

02

기초문법

목차

1. 변수와 자료형
2. 입출력 함수
3. 리스트
4. 조건문
5. 반복문

01. 변수와 자료 형

■ 변수명 선언

- 알파벳, 숫자, 밑줄(_)로 선언할 수 있다.
- 변수명은 의미 있는 단어로 표기하는 것이 좋다.
- 변수명은 대소문자가 구분된다.
- 특별한 의미가 있는 예약어는 사용할 수 없다.

01. 변수와 자료 형

■ 기본 자료형

- 정수형(integer type) : 자연수를 포함해 값의 영역이 정수로 한정된 값.
- 실수형(floating-point type) : 소수점이 포함된 값.
- 문자열형(string type) : 값이 문자로 출력되는 자료형.
- 불린형(boolean type) : 논리형으로, 참(True) 또는 거짓(False)을 표현할 때 사용.

유형	자료형	설명	예	선언 형태
수치형	정수형	양수와 정수	1, 2, 3, 100, -9	data = 1
	실수형	소수점이 포함된 실수	10.2, -9.3, 9.0	data = 9.0
문자형	문자형	따옴표에 들어가 있는 문자형	abc, a20abc	data = 'abc'
논리형	불린형	참 또는 거짓	True, False	data = True

01. 변수와 자료 형

여기서 잠깐! 동적 타이핑

- 동적 타이핑(dynamic typing): 변수의 메모리 공간을 확보하는 행위가 실행 시점에서 발생하는 것을 뜻한다.
- C나 자바 : `int data=8` 과 같이 `data`라는 변수가 정수 형이라고 사전에 선언
- 파이썬 : `data = 8` 형태로 선언, `data`라는 변수의 자료 형이 정수(integer)인지 실수(float)인지를 프로그래머가 아닌 인터프리터가 스스로 판단
- 실행 시점에 동적으로 판단하므로 파이썬 언어가 동적으로 자료 형의 결정을 지원
- 다른 언어들과 달리 파이썬은 매우 유연한 언어로, 할당 받는 메모리 공간도 저장되는 값의 크기에 따라 동적으로 다르게 할당 받을 수 있다.

01. 변수와 자료 형

■ 기본 자료형

```
>>> a = 1                # 정수형
>>> b = 1                # 정수형
>>> print(a, b)
1 1
>>> a = 1.5              # 실수형
>>> b = 3.5              # 실수형
>>> print(a, b)
1.5 3.5
>>> a = "ABC"            # 문자형
>>> b = "101010"         # 문자형
>>> print(a, b)
ABC 101010
>>> a = True             # 불린형
>>> b = False            # 불린형
>>> print(a, b)
True False
```

01. 변수와 자료 형

■ 간단한 연산 : 사칙연산(+, -, *, /)

```
>>> 25 + 30
55
>>> 30 - 12
18
>>> 50 * 3
150
>>> 30 / 5
6.0
```

■ 간단한 연산 : 제곱승(**)

```
>>> print(3 * 3 * 3 * 3 * 3)    # 3을 다섯 번 곱함
243
>>> print(3 ** 5)              # 3의 5승
243
```

■ 간단한 연산 : 나눗셈의 정수 몫(//)과 나머지(%) 연산

- 몫을 반환하는 연산자는 2개의 빗금 기호(/), 나머지 연산자는 백분율 기호(%)

```
>>> print(7 // 2)               # 7 나누기 2의 몫
3
>>> print(7 % 2)               # 7 나누기 2의 나머지
1
```

01. 변수와 자료 형

■ 간단한 연산 : 증가 연산과 감소 연산

- 증가 연산자는 +=이고, 감소 연산자는 -=이다.

```
>>> a = 1                # 변수 a에 1을 할당
>>> a = a + 1            # a에 1를 더한 후 그 값을 a에 할당
>>> print(a)             # a 출력
2
>>> a += 1               # a 증가 연산
>>> print(a)             # a 출력
3
>>> a = a - 1            # a에서 1을 뺀 후 그 값을 a에 할당
>>> a -= 1               # a 감소 연산
>>> print(a)             # a 출력
1
```


01. 변수와 자료 형

■ 관계연산

연산자	의미
>	크다
<	작다
>=	크거나 같다
<=	작거나 같다
==	같다
!=	같지 않다

■ 비트 연산

연산자	의미
&	비트 단위 and
	비트 단위 or
^	비트 단위 배타적 or(xor)
~	비트 단위 not
<<	비트 단위 왼쪽 쉬프트
>>	비트 단위 오른쪽 쉬프트

■ 논리연산

연산자	의미
and	그리고
or	또는
not	부정

01. 변수와 자료 형

■ 자료형 변환 : 정수형 -> 실수형 변환

- `float()` 함수 : 정수를 실수형으로 변환해 주는 함수.

```
>>> a = 10                                # a 변수에 정수 데이터 10을 할당
>>> print(a)                              # a가 정수형으로 출력
10
>>> a = float(10)                         # a를 실수형으로 변환 / 정수형인 경우 int()
>>> print(a)                              # a를 출력
10.0                                     # a가 실수형으로 출력됨

>>> a = 10                                # a에 정수 데이터 10 할당
>>> b = 3                                  # b에 정수 데이터 3 할당
>>> print(a / b)                          # 실수형으로 a 나누기 b를 출력
3.3333333333333335                       # 실수형 결과값 출력
```

01. 변수와 자료 형

■ 자료형 변환 : 실수형-> 정수형 변환

- `int()` 함수 : 실수형을 정수형으로 변환해 주는 함수.

```
>>> a = int(10.7)
```

```
>>> b = int(10.3)
```

```
>>> print(a + b)           # 정수형 a와 b의 합을 출력
```

```
20
```

```
>>> print(a)              # 정수형 a값 출력
```

```
10
```

```
>>> print(b)              # 정수형 b값 출력
```

```
10
```

