

01 자료형이란?

◆ 자료형의 정의

자료형은 프로그래밍 언어에서 사용되는 **변수의 데이터 타입**을 뜻함

ex) 정수형 (1, 2, 3) , 실수형 (0.1, 0.2, 0.3) , 문자형 ('가', '나', '안녕'), 논리값 (true or false)

변수: 메모리에 값을 저장하기 위해 **할당하는 공간**

(할당 후 내부의 값 변경 가능)

x	=	10
변수 이름		값

→ 변수 x에 10이라는 값을 할당한다.

01 자료형이란?

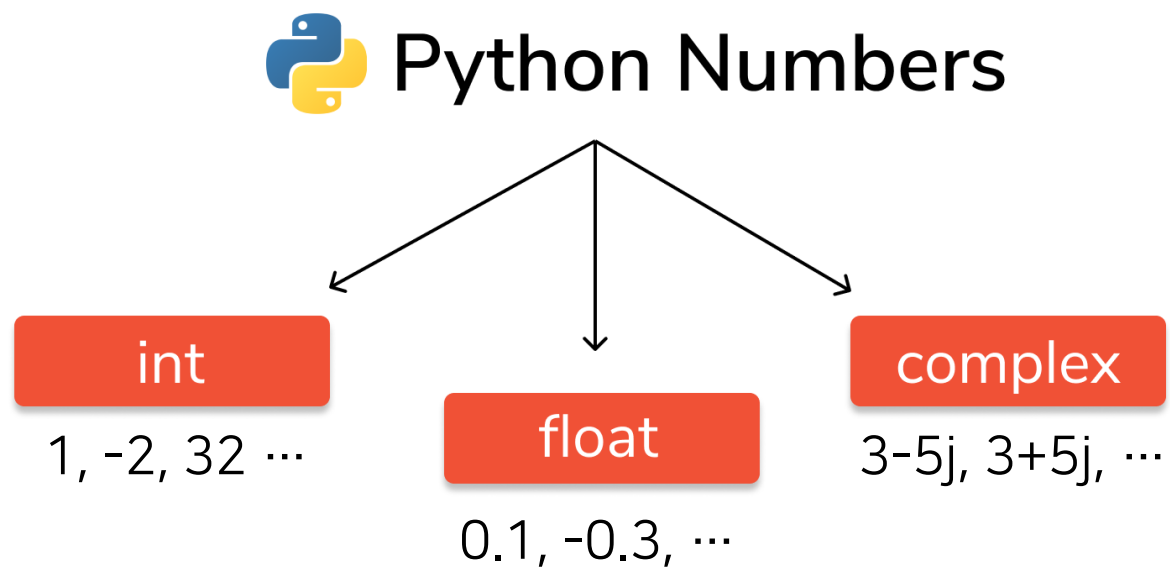
자료형의 종류

	Example
Numbers	1, 23, -34.22, 1e-05
Boolean	True, False
String	'딥앤하이라닝', 'Deep&HighLearning'
List	[1, 3, 5], ['a', 'b', 'c'], [23.5, 'ai', True]
Tuple	(1, 3, 5), ('a', 'b', 'c'), (23.5, 'ai', True)
Dictionary	{'a':1, 'b':2, 'c':3, 'd':4}
Set	(kim, choi, chang, lee)

◈ 숫자 자료형의 종류

숫자 형태로 이루어진 자료형

정수인 **integer**와 실수인 **float**, 복소수인 **complex** 세가지로 구분



◆ int 자료형

- 정수형을 나타내는 자료형.
- 범위: $-\infty \sim +\infty$

Int 선언

```
a = 30
b = 2
c = 0
d = -12
print(a, b, c, d)
print(type(a))
```

Result

```
30 2 0 -12
<class 'int'>
```

◆ float 자료형

- 실수형을 나타내는 자료형.
- 범위: $4.9 \times 10^{-324} \sim 1.8 \times 10^{308}$

float 선언

```
a = 3.4
b = -2.5
c = 3.5e-3
d = 2e+3
print(a, b, c, d)
print(type(a))
```

Result

```
3.4 -2.5 0.0035 2000.0
<class 'float'>
```

◈ int와 float 사이 형변환

- float to int : 소수점이 버려짐
- int to float : 뒤에 .0이 붙음

float to int

```
a = 3.7  
b = int(a)  
print(a, b)
```

Result

```
3.7 3
```

int to float

```
a = 3  
b = float(a)  
print(a, b)
```

Result

```
3 3.0
```

◆ complex 자료형

- 복소수형을 나타내는 자료형
- 실수부와 허수부, 켈레복소수, 복소수의 크기, 복소수의 연산 등 수행 가능

float to int

```
a = 2 + 3j  
b = complex(3, -4)  
print(a, b)
```

Result

```
(2+3j) (3-4j)
```

float to int

```
a = 2 + 3j  
b = complex(3, -4)  
print(a.real, a.imag, b.real, b.imag)
```

Result

```
2.0 3.0 3.0 -4.0
```

complex 자료형

복소수 연산

```
a = 2 + 3j
b = complex(3, -4)
print(a + b, a - b)
print(a * b, a / b)
```

Result

```
(5-1j) (-1+7j)
(18+1j) (-0.24+0.68j)
```

켈레복소수와 크기

```
a = 2 + 3j
b = complex(3, -4)
print(a.conjugate(), b.conjugate())
print(abs(a), abs(b))
```

Result

```
(2-3j) (3+4j)
3.605551275463989 5.0
```


◆ 숫자형 연산자

숫자형 연산자

```
a = 8
b = 3

print(a + b, a - b)
print(a * b, a / b)
print(a ** b)
print(a % b)
print(a // b)
```

Result

```
11 5
24 2.6666666666666665
512
2
2
```

연습 문제!

