input 사용하기

▪ 사용자가 키보드로 입력한 모든 것을 문자열로 저장

```
>>> a = input()
Life is too short, you need python  <-- 사용자가 문장을 입력
>>> a
'Life is too short, you need python'
```

04-2 사용자 입출력

▪ 사용자 입력 활용하기

- 프롬프트를 띄워 사용자 입력받기
  - 사용자에게 입력받을 때 안내 문구 또는 질문을 보여 주고 싶을 때

```
input("안내_문구")
```

```
>>> number = input("숫자를 입력하세요: ")
숫자를 입력하세요:
```

```
>>> number = input("숫자를 입력하세요: ")
숫자를 입력하세요: 3 <-- 3 입력
>>> print(number)
3
```

04-2 사용자 입출력

print 자세히 알기

- 데이터를 출력하는 데 사용

```
>>> a = 123
>>> print(a) <-- 숫자 출력하기
123
>>> a = "Python"
>>> print(a) <-- 문자열 출력하기
Python
>>> a = [1, 2, 3]
>>> print(a) <-- 리스트 출력하기
[1, 2, 3]
```

04-2 사용자 입출력

print 자세히 알기

- 큰따옴표로 둘러싸인 문자열은 + 연산과 동일하다

```
>>> print("life" "is" "too short") <-①
lifeistoo short
>>> print("life"+"is"+"too short") <-②
lifeistoo short
```

- 문자열 띄어쓰기는 쉼표로 한다

```
>>> print("life", "is", "too short")
life is too short
```

- 한 줄에 결괏값 출력하기

```
>>> for i in range(10):
...     print(i, end = ' ')
...
0 1 2 3 4 5 6 7 8 9 >>>
```

04-3 파일 읽고 쓰기

▪ 파일 생성하기

- 사용자가 직접 '입력'하고 모니터 화면에 결괏값을 '출력'하는 방법만 있는 것은 아님
- 파일을 통한 입출력도 가능

```
f = open("새파일.txt", 'w')
f.close()
```

- 소스코드를 실행하면 프로그램을 실행한 디렉터리에 새로운 파일이 하나 생성됨
- 파일을 생성하기 위해 파이썬 내장 함수 open을 사용한 것

```
파일_객체 = open(파일_이름, 파일_열기_모드)
```

- f.close( )는 열려 있는 파일 객체를 닫아 주는 역할(생략 가능하지만 사용하는 것을 추천)

04-3 파일 읽고 쓰기

▪ 파일 생성하기

▪ 파일 열기 모드

파일 열기 모드	설명
r	읽기 모드: 파일을 읽기만 할 때 사용한다.
w	쓰기 모드: 파일에 내용을 쓸 때 사용한다.
a	추가 모드: 파일의 마지막에 새로운 내용을 추가할 때 사용한다.

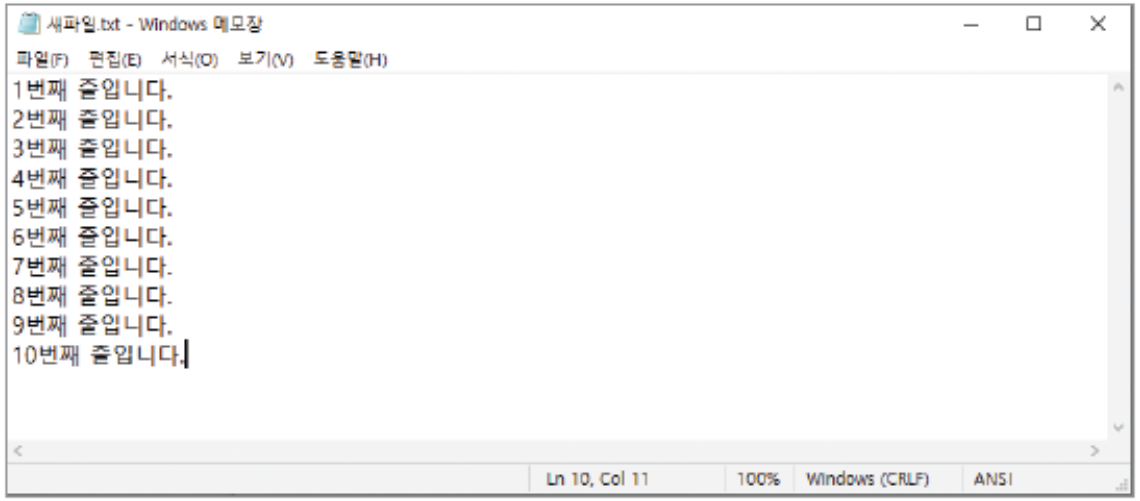
- 파일을 쓰기 모드(w)로 열면 해당 파일이 이미 존재할 경우 원래 있던 내용이 모두 사라지고, 해당 파일이 존재하지 않으면 새로운 파일이 생성됨