01. 변수와 자료 형

여기서 잠깐! 자동 형 변환이 일어나는 경우

- '10 / 3' : 별도의 형 변환을 하지 않아도 자료 형이 실수로 변환됨, 파이썬의 대표적인 특징인 동적 타이핑 때문에 나타나는 현상 중 하나.
- 값의 크기를 비교: 예로 '1 == True'라고 입력하면 결과는 True로 출력, 아무것도 넣지 않은 " " 같은 문자열을 불린형과 비교하면 False로 인식.
- 모두 파이썬의 특징에 의해 나타나는 현상

01. 변수와 자료 형

■ 자료형 변환 : 문자열형 -> 숫자형 변환

- 실수형 값을 문자형으로 선언하기 위해서는 반드시 따옴표를 붙여 선언해야 한다.

```
>>> a = '76.3'           # a에 문자열 76.3을 할당, 문자열을 의미
>>> b = float(a)         # a를 실수형으로 변환 후 b에 할당
>>> print(a)             # a값 출력
76.3
>>> print(b)             # b값 출력
76.3
>>> print(a + b)         # a와 b를 더함 → 문자열과 숫자열의 덧셈이 불가능하여 에러 발생
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: can only concatenate str (nor "float") to str
```

01. 변수와 자료 형

■ 자료형 변환 : 더하기 연산시 자료형 통일

- 두 변수를 더하기 위해서는 다음과 같이 두 변수의 자료형을 통일해야 한다.

```
>>> a = float(a)           # a를 실수형으로 변환 후 a에 할당
>>> b = a                   # 실수형 a값을 b에 할당
>>> print(a + b)            # 두 실수형을 더한 후 출력
152.6
```

■ 자료형 변환 : 숫자형 -> 문자형 변환

- **str() 함수** : 기존의 정수형이나 실수형을 문자열로 바꿔 준다. 문자형 간의 덧셈은 숫자 연산이 아닌 단순 붙이기(concatenate)가 일어난다.

```
>>> a = str(a)              # 실수형 a값을 문자열로 변환 후 a 할당
>>> b = str(b)              # 실수형 b값을 문자열로 변환 후 b 할당
>>> print(a + b)            # 두 값을 더한 후 출력
76.376.3                    # 문자열 간 덧셈은 문자열 간 단순 연결
```

01. 변수와 자료 형

■ 자료형 확인하기

- **type() 함수** : 자료형을 확인할 수 있는 함수.

```
>>> a = int(10.3)           # a는 정수형으로 10.3을 할당
>>> b = float(10.3)        # b는 실수형으로 10.3을 할당
>>> c = str(10.3)          # c는 문자형으로 10.3을 할당
>>>
>>> type(a)                 # a의 타입을 출력
<class 'int'>
>>> type(b)                 # b의 타입을 출력
<class 'float'>
>>> type(c)                 # c의 타입을 출력
<class 'str'>
```

02. 입출력 함수

■ 출력 함수 : print()

- 문자열 출력 : `print('안녕하세요')`
- 정수, 실수, 부울형 출력 : `print(3), print(3.5), print(True)`
- 변수에 저장된 내용 출력 : `a=10; print(a)`
- 다수의 데이터 형 출력

```
age=20
```

```
print('나이 : ', age)
```

```
name='홍길동'
```

```
print('나는', name, '입니다.')
```

• 출력 형식 사용

문자	의미
%s	문자열(String)
%d	정수(Integer)
%f	실수(float)
%o	8진수
%x	16진수
%%	'%' 문자

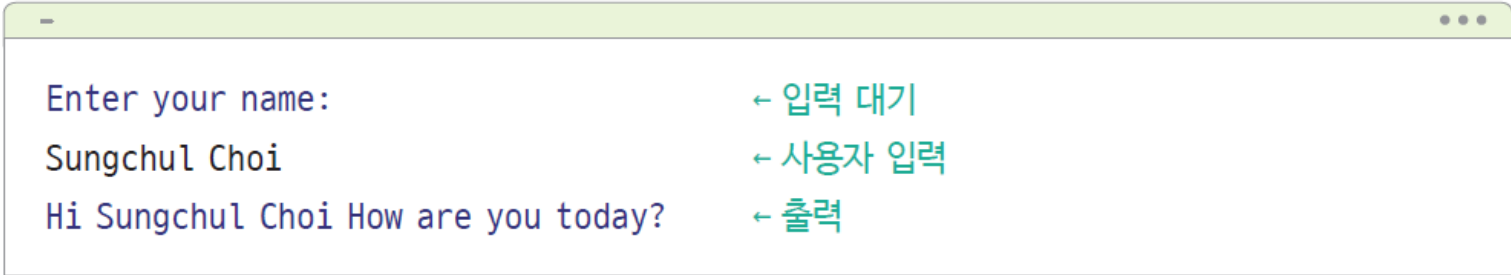
```
a=3  
b=3.5  
c='python'  
print('%d, %f, %s' %(a, b, c))
```


02. 입출력 함수

■ 표준 입력 함수: input() 함수

- input() 함수: 표준 입력 함수로, 사용자가 문자열을 콘솔 창에 입력할 수 있게 해 준다.

```
1 print("Enter your name:")
2 somebody = input()          # 콘솔 창에서 입력한 값을 somebody에 저장
3 print("Hi", somebody, "How are you today?")
```



Enter your name:	← 입력 대기
Sungchul Choi	← 사용자 입력
Hi Sungchul Choi How are you today?	← 출력

```
D:\workspace\Ch03>python input.py
Enter your name:
Sungchul Choi
Hi Sungchul Choi How are you today?
```

[input.py를 cmd 창에서 실행]

02. 입출력 함수

■ 표준 출력 함수: print() 함수

- **print() 함수:** 표준 출력 함수로, 결과를 화면에 출력하는 함수이다.

```
>>> print("Hello World!", "Hello Again!!!")    # 콤마 사용
Hello World! Hello Again!!!                    # 실행 시 두 문장이 연결되어 출력
```

```
1 temperature = float(input("온도를 입력하세요: ")) # 입력 시 바로 형 변환
2 print(temperature)
```



```
온도를 입력하세요: 103    ← 입력 대기 및 사용자 입력
103.0                     ← 출력
```