언어 = 90, 영어 = 60, 수학 = 81

위 학생의 평균 성적을 구하는 코드를 작성하세요.

Code

```
kor = 90
eng = 60
mat = 81

...

print(average)
```

◆ string 자료형

- 문자, 단어 등으로 구성된 문자들의 집합 나타내는 자료형
- " " 나 ' ' 안에 문자열을 넣어 선언

string 선언

```
a = "딥앤하이라닝"  
b = "딥앤하이\"러닝"  
c = '딥앤하이"러닝'  
d = "딥앤하이'러닝"
```

```
print(a, b)  
print(c, d)  
print(type(a))
```

Result

```
딥앤하이라닝 딥앤하이"러닝 딥앤하이"  
러닝 딥앤하이'러닝  
<class 'str'>
```

03 문자 자료형

◆ string 자료형

긴문자열을 `""" """` 나 `' '`를 사용하여 표현

복소수 연산

```
multiline = """  
Life is too short  
You need python  
"""  
  
print(multiline)
```

Result

```
Life is too short  
You need python
```

◆ string 자료형

+와 *연산자를 활용하여 문자열 반복 저장 가능

multiline (+)

```
print(1 + 1)
print('a' + 'b')
```

Result

2 ab

multiline (*)

```
a = "Deep " * 4
print(a)
```

Result

Deep Deep Deep Deep

◆ Offset (오프셋)

Offset (오프셋) : 컴퓨터 내 특정 주소로부터의 간격
문자열을 자르거나 특정 위치의 문자를 출력 가능

```
D e e p & H i g h L e a r n i n g
0 1 2 3 4 5 6      ...      15 16
```

Offset (오프셋)

Indexing : 문자열의 특정 위치 문자 가져오기

Slicing : 문자열의 특정 부분 가져오기

오프셋 형태 : [start:end:stride]

양수는 첫째 문자부터 시작, 음수는 가장 뒤 문자부터 시작

offset

```
a = "abcdefghijk"
print(a[1:3], a[:5], a[-3:], a[:], a[::2])
```

Result

```
bc abcde ijk abcdefghijk acegik
```

◈ Offset (오프셋)

offset 활용

```
teacher = "Kim's "  
title = "Deep&High Learning"  
print(teacher + title)  
print("=" * 30)  
print(len(title))  
print(title[0])  
print(title[-1])  
print(title[:2])  
print(title[3:])
```

Result

```
Kim's Deep&High Learning  
=====  
18  
D  
g  
De  
p&High Learning
```


◆ 주요 문자열 메서드 (method)

문자열에 여러 가지 변환을 가하는 데에 사용

count() : 문자열 개수 반환

find() : 해당 문자열 위치 반환

(없다면 -1 반환)

index() : 해당 문자열 위치 반환

(없다면 'value error' 반환)

upper() : 대문자로 변환

lower() : 소문자로 변환

strip() : 양쪽 공백 제거

lstrip() : 왼쪽 공백 제거

rstring() : 오른쪽 공백 제거

replace() : 특정 문자열 치환

split() : 특정 문자열로 분리하여 반환

join() : 문자열 리스트를 결합

주요 문자열 메서드 (method)

메서드 -1

```
a = "apple"
print(a.count("p"))
print(a.find("p"))
print(a.index("p"))
print(".".join(a))
a = a.upper()
print(a)
print(a.lower())
```

Result

```
2
1
1
a.p.p.l.e
APPLE
apple
```

주요 문자열 메서드 (method)

메서드 -2

```
b = " How can I improve my  
coding skills? "  
print(b)  
b = b.strip()  
print(b)  
b = b.replace("?", "")  
print(b)  
word_list = b.split(" ")  
print(word_list)
```

Result

```
How can I improve my coding skills?  
How can I improve my coding skills?  
How can I improve my coding skills  
['How', 'can', 'I', 'improve', 'my',  
'conding', 'skills']
```

연습 문제!

- 1) Mary's cosmetics 을 출력하세요.
- 2) "dk2jd923i1jdk2jd93jfd92"의 길이를 구하세요.
- 3) t1 = 'python', t2 = 'java'일 때 문자열 더하기와 곱하기를 이용하여 "python java python java python java"를 출력 하세요.
- 4) id = "890910-1157963"에서 성별을 나타내는 수를 출력하세요.
- 5) license_plate = "24가 2210"에서 번호판 뒷자리만 출력하세요.
- 6) url = portal.ac.kr 에서 kr만 출력하세요. (split 함수 사용)

04 리스트 (List)

◆ 리스트의 정의

순서가 있는 데이터들의 집합을 나타내는 데이터 타입

◆ 리스트의 정의

대괄호([])를 사용하여 선언

리스트의 원소로는 모든 데이터 타입 설정 가능

list 선언

```
a = ["deep", "and", "high"]
b = [1, 2, [3, 4]]
c = [1, "deep", True]
print(type(a))
print(a, b, c)
```

Result

```
<class 'list'>
['deep', 'and', 'high'][1, 2, [3, 4]][1, 'deep', True]
```

04 리스트 (List)

◆ 인덱스와 인덱싱

- index: 리스트에서 **요소의 위치**
- indexing: **특정 위치의 요소**를 가져오는 것
- [] 안에 인덱스 번호를 넣어 표현

ex) a[0], a[1], a[-1] ...