04-3 파일 읽고 쓰기

▪ 파일을 쓰기 모드로 열어 내용 쓰기

▪ 문자열 데이터를 파일에 직접 써서 출력

```
f = open("C:/doit/새파일.txt", 'w')
for i in range(1, 11):
    data = "%d번째 줄입니다.\n" % i
    f.write(data)    # data를 파일 객체 f에 써라.
f.close()
```



04-3 파일 읽고 쓰기

▪ 파일을 읽는 여러 가지 방법

- readline 함수 사용하기
  - f.open("새파일.txt", 'r')로 파일을 읽기 모드로 연 후 readline( )을 사용해서 파일의 첫 번째
  - 줄을 읽어 출력하는 코드

```
f = open("C:/doit/새파일.txt", 'r')
line = f.readline()
print(line)
f.close()
```

```
1번째 줄입니다.
```

- 모든 줄을 읽어 화면에 출력하는 코드

```
f = open("C:/doit/새파일.txt", 'r')
while True:
    line = f.readline()
    if not line: break
    print(line)
f.close()
```

  - 무한루프 안에서 f.readline( )을 사용해 파일을 계속 한 줄씩 읽어 들임
  - 더 이상 읽을 줄이 없으면 break 수행
  - readline( )은 더 이상 읽을 줄이 없을 경우 빈 문자열('')을 리턴

04-3 파일 읽고 쓰기

▪ 파일을 읽는 여러 가지 방법

- readlines 함수 사용하기
  - 파일의 모든 줄을 읽어서 각각의 줄을 요소로 가지는 리스트를 리턴
  - ["1번째 줄입니다.₩n", "2번째 줄입니다.₩n", ..., "10번째 줄입니다.₩n"]를 리턴

```
f = open("C:/doit/새파일.txt", 'r')
lines = f.readlines()
for line in lines:
    print(line)
f.close()
```

04-3 파일 읽고 쓰기

▪ 파일을 읽는 여러 가지 방법

- read 함수 사용하기
  - f.read( )는 파일의 내용 전체를 문자열로 리턴
  - data는 파일의 전체 내용

```
f = open("C:/doit/새파일.txt", 'r')
data = f.read()
print(data)
f.close()
```