02. 입출력 함수

■ 실습 내용

- 이번 Lab에서는 앞에서 배운 `input()` 함수, `print()` 함수, 간단한 사칙연산을 이용하여 화씨 온도 변환기(Fahrenheit temperature converter) 프로그램을 만든다.
- 섭씨온도와 화씨온도의 변환 공식은 다음과 같다.

$$\text{화씨온도(°F)} = (\text{섭씨온도(°C)} * 1.8) + 32$$

■ 실행 결과

- 화씨온도 변환기 프로그램의 실행 결과는 다음과 같다.

```
-
```

본 프로그램은 섭씨온도를 화씨온도로 변환하는 프로그램입니다.
변환하고 싶은 섭씨온도를 입력하세요.
32.2
섭씨온도: 32.2
화씨온도: 89.96

02. 입출력 함수

■ 문제 해결

- 화씨온도 변환기 프로그램의 결과 코드는 [코드 3-3]과 같다.

```
1 print("본 프로그램은 섭씨온도를 화씨온도로 변환하는 프로그램입니다.")
2 print("변환하고 싶은 섭씨온도를 입력하세요.")
3
4 celsius = input()
5 fahrenheit = (float(celsius) * 1.8 ) + 32
6
7 print("섭씨온도:", celsius)
8 print("화씨온도:", fahrenheit)
```

03. 리스트

■ 리스트의 개념

- **리스트(list):** 하나의 변수에 여러 값을 할당하는 자료형.
- 리스트는 여러 데이터를 순서대로 하나의 변수에 할당하는 시퀀스 자료형
- 리스트는 하나의 자료형만 저장하지 않고, 정수형이나 실수형 같은 다양한 자료형을 포함할 수 있다.

```
colors = ['red', 'blue', 'green']
```

colors →

'red'	'blue'	'green'
-------	--------	---------

[리스트의 예]

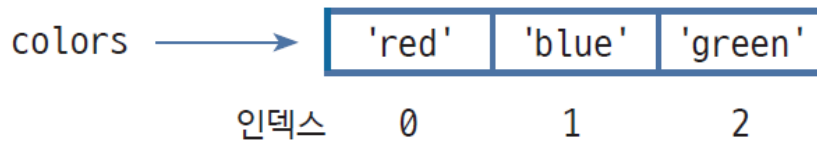
03. 리스트

■ 인덱싱과 슬라이싱 : 인덱싱(indexing)

- **인덱싱**: 리스트에 있는 값에 접근하기 위해, 이 값의 상대적인 주소를 사용하는 것이다.

```
1 colors = ['red', 'blue', 'green']  
2 print(colors[0])  
3 print(colors[2])  
4 print(len(colors))
```

```
red  
green  
3
```



[리스트의 인덱싱]

03. 리스트

■ 인덱싱과 슬라이싱 : 슬라이싱(slicing)

- 슬라이싱: 리스트의 인덱스를 사용하여 전체 리스트에서 일부를 잘라내어 반환한다.

```
>>> cities = ['서울', '부산', '인천', '대구', '대전', '광주', '울산', '수원']
```

값	['서울', '부산', '인천', '대구', '대전', '광주', '울산', '수원']							
인덱스	0	1	2	3	4	5	6	7

[cities 변수의 리스트]

03. 리스트

■ 인덱싱과 슬라이싱 : 슬라이싱(slicing)

- 슬라이싱의 기본 문법

```
변수명[시작 인덱스:마지막 인덱스]
```

```
>>> cities = ['서울', '부산', '인천', '대구', '대전', '광주', '울산', '수원']  
>>> cities[0:6]  
['서울', '부산', '인천', '대구', '대전', '광주']
```

- 파이썬의 리스트에서 '마지막 인덱스 - 1'까지만 출력된다. 만약 한 번 이상 리스트 변수를 사용하면 마지막 인덱스가 다음 리스트의 시작 인덱스가 되어 코드를 작성할 때 조금 더 쉽게 이해할 수 있다는 장점이 있다.

```
>>> cities[0:5]  
['서울', '부산', '인천', '대구', '대전']  
>>> cities[5:]  
['광주', '울산', '수원']
```


03. 리스트

■ 인덱싱과 슬라이싱 : 리버스 인덱스(reverse index)

- 리스트에는 인덱스를 마지막 값부터 시작하는 리버스 인덱스 기능이 있다.

값	['서울', '부산', '인천', '대구', '대전', '광주', '울산', '수원']							
인덱스	-8	-7	-6	-5	-4	-3	-2	-1

[cities 변수의 리버스 인덱스]

- 일반적으로 시작 인덱스가 비어 있으면 처음부터, 마지막 인덱스가 비어 있으면 마지막까지라는 의미로 사용된다. 즉, cities[-8:]은 인덱스가 -8인 '서울'부터 '수원'까지 출력하라는 뜻이다.

```
>>> cities = ['서울', '부산', '인천', '대구', '대전', '광주', '울산', '수원']
>>> print(cities[-8:])
['서울', '부산', '인천', '대구', '대전', '광주', '울산', '수원']
```

03. 리스트

■ 인덱싱과 슬라이싱 : 인덱스 범위를 넘어가는 슬라이싱

- 인덱스를 따로 넣지 않고 `print(cities[:])`과 같이 콜론(:)을 넣으면 `cities` 변수의 모든 값을 다 반환한다.
- 슬라이싱에서는 인덱스를 넘어서거나 입력하지 않더라도 자동으로 시작 인덱스와 마지막 인덱스로 지정된다

```
>>> cities = ['서울', '부산', '인천', '대구', '대전', '광주', '울산', '수원']
>>> print(cities[:])           # cities 변수의 처음부터 끝까지
['서울', '부산', '인천', '대구', '대전', '광주', '울산', '수원']
>>> print(cities[-50:50])      # 범위를 넘어갈 경우 자동으로 최대 범위를 지정
['서울', '부산', '인천', '대구', '대전', '광주', '울산', '수원']
```

03. 리스트

■ 인덱싱과 슬라이싱 : 증가값(step)

- 슬라이싱에서는 시작 인덱스와 마지막 인덱스 외에 마지막 자리에 증가값을 넣을 수 있다.
- 증가값은 한 번에 건너뛰는 값의 개수이다.