- 첫번째 요소부터 위치를 부여 한다면 인덱스 번호가 양수, 마지막 요소부터 위치를 부여한다면 인덱스 번호가 음수

리스트 → ["deep", "and", "high"]

index → 0 1 2

→ -3 -2 -1

list 인덱싱

```
a = ["deep", "and", "high"]
print(a[1], a[-1])
a[-2] = "pdj"
print(a)
```

Result

```
and high
['deep', 'pdj', 'high']
```

04 리스트 (List)

◆ 슬라이싱

- 문자열의 특정 부분을 가져오는 것
- [start:end:stride] 형태로 표현

ex) a[1:8:2], a[1:3], a[:3], a[-3:] ...

list 슬라이싱

```
a = ["deep", "and", "high"]  
print(a[1:3])  
print(a[:])  
print(a[::-2])
```

Result

```
['and', 'high']  
['deep', 'and', 'high']  
['high', 'deep']
```

04 리스트 (List)

리스트 메서드 (method)

리스트에 여러 가지 변환을 가하는 데에 사용

- append() : 가장 마지막에 데이터 추가
- sort() : 오름차순 정렬
- reverse() : 순서 뒤집기
- insert() : 특정 위치에 데이터 추가
- remove() : 해당되는 데이터 값을 삭제
- pop() : 가장 마지막 값을 반환하고 마지막 값을 삭제
- extend() : 가장 마지막에 데이터 추가
(데이터 내부 원소를 추가)

04 리스트 (List)

◆ 리스트 메서드 (method)

append()

```
a = ["deep", "and", "high", "learning"]  
a.append('fighting!')  
print(a)
```

Result

```
['deep', 'and', 'high', 'learning', 'fighting!']
```

04 리스트 (List)

◆ 리스트 메서드 (method)

reverse()

```
a = ["deep", "and", "high", "learning"]  
a.reverse()  
print(a)
```

Result

```
['learning', 'high', 'and', 'deep']
```

04 리스트 (List)

◆ 리스트 메서드 (method)

sort()

```
a = ["deep", "and", "high", "learning"]  
a.sort()  
print(a)
```

Result

```
['and', 'deep', 'high', 'learning']
```

04 리스트 (List)

◆ 리스트 메서드 (method)

index()

```
a = ["deep", "and", "high", "learning"]  
print(a.index('high'))
```

Result

2

04 리스트 (List)

◆ 리스트 메서드 (method)

insert()

```
a = ["deep", "and", "high", "learning"]  
a.insert(2, 'index1')  
a.insert(6, 'index2')  
print(a)
```

Result

```
['deep', 'and', 'index1', 'high', 'learning', 'index2']
```

04 리스트 (List)

◆ 리스트 메서드 (method)

remove()

```
a = ["deep", "and", "high", "learning"]  
a.remove('index1')  
print(a)
```

Result

```
['deep', 'and', 'learning']
```

04 리스트 (List)

◆ 리스트 메서드 (method)

pop()

```
a = ["deep", "and", "high", "learning"]  
print(a.pop())
```

Result

learning

04 리스트 (List)

◆ 리스트 메서드 (method)

del

```
a = ["deep", "and", "high", "learning"]  
del a[2]  
print(a)
```

Result

```
['deep', 'and', 'learning']
```


04 리스트 (List)

리스트 메서드 (method)

extend()

```
a = ["deep", "and", "high", "learning"]  
b = ['artificial', 'intelligence']  
a.extend(b)  
print(a)
```

Result

```
['deep', 'and', 'high', 'learning', 'artificial', 'intelligence']
```

04 리스트 (List)

◆ 리스트 메서드 (method)

append()와의 비교

```
a = ["deep", "and", "high", "learning"]  
b = ['artificial', 'intelligence']  
a.append(b)  
print(a)
```

Result

```
['deep', 'and', 'high', 'learning', ['artificial', 'intelligence']]
```

04 리스트 (List)

◆ 리스트 원소 수정

리스트 원소 수정

```
a = [1, 2, 3, 4, 5]
a[1] = "Deep"
print(a)
a[1:4] = "Hear"
print(a)
```

Result

```
[1, 'Deep', 3, 4, 5]
[1, 'H', 'e', 'a', 'r', 5]
```

04 리스트 (List)

◆ 리스트 복사 : 얇은 복사와 깊은 복사

`b = a` 형태로 복사를 했을 때, `a`를 변경하게 되면 `b`도 변경됨 (얇은 복사)

별도의 저장공간을 가지게 하려면 `copy()` 함수 사용 (깊은 복사)

얇은 복사

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

Result

```
[1, 2, 3] [1, 2, 3]
[1, 2, 4] [1, 2, 4]
```

`a`를 수정했지만 `b` 또한 함께 수정됨

04 리스트 (List)

◆ 리스트 복사 : 얇은 복사와 깊은 복사

`b = a` 형태로 복사를 했을 때, `a`를 변경하게 되면 `b`도 변경됨 (얇은 복사)

별도의 저장공간을 가지게 하려면 `copy()` 함수 사용 (깊은 복사)

깊은 복사

```
a = [1, 2, 3]
c = a.copy()
print(a, c)
a[2] = 5
print(a, c)
```

Result

```
[1, 2, 3] [1, 2, 3]
[1, 2, 5] [1, 2, 3]
```

`a`만 변경되고 `c`는 불변

04 리스트 (List)

연습 문제!