- 파일 객체를 for 문과 함께 사용하기
  - 파일 객체(f)는 for 문과 함께 사용하여 파일을 줄 단위로 읽을 수 있음

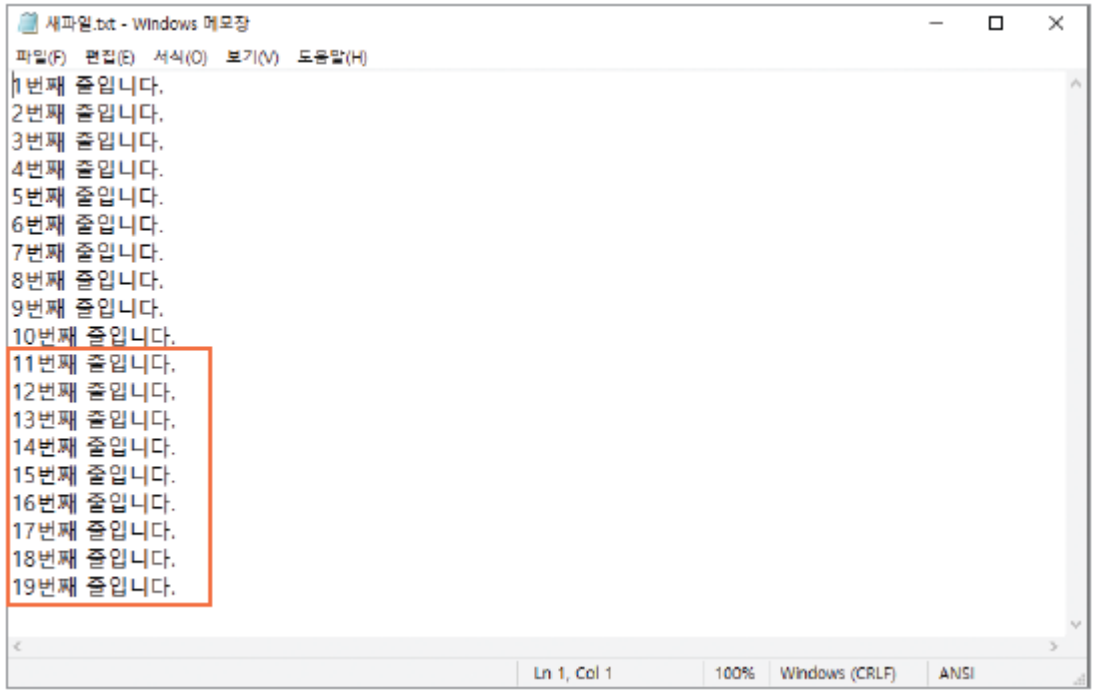
```
f = open("C:/doit/새파일.txt", 'r')
for line in f:
    print(line)
f.close()
```

04-3 파일 읽고 쓰기

▪ 파일에 새로운 내용 추가하기

- 원래 있던 값을 유지하면서 단지 새로운 값만 추가해야 할 경우
- 파일을 추가 모드('a')로 열기

```
f = open("C:/doit/새파일.txt", 'a')
for i in range(11, 20):
    data = "%d번째 줄입니다.\n" % i
    f.write(data)
f.close()
```



04-3 파일 읽고 쓰기

with 문과 함께 사용하기

- 지금까지 파일을 열고 닫은 방법

```
f = open("foo.txt", 'w') <-- 파일 열기
f.write("Life is too short, you need python")
f.close() <-- 파일 닫기
```

  - f.close( )는 열려 있는 파일 객체를 닫아 주는 역할
  - 쓰기 모드로 열었던 파일을 닫지 않고 다시 사용하면 오류가 발생하기 때문에, close( )를 사용해서 열려 있는 파일을 직접 닫아 주는 것이 좋음

04-3 파일 읽고 쓰기

with 문과 함께 사용하기

- with 문은 파일을 열고 닫는 것을 자동으로 처리해주는 문법
- 앞선 예제를 with 문을 사용하여 수정한 코드

```
with open("foo.txt", "w") as f:
    f.write("Life is too short, you need python")
```

- with 문을 사용하면 with 블록을 벗어나는 순간 열린 파일 객체 f가 자동으로 닫힘

04-4 프로그램의 입출력

sys 모듈 사용하기

- 파이썬에서는 sys 모듈을 사용하여 프로그램에 인수 전달 가능
- import 명령어 사용

```
import sys

args = sys.argv[1:]
for i in args:
    print(i)
```

- argv는 프로그램 실행 시 전달된 인수

- argv[0]은 파일 이름 sys1.py, argv[1]부터는 뒤에 따라오는 인수가 차례대로 argv의 요소

04-4 프로그램의 입출력

sys 모듈 사용하기

- 전달된 인수를 모두 대문자로 바꾸는 간단한 프로그램 만들기

```
import sys
args = sys.argv[1:]
for i in args:
    print(i.upper(), end=' ')
```

- 명령 프롬프트

```
C:\doit>python sys2.py life is too short, you need python
```

- 실행 결과

```
LIFE IS TOO SHORT, YOU NEED PYTHON
```

감사합니다.