변수명[시작 인덱스:마지막 인덱스:증가값]

```
>>> cities = ['서울', '부산', '인천', '대구', '대전', '광주', '울산', '수원']
>>> cities[::2]                                # 2칸 단위로
['서울', '인천', '대전', '울산']
>>> cities[::-1]                              # 역으로 슬라이싱
['수원', '울산', '광주', '대전', '대구', '인천', '부산', '서울']
```

03. 리스트

■ 리스트의 연산

- **덧셈 연산** : 덧셈 연산을 하더라도 따로 어딘가 변수에 할당해 주지 않으면 기존 변수는 변화가 없다

```
>>> color1 = ['red', 'blue', 'green']
>>> color2 = ['orange', 'black', 'white']
>>> print(color1 + color2)           # 두 리스트 합치기
['red', 'blue', 'green', 'orange', 'black', 'white']
>>> len(color1)                     # 리스트 길이
3

>>> total_color = color1 + color2
>>> total_color
['red', 'blue', 'green', 'orange', 'black', 'white']
```

03. 리스트

■ 리스트의 연산

- **곱셈 연산** : 리스트의 곱셈 연산은 기준 리스트에 n을 곱했을 때, 같은 리스트를 n배만큼 늘려 준다.

```
>>> color1 * 2                                     # color1 리스트 2회 반복
['red', 'blue', 'green', 'red', 'blue', 'green']
```

- **in 연산** : 포함 여부를 확인하는 연산으로, 하나의 값이 해당 리스트에 들어 있는지 확인할 수 있다.

```
>>> 'blue' in color2                               # color2 변수에서 문자열 'blue'의 존재 여부 반환
False
```

03. 리스트

■ 리스트 추가 및 삭제

- **append() 함수** : 새로운 값을 기존 리스트의 맨 끝에 추가

```
>>> color = ['red', 'blue', 'green']  
>>> color.append('white')           # 리스트에 'white' 추가  
>>> color  
['red', 'blue', 'green', 'white']
```

- **extend() 함수** : 새로운 리스트를 기존 리스트에 추가

```
>>> color = ['red', 'blue', 'green']  
>>>  
>>> color.extend(['black', 'purple']) # 리스트에 새로운 리스트 추가  
>>> color  
['red', 'blue', 'green', 'black', 'purple']
```

03. 리스트

■ 리스트 추가 및 삭제

- **insert() 함수** : 기존 리스트의 i번째 인덱스에 새로운 값을 추가, i번째 인덱스를 기준으로 뒤쪽의 인덱스가 하나씩 밀림.

```
>>> color = ['red', 'blue', 'green']
>>>
>>> color.insert(0, 'orange')
>>> color
['orange', 'red', 'blue', 'green']
```

- **remove() 함수** : 리스트 내의 특정 값을 삭제.

```
>>> color
['orange', 'red', 'blue', 'green']
>>>
>>> color.remove('red')
>>> color
['orange', 'blue', 'green']
```

03. 리스트의 이해

■ 리스트 추가 및 삭제

- **인덱스의 재할당** : 인덱스에 새로운 값을 할당한다.
- **인덱스 삭제** : del 함수를 사용한다.

```
>>> color = ['red', 'blue', 'green']
>>> color[0] = 'orange'
>>> color
['orange', 'blue', 'green']
>>> del color[0]
>>> color
['blue', 'green']
```


03. 리스트의 이해

■ 리스트 추가 및 삭제 함수

함수	기능	용례
append()	새로운 값을 기존 리스트의 맨 끝에 추가	color.append('white')
extend()	새로운 리스트를 기존 리스트에 추가(덧셈 연산과 같은 효과)	color.extend(['black','purple'])
insert()	기존 리스트의 i번째 인덱스에 새로운 값을 추가. i번째 인덱스를 기준으로 뒤쪽의 인덱스는 하나씩 밀림	color.insert(0, 'orange')
remove()	리스트 내의 특정 값을 삭제	color.remove('white')
del	특정 인덱스값을 삭제	del color[0]

03. 리스트

■ 패킹과 언패킹

- 패킹(packing): 한 변수에 여러 개의 데이터를 할당하는 것.
- 언패킹(unpacking): 한 변수의 데이터를 각각의 변수로 반환하는 것.

```
>>> t = [1, 2, 3]
```

```
# 1, 2, 3을 변수 t에 패킹
```

```
>>> a, b, c = t
```

```
# t에 있는 값 1, 2, 3을 변수 a, b, c에 언패킹
```

```
>>> print(t, a, b, c)
```

```
[1, 2, 3] 1 2 3
```

03. 리스트의 이해

■ 패킹과 언패킹

- 다음 코드처럼 리스트에 값이 3개인데, 5개로 언패킹을 시도한다면 어떤 결과가 나올까? 다음 코드에서 보는 것처럼 언패킹 시 할당받는 변수의 개수가 적거나 많으면 모두 에러가 발생한다.

```
>>> t = [1, 2, 3]
>>> a, b, c, d, e = t
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ValueError: not enough values to unpack (expected 5, got 3)
>>> a, b = t
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ValueError: too many values to unpack (expected 2)
```

03. 리스트

■ 이차원 리스트

- 리스트를 효율적으로 활용하기 위해 여러 개의 리스트를 하나의 변수에 할당하는 이차원 리스트를 사용할 수 있다.
- 이차원 리스트는 표의 칸에 값을 채웠을 때 생기는 값들의 집합이다.