1) language1 = ["C", "C++", "JAVA"], language2 = ["Python", "Go", "C#"]

두 리스트의 원소를 모두 갖는 languages를 만드세요.

2) nums = [12, 245, 33, 77, 858]의 평균을 구하세요.

3) a = ["b", "a", "d", "c"] 리스트를 알파벳 순으로 정렬하세요.

05 튜플 (Tuple)

◆ 튜플의 정의

- List와 같이 순서가 있는 데이터 타입이지만 데이터를 변경할 수 없음
- List보다 컴퓨터의 자원(메모리)를 적게 사용

◆ 튜플의 선언

- **''(콤마)**로 구분하여 선언. 또는 **''**로 구분하고 괄호로 묶어서 선언
ex) a = 1, 2, 3 or b = ('가', '나', '다')
- 원소로는 **모든 데이터 타입** 설정 가능
- 인덱싱과 슬라이싱 가능

05 튜플 (Tuple)

◆ 튜플의 선언

Tuple 선언

```
a = 1, 2, 3, 4
b = "deep", "and", "high", "learning"
c = (1, "fast", True)
print(type(a), type(b), type(c))
print(a, b, c)
```

Result

```
<class 'tuple'> <class 'tuple'> <class 'tuple'>
(1, 2, 3, 4)('deep', 'and', 'high', 'learning') (1, 'fast', True)
```


05 튜플 (Tuple)

◆ 튜플의 활용

인덱싱과 슬라이싱

```
a = 1, 2, 3, 4  
print(a[2], a[1:3])
```

Result

```
3 (2, 3)
```

데이터 수정 불가

```
a = 1, 2, 3, 4  
a[1] = 3
```

Result

```
TypeError: 'tuple' object does not  
support item assignment
```

05 튜플 (Tuple)



튜플의 활용

리스트와 메모리 크기 비교

```
import sys
#getsizeof의 단위 : byte
ls = [1, 2, 3, 4, 5]
print(type(ls), sys.getsizeof(ls), "byte")
t = tuple(ls)
print(type(t), sys.getsizeof(t), "byte")
```

Result

```
<class 'list'> 96 byte
<class 'tuple'> 80 byte
```

06 딕셔너리 (Dictionary)

◆ 딕셔너리의 정의

key와 value의 쌍으로 데이터가 순서 없이 모여 있는 데이터 타입

◆ 딕셔너리의 선언

{ } 기호 안에 '키 : 값' 형태로 선언 ({키 : 값})

키 값으로 정수나 문자열의 데이터 타입 사용 가능

dictionary 선언

```
dic = {  
    1: "one",  
    "A": ["data", "science"],  
    "숫자": 1234,  
}  
print(type(dic))  
print(dic)
```

Result

```
<class 'dict'>  
{1: 'one', 'A': ['data', 'science'],  
'숫자': 1234}
```

06 딕셔너리 (Dictionary)

◈ 딕셔너리 데이터 수정

키 값으로 데이터 수정 가능

dictionary 데이터 수정

```
dic = {  
    1: "one",  
    "A": ["data", "science"],  
    "숫자": 1234,  
}  
print(dic["숫자"])  
dic[1] = "하나"  
dic["A"] = "알파벳"  
print(dic)
```

Result

```
1234  
{1: '하나', 'A': '알파벳', '숫자': 1234}
```

06 딕셔너리 (Dictionary)

딕셔너리 데이터 삭제

del을 사용하여 삭제 (키 값으로 접근)

dictionary 데이터 삭제

```
dic = {  
    1: "one",  
    2: "two",  
}  
print(dic)  
  
del dic[1]  
print(dic)
```

Result

```
{1: 'one', 2: 'two'}  
{2: 'two'}
```

06 딕셔너리 (Dictionary)

딕셔너리 메서드 (method)

딕셔너리 메서드를 이용해 딕셔너리 데이터를 가공

keys() : 키를 반환

values() : 값을 반환

items() : 키와 값을 리턴

clear() : dictionary 데이터를 모두 삭제

get() : 매개변수에 해당하는 값을 반환

copy() : 리스트와 마찬가지로

다른 저장공간을 가지는 데이터 대입

06 딕셔너리 (Dictionary)

◆ 딕셔너리 메서드 (method)

딕셔너리 값 접근 및 전체 삭제

메서드 -1

```
dic = {  
    1: 'one',  
    "A": ["deep", "learning"],  
    "숫자": 1234,  
}  
print(dic.keys())  
print(dic.values())  
print(dic.items())  
dic.clear()  
print(dic)
```

Result

```
dict_keys([1, 'A', '숫자'])  
dict_values(['one', ['deep',  
'learning'], 1234])  
dict_items([(1, 'one'), ('A', ['deep',  
'learning']), ('숫자', 1234)])  
{}
```