학생	A	B	C	D	E
국어 점수	49	79	20	100	80
수학 점수	43	59	85	30	90
영어 점수	49	79	48	60	100

[이차원 리스트를 설명하기 위한 예]

03. 리스트

■ 이차원 리스트

- 이차원 리스트를 하나의 변수로 표현하기 위해서는 다음과 같이 코드를 작성
- 이차원 리스트에 인덱싱하여 값에 접근하기 위해서는 대괄호 2개를 사용.

```
>>> kor_score = [49, 79, 20, 100, 80]
>>> math_score = [43, 59, 85, 30, 90]
>>> eng_score = [49, 79, 48, 60, 100]
>>> midterm_score = [kor_score, math_score, eng_score]
>>> midterm_score
[[49, 79, 20, 100, 80], [43, 59, 85, 30, 90], [49, 79, 48, 60, 100]]
```

```
>>> print(midterm_score[0][2])
20
```

03. 리스트

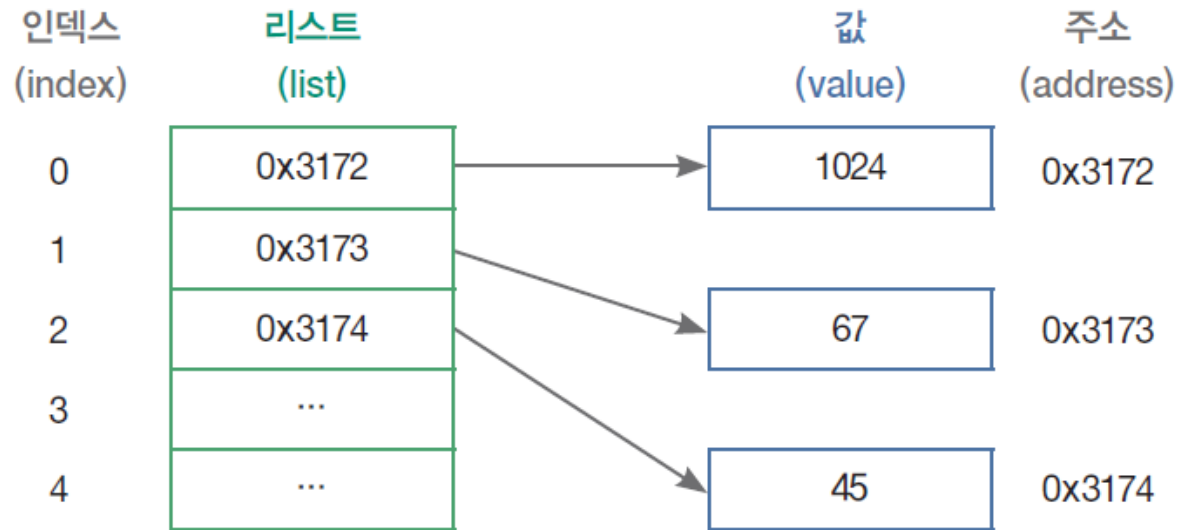
■ 리스트의 메모리 저장

```
>>> kor_score = [49, 79, 20, 100, 80]
>>> math_score = [43, 59, 85, 30, 90]
>>> eng_score = [49, 79, 48, 60, 100]
>>>
>>> midterm_score = [kor_score, math_score, eng_score]
>>> midterm_score
[[49, 79, 20, 100, 80], [43, 59, 85, 30, 90], [49, 79, 48, 60, 100]]
>>>
>>> math_score[0] = 1000
>>> midterm_score
[[49, 79, 20, 100, 80], [1000, 59, 85, 30, 90], [49, 79, 48, 60, 100]]
```

03. 리스트

■ 리스트의 메모리 저장

- 파이썬은 리스트를 저장할 때 값 자체가 아니라, 값이 위치한 메모리 주소 (reference)를 저장.



[리스트의 메모리 저장]

03. 리스트

■ 리스트의 메모리 저장

- ==은 값을 비교하는 연산이고, is는 메모리의 주소를 비교하는 연산.
- 아래 코드에서 a와 b는 값은 같지만, 메모리의 저장 주소는 다른 것이다.

```
>>> a = 300
>>> b = 300
>>> a is b
False
>>> a == b
True
```

차이점

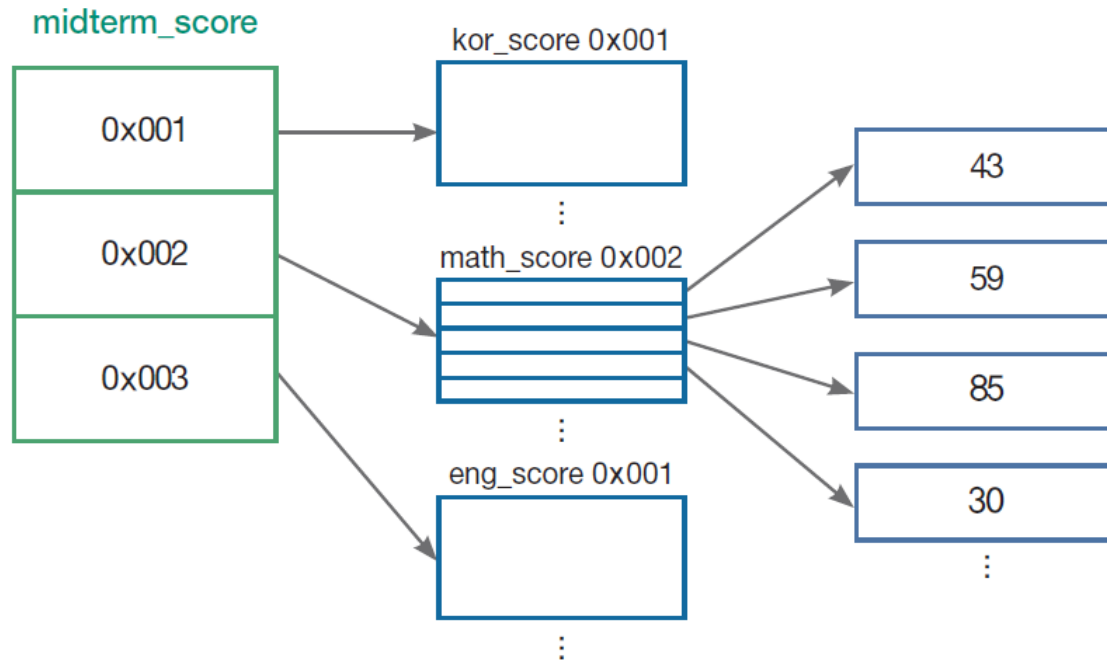
```
>>> a = 1
>>> b = 1
>>> a is b
True
>>> a == b
True
```

- 오른쪽은 is와 == 연산자는 모두 True를 반환한다. 이것은 파이썬의 정수형 저장 방식의 특성 때문이다. 파이썬은 인터프리터가 구동될 때, -5부터 256까지의 정수 값을 특정 메모리 주소에 저장한다. 그리고 해당 숫자를 할당하려고 하면 해당 변수는 그 숫자가 가진 메모리 주소로 연결한다.

03. 리스트

■ 리스트의 메모리 저장

- 리스트는 기본적으로 값을 연속으로 저장하는 것이 아니라, 값이 있는 주소를 저장하는 방식이다.



[리스트의 메모리 저장 연결 관계]

03. 리스트

■ 메모리 저장 구조로 인한 리스트의 특징

- 다양한 형태의 변수가 하나의 리스트에 들어갈 수 있다.

```
>>> a = ["color", 1, 0.2]
```

- 기존 변수들과 함께 리스트 안에 다른 리스트를 넣을 수 있다. 흔히 이를 중첩 리스트라고 한다. 이러한 특징은 파이썬의 리스트가 값이 아닌 메모리의 주소를 저장해 메모리에 새로운 값을 할당하는 데 있어 매우 높은 자유도를 보장하므로 가능하다.

```
>>> a = ["color", 1, 0.2]
>>> color = ['yellow', 'blue', 'green', 'black', 'purple']
>>> a[0] = color                                # 리스트 안에 리스트도 입력 가능
>>> print(a)
[['yellow', 'blue', 'green', 'black', 'purple'], 1, 0.2]
```

03. 리스트

■ 메모리 저장 구조로 인한 리스트의 특징

- 리스트의 저장 방식
 - b와 a 변수를 각각 다른 값으로 선언한 후, b에 a를 할당하였다. 왼쪽 그림과 같이 b를 출력하면, a 변수와 같은 값이 화면에 출력

```
>>> a = [5, 4, 3, 2, 1]
>>> b = [1, 2, 3, 4, 5]
>>> b = a
>>> print(b)
[5, 4, 3, 2, 1]
```

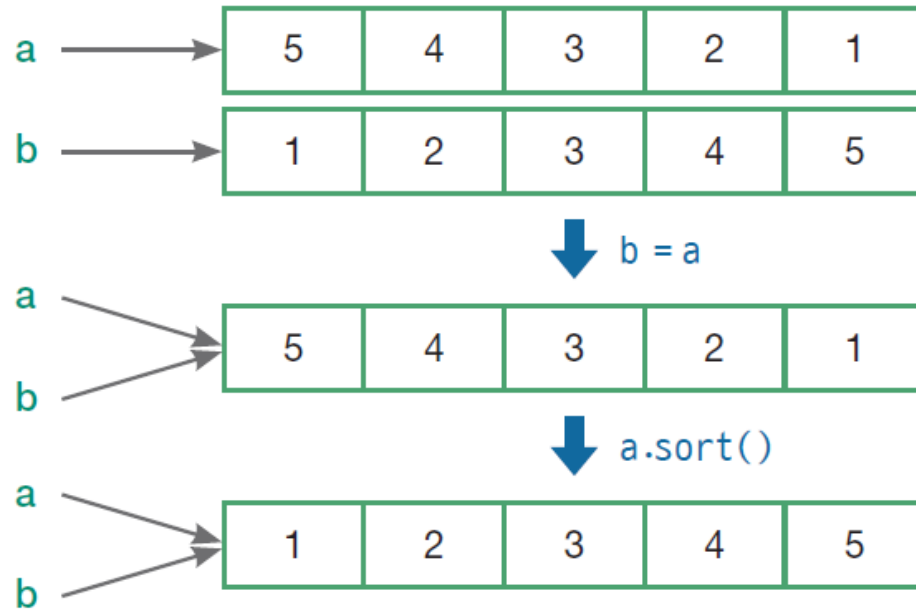
```
>>> a.sort()
>>> print(b)
[1, 2, 3, 4, 5]
```

- 오른쪽 그림과 같이 a만 정렬하고 b를 출력했을 때 b도 정렬되었다.

03. 리스트

■ 메모리 저장 구조로 인한 리스트의 특징

- 리스트의 저장 방식



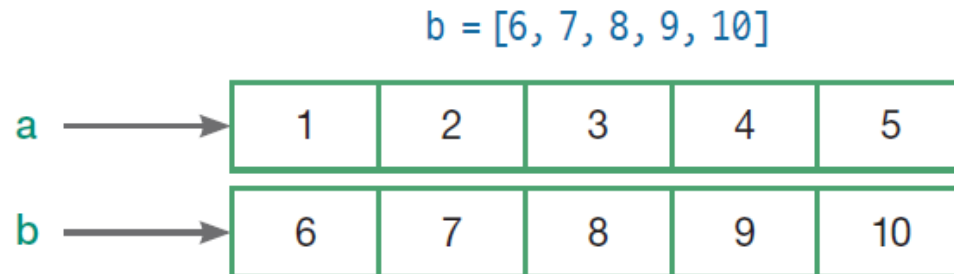
[`a`를 정렬했는데 `b`도 정렬되는 이유]

03. 리스트

■ 메모리 저장 구조로 인한 리스트의 특징

- 리스트의 저장 방식
 - b에 새로운 값을 할당하면 b는 이제 새로운 메모리 주소에 새로운 값을 할당할 수 있는 것이다.

```
>>> b = [6, 7, 8, 9, 10]
>>> print(a, b)
[1, 2, 3, 4, 5] [6, 7, 8, 9, 10]
```