06 딕셔너리 (Dictionary)

◆ 딕셔너리 메서드 (method)

깊은 복사를 위해서는 copy 함수 사용

메서드 -2

```
dic = {  
    1: 'one',  
    "A": ["deep", "learning"],  
    "숫자": 1234,  
}  
dic2 = dic  
dic3 = dic.copy()  
dic[1] = "하나"  
print(dic)  
print(dic2)  
print(dic3)
```

Result

```
{1: '하나', 'A': ['deep', 'learning'],  
'숫자': 1234}  
{1: '하나', 'A': ['deep', 'learning'],  
'숫자': 1234}  
{1: 'one', 'A': ['deep', 'learning'],  
'숫자': 1234}
```


06 딕셔너리 (Dictionary)

연습 문제!

```
name_to_age = {"Jenny": 20, "Ella": 31}
```

name_to_age의 key는 이름, value는 나이를 나타냅니다.

- 1) name_to_age에 26살의 John, 29살의 Tom에 대한 정보를 추가하세요.
- 2) Jenny의 나이를 21살로 바꾸세요.
- 3) name_to_age의 구성원들이 가지는 나이를 전부 출력하세요.

집합의 정의 및 특성

중복되는 데이터가 없는 데이터 타입

교집합, 합집합, 차집합과 같은 집합의 연산 가능

리스트 데이터에서 중복을 제거할 때 사용

딕셔너리와 같이 순서가 없는 데이터 타입

특정 인덱스 값을 가져오거나 슬라이싱으로 데이터 수정이 불가능

07 집합 (Set)

◆ 집합의 선언

집합은 리스트 형태의 데이터에 `set()`으로 **형변환**을 해주는 방법으로 선언
중복된 데이터는 제거됨

집합의 선언

```
ls = [1, 2, 3, 4, 5, 1, 2, 3]
s = set(ls)
print(type(s), s)
```

Result

```
<class 'set'> {1, 2, 3, 4, 5}
```

07 집합 (Set)

◆ 집합의 연산

교집합 (intersection)

```
s1 = set([1, 2, 3, 4])  
s2 = set([3, 4, 5, 6])  
print(s1 & s2)  
print(s1.intersection(s2))
```

Result

```
{3, 4}  
{3, 4}
```

07 집합 (Set)

◆ 집합의 연산

합집합 (union)

```
s1 = set([1, 2, 3, 4])  
s2 = set([3, 4, 5, 6])  
print(s1 | s2)  
print(s1.union(s2))
```

Result

```
{1, 2, 3, 4, 5, 6}  
{1, 2, 3, 4, 5, 6}
```

07 집합 (Set)

◆ 집합의 연산

차집합 (difference)

```
s1 = set([1, 2, 3, 4])  
s2 = set([3, 4, 5, 6])  
print(s1 - s2)  
print(s1.difference(s2))
```

Result

```
{1, 2}  
{1, 2}
```

07 집합 (Set)

◆ 집합의 선언

list로 형변환 후 다시 set으로 변환

집합의 형변환

```
ls = [1, 2, 3, 4, 5, 1, 2, 3]
s = set(ls)
s = list(s)
s[4] = 10
s = set(s)
print(s)
```

Result

```
{1, 2, 3, 4, 10}
```

08 Mutable vs Immutable

Mutable과 Immutable의 정의

Immutable : (값이) 변하지 않는

Mutable : (값이) 변하는

각 자료형의 소속

Immutable : 숫자(number), 문자열(string), 튜플(tuple), 논리(Boolean)

Mutable : 리스트(list), 딕셔너리(dictionary), 집합(set)

08 Mutable vs Immutable

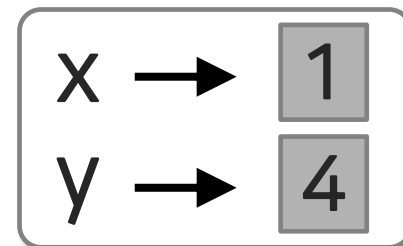
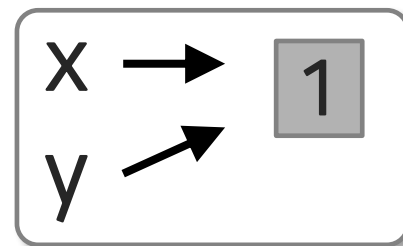
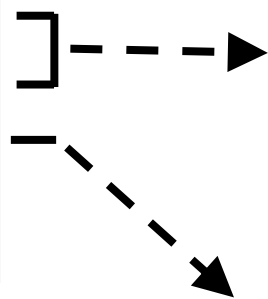
Mutable와 Immutable의 정의