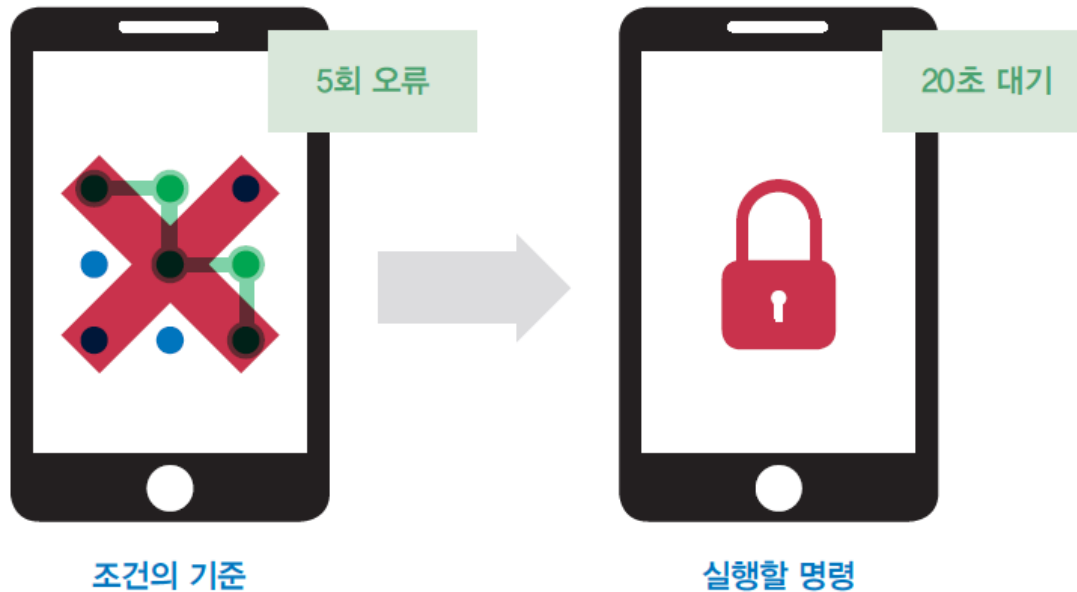


[b에 새로운 값을 할당하는 경우]

04. 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]

04. 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.

04. 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
```


04. 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

[비교 연산자]

04. 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
```

04. 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

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

04. 조건문

■ if-elif-else문

- 중첩 if문을 간단히 표현하려면 if-elif-else문을 사용한다.

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

[점수판]

```
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

04. 조건문

■ if-elif-else문

- 앞의 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.

```
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      ← 사용자 점수 입력
A                          ← 올바른 값 출력
```

04. 조건문

■ 실습 내용

- 조건문을 이용하여 '어떤 종류의 학생인지 맞히는 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

04. 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 `1`과 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.

05. 반복문

■ for문

- **for문:** 기본적인 반복문으로, 반복 범위를 지정하여 반복을 수행한다.
- for문으로 반복문을 만들 때는 먼저 for를 입력하고 반복되는 범위를 지정해야 한다.

- ① 리스트를 사용해서 반복되는 범위를 지정하는 방법

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

```
-  
  
hello  
hello  
hello  
hello  
hello
```

- ② 변수 자체를 출력하여 반복되는 범위를 지정하는 방법

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

```
-  
  
1  
2  
3  
4  
5
```


02. 반복문

■ For문

■ range 문법

`for 변수 in range(시작 번호, 마지막 번호, 증가값)`

- 마지막 번호의 마지막 숫자 바로 앞까지 리스트를 만든다.
- `range(1, 5)`라고 하면 `[1, 2, 3, 4]`를, `range(0, 5)`라고 하면 `[0, 1, 2, 3, 4]` 리스트를 작성
- 시작 번호와 증가 값은 생략 가능, 생략했을 경우 시작 번호는 0을, 증가값은 1을 사용.

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

```
hello  
:  
:  
hello
```

← 100번 반복
← 생략

05. 반복문

■ for문

- 문자열은 리스트와 같은 연속적인 데이터를 표현, 각 문자를 변수 i에 할당하여 반복문 사용 화면에 출력 .

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

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

- 문자열로 이루어진 리스트의 값 반복문 사용.

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

```
-  
  
americano  
latte  
frappuccino
```

05. 반복문

■ for문

- range 구문을 이용하여 1부터 9까지 2씩 증가시키는 for문

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

- range 구문을 사용하여 10부터 2까지 1씩 감소시키는 반복문

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

- range 구문을 사용하여 10부터 2까지 1씩 감소시키는 반복문

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

05. 반복문

■ while문

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

```
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
```

05. 반복문

■ 반복문의 제어 : **break**문

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

```
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
```

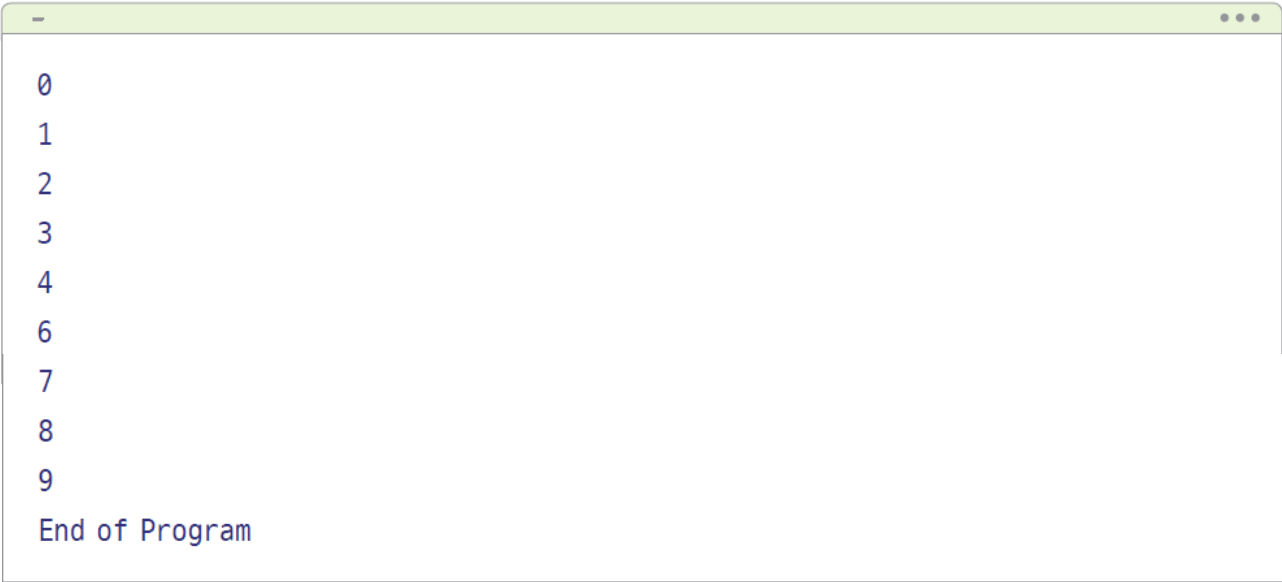
```
0  
1  
2  
3  
4  
End of Program
```

05. 반복문

■ 반복문의 제어 : **continue**문

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