Immutable

```
x = 1  
y = x  
y += 3  
print(x, y)
```

Result

14



08 Mutable vs Immutable

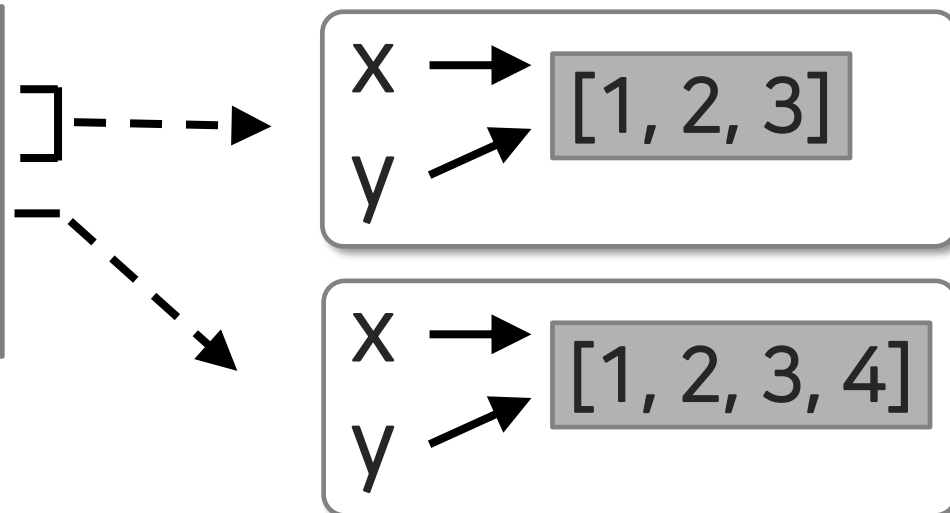
Mutable와 Immutable의 정의

Mutable

```
x = [1, 2, 3]
y = x
y += [4]
print(x, y)
```

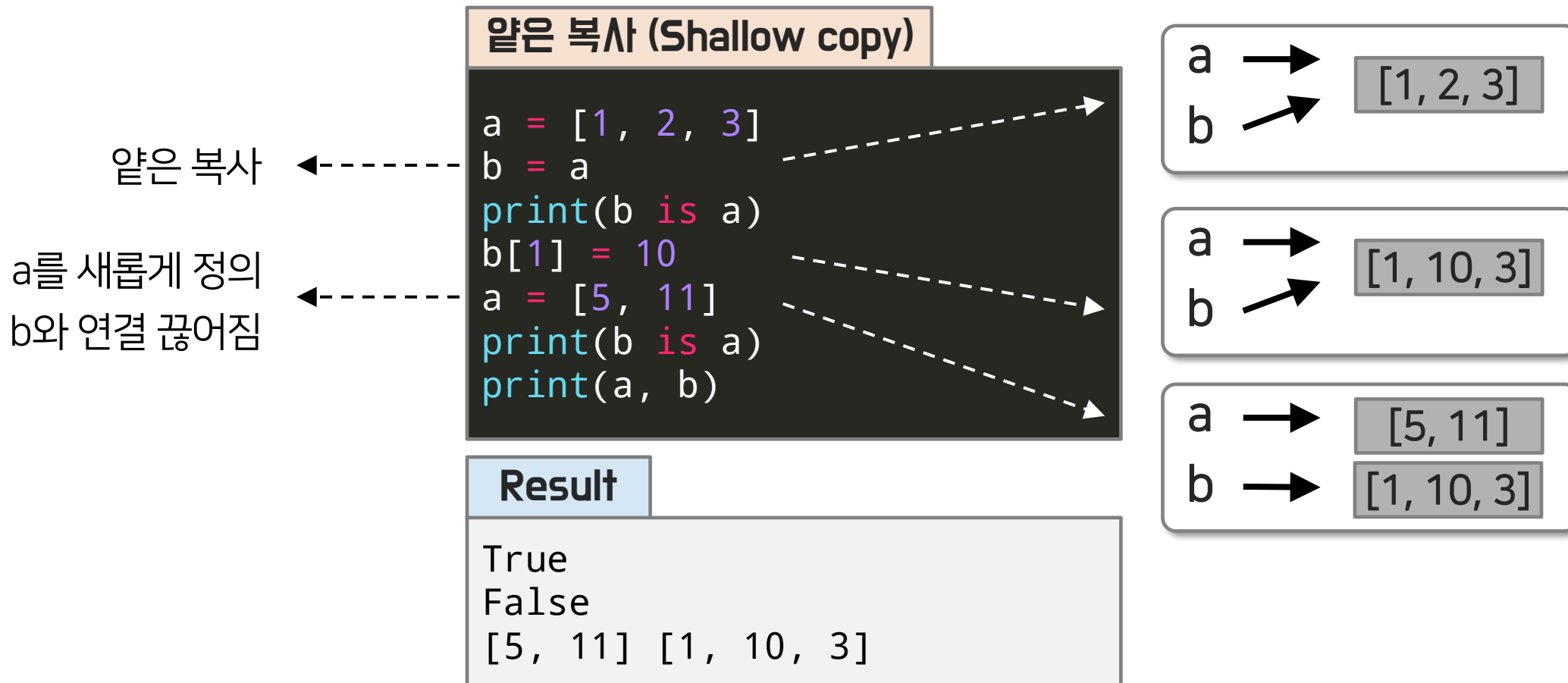
Result

[1, 2, 3, 4] [1, 2, 3, 4]



08 Mutable vs Immutable

◈ 얕은 복사와 깊은 복사의 정확한 이해



08 Mutable vs Immutable

◈ 얇은 복사와 깊은 복사의 정확한 이해

얇은 복사 (Shallow copy)

```
a = [1, 2, 3]
b = a.copy()
print(b is a)
b[1] = 10
a = [5, 11]
print(b is a)
print(a, b)
```

깊은 복사
a와 b는 애초에 다름

a를 새롭게 정의

Result

```
False
False
[5, 11] [1, 10, 3]
```

a → [1, 2, 3]

b → [1, 2, 3]

a → [1, 2, 3]

b → [1, 10, 3]

a → [5, 11]

b → [1, 10, 3]

08 Mutable vs Immutable

◈ 얕은 복사와 깊은 복사의 정확한 이해

Mutable -list

```
a = [1, 2, 3]
b = a
print(a is b)
```

```
a = [1, 2, 3]
b = [1, 2, 3]
print(a is b)
```

```
a = [1, 2, 3]
b = a.copy()
print(a is b)
```

Result

```
True
False
False
```

08 Mutable vs Immutable

◈ 얕은 복사와 깊은 복사의 정확한 이해

Mutable -set

```
a = set([1, 2, 3])  
b = a  
print(a is b)  
  
b |= set([4, 5])  
print(b)  
print(a is b)
```

Result

```
True  
{1, 2, 3, 4, 5}  
True
```

08 Mutable vs Immutable

◈ 얕은 복사와 깊은 복사의 정확한 이해

Mutable -dictionary

```
a = {"a":1, "b":2}
b = a
print(a is b)
```

```
a = {"a":1, "b":2}
b = {"a":1, "b":2}
print(a is b)
```

```
a = {"a":1, "b":2}
b = a.copy()
print(a is b)
```

Result

```
True
False
False
```

08 Mutable vs Immutable

◈ 얕은 복사와 깊은 복사의 정확한 이해

Immutable -tuple

```
a = (1, 2, 3)
b = a
print(a is b)
```

```
a = (1, 2, 3)
b = (1, 2, 3)
print(a is b)
```

tuple은 copy 메소드 없음

Result

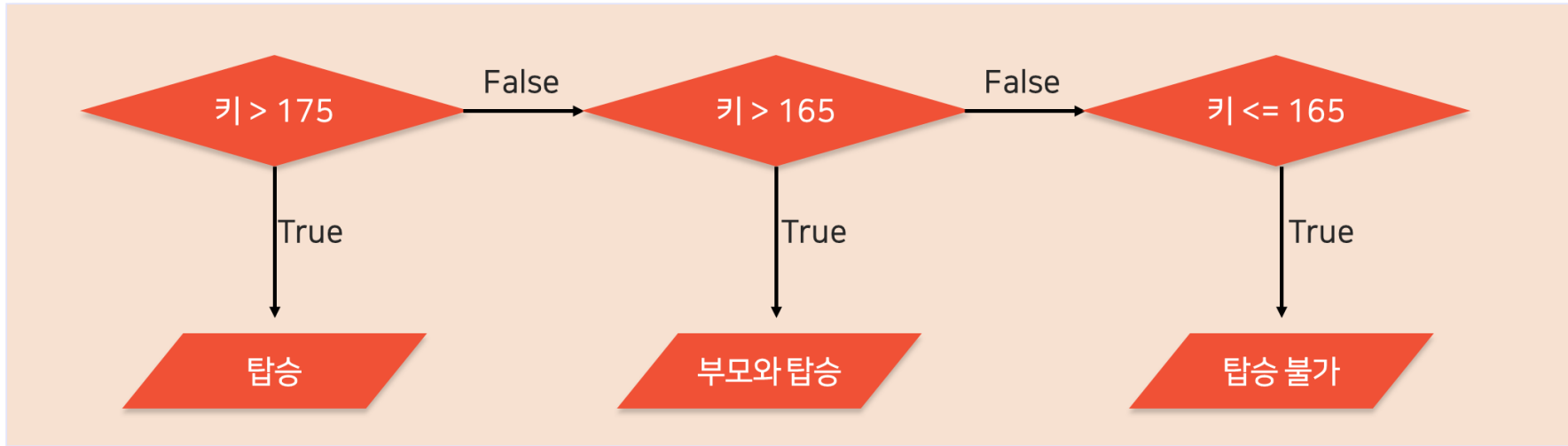
True
False

◇ 조건문의 정의

조건에 따라 다른 코드를 실행할 수 있는 제어문

의도한 상황에 따라 자유롭게 분기 가능

if, elif, else 등의 키워드로 구현됨



◆ 조건문의 용법 - if/else

if/else 선언

```
if height > 175:  
    print("175cm 이상입니다. 탑승하세요")  
  
else:  
    print("고객님은 탑승할 수 없습니다.")
```

Result (height=180)

175cm 이상입니다. 탑승하세요.

Result (height=160)

고객님은 탑승할 수 없습니다.

else문은 if문 뒤에 위치할 수 있으며 단독으로는 사용할 수 없음

◆ 조건문의 용법 - elif

elif 추가

```
if height > 175:
    print("175cm 이상입니다. 탑승하세요")

if height > 165:
    print("부모님과 탑승하세요.")

else:
    print("고객님은 탑승할 수 없습니다.")
```

Result (height=170)

부모님과 탑승하세요.

Result (height=160)

고객님은 탑승할 수 없습니다.

elif문은 if문 뒤에 위치할 수 있으며 단독으로는 사용할 수 없음

◇ 조건문의 용법 - 중첩

if문 중첩

```
if height > 175:
    print("175cm 이상입니다. 탑승하세요")

if height > 165:
    if with_parent:
        print("부모님과 탑승하세요.")
    else:
        print("부모님 모셔오세요.")
else:
    print("고객님은 탑승할 수 없습니다.")
```

Result (height=170, with_parent=True)

부모님과 탑승하세요.