```
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
```



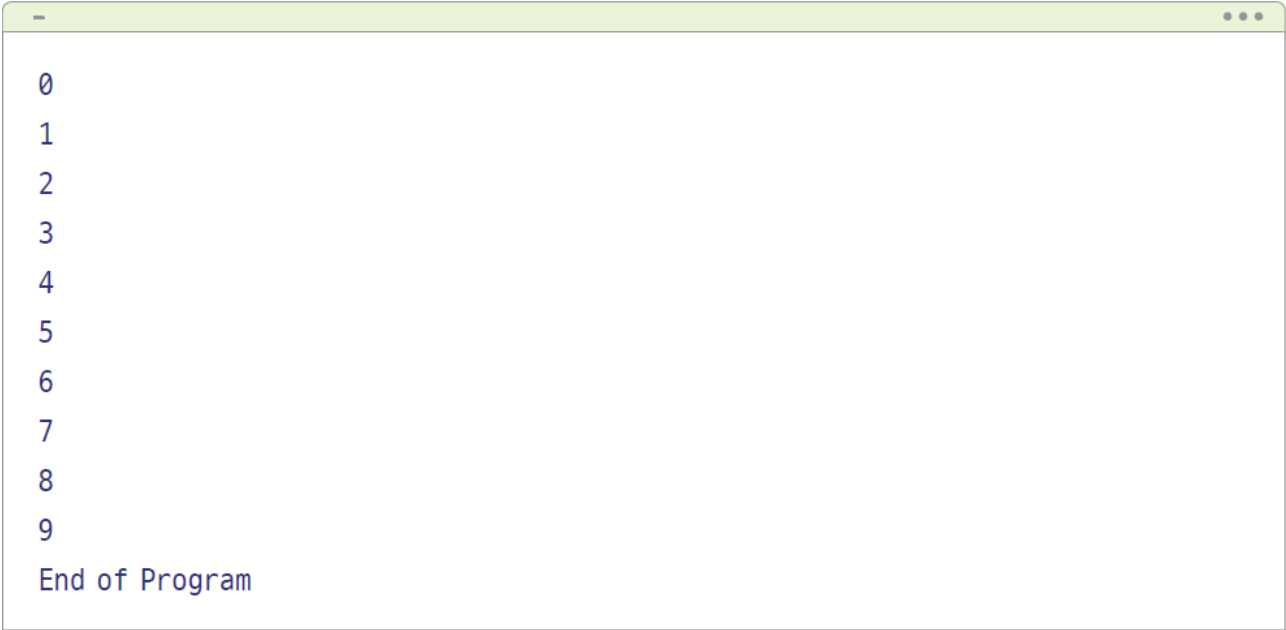
```
0  
1  
2  
3  
4  
6  
7  
8  
9  
End of Program
```

02. 반복문

■ 반복문의 제어 : **else문**

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

```
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
```

06. 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.

06. 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)
```

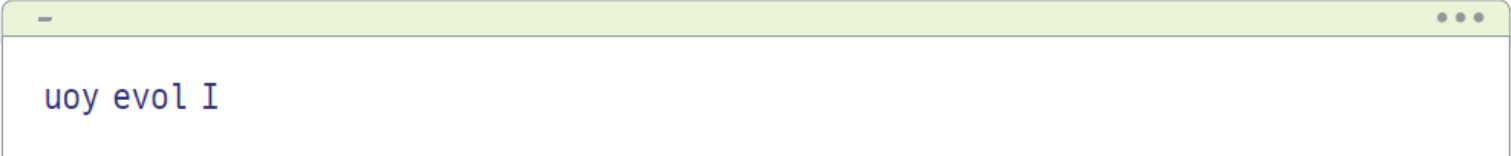
06. 조건문과 반복문 실습

■ 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)
```

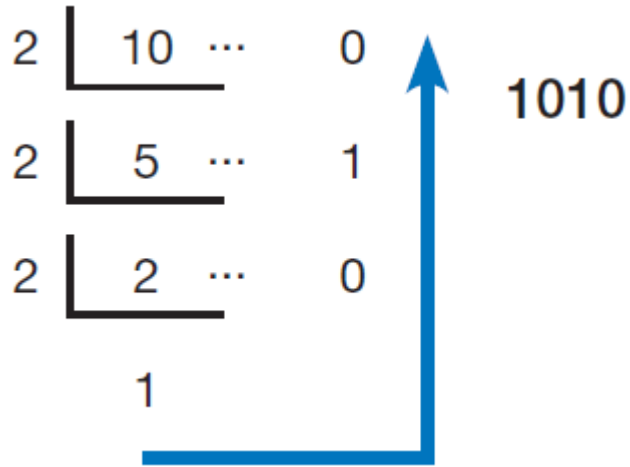


uoy evol I

06. 조건문과 반복문 실습

■ 십진수를 이진수로 변환

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



[십진수를 이진수로 변환하는 방법]

06. 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

0. 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
```

06. 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.

06. 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.

$$5 \times 1 = 5$$

$$5 \times 2 = 10$$

$$5 \times 3 = 15$$

$$5 \times 4 = 20$$

$$5 \times 5 = 25$$

$$5 \times 6 = 30$$

$$5 \times 7 = 35$$

$$5 \times 8 = 40$$

$$5 \times 9 = 45$$

구구단 몇 단을 계산할까요(1~9)?

0

구구단 게임을 종료합니다.

06. 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.")
```


06. 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]
```

06. 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)
```

06. 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]]