Result (height=170, with_parent=False)

부모님 모셔오세요.

Result (height=160, with_parent=True)

고객님은 탑승할 수 없습니다.

조건문의 용법 - 한 줄 조건문

한 줄 조건문

```
print("175cm 이상입니다. 탑승하세요.") if height > 175 else print("탑승할 수 없습니다.")
```

Result (height=180)

175cm 이상입니다. 탑승하세요.

else문은 필요하지 않다면 적지 않을 수 있음

반복문의 정의

프로그램이 특정 부분을 반복하여 실행하도록 제어하는 명령문
`for, while` 등의 키워드로 구현됨

```
while (반복 조건):  
    반복할 코드 블록
```

```
for 변수 in 시퀀스 데이터:  
    반복할 코드 블록
```

◈ 반복문의 용법 -while

while 반복문

```
a = 3
b = []
while a:
    a -= 1
    b.append(a)
    print(a)
print(b)
```

Result

```
2
1
0
[2, 1, 0]
```

조건으로 들어간 변수 **a**가 0(False)가 되는 시점에 while 반복문 종료

◈ 반복문의 용법 -continue

continue 제어

```
a = 3
b = []
while a:
    a -= 1
    if a == 2:
        continue
    b.append(a)
    print(a)
print(b)
```

Result

```
1
0
[1, 0]
```

continue를 만나면 반복문 안의 코드를 실행하지 않고 다시 반복문 코드 라인으로 이동

◈ 반복문의 용법 -break

break 제어

```
a = 3
b = []
while a:
    a -= 1
    if a == 2:
        break
    b.append(a)
    print(a)
print(b)
```

Result

```
4
3
2
[4, 3, 2]
```

break를 만나면 가장 가까운 반복문은 탈출하고 해당 반복이 끝남

10 반복문

◈ 반복문 활용 예시 -while

반복문 활용

```
a = 10
b = []
while a:
    if a == 2:
        break
    elif a % 2 == 1:
        a -= 1
        continue
    else:
        b.append(a)
        a -= 1
        print(a)
print(b)
```

Result

```
4
3
2
[4, 3, 2]
```

반복문의 **탈출 조건**을 적절히 선언하지 않으면
무한 loop에 빠질 수 있음

◈ 반복문의 용법 -for

리스트 순회

```
a = [1, 2, 3, 4, 5, 6, 7, 8]
for number in a:
    print(number)
```

Result

1
2
3
4
5
6
7
8

리스트와 같은 시퀀스 데이터에 하나씩 접근하여 원하는 코드를 반복할 때 쓰임

◈ 반복문의 용법 -for

딕셔너리 순회

```
dic = {"수학": 100, "영어": 90}
for key, val in dic.items():
    if val == 100:
        print(key, val, "굳굳")
    else:
        print(key, val, "분발하세요!")
```

Result

```
수학 100 굳굳
영어 90 분발하세요!
```

딕셔너리를 순회하며 key와 value에 접근할 수 있음

10 반복문

◈ 반복문의 용법 -for

range 함수를 활용해 for문을 선언할 수 있음

range 함수 활용 반복문

```
for val in range(1, 10, 1):  
    if val % 2 == 0:  
        print(val, "-> 짝수입니다.")
```

Result

```
2 -> 짝수입니다.  
4 -> 짝수입니다.  
6 -> 짝수입니다.  
8 -> 짝수입니다.
```

range(1, 10, 1) 함수의 첫번째 인자 1부터 두번째 인자 10미만까지
세번째 인자인 1만큼 늘려가며 반복

◈ 반복문의 용법 - enumerate

enumerate 활용 반복문

```
subjects = ["수학", "영어", "국어"]  
for index, val in enumerate(subjects):  
    print(index, val)
```

Result

```
0 수학  
1 영어  
2 국어
```

enumerate 함수를 활용해 리스트의 인덱스와 값에 동시에 접근 가능

◈ 반복문의 용법 -zip

zip 함수를 활용해 여러 리스트의 값을 묶어 순회 할 수 있음

zip 활용 반복문

```
subjects = ["수학", "영어", "국어", "탐구"]  
scores = [100, 90, 80]  
for subj, scr in zip(subjects, scores):  
    print(subj, scr)
```

Result

수학	100
영어	90
국어	80

묶이는 리스트의 길이가 다를 경우, 가장 짧은 리스트 기준으로 묶고 남은 데이터를 버림

❖ 반복문의 용법 -리스트 내포

반복문의 결과를 바로 리스트로 생성하는 방법

리스트 내포

```
ls = [num*10 for num in range(1, 8)]  
print(ls)
```

Result

```
[10, 20, 30, 40, 50, 60, 70]
```

단순히 for문을 반복하여 append 하는 방식보다 빠름 (function call 최소화)

10 반복문

◈ 반복문의 용법 -리스트 내포

리스트 안에 반복문과 함께 조건문을 활용할 수 있음

리스트 내포 조건문

```
ls = [num*10 for num in range(1, 6) if num % 2 == 0]  
print(ls)
```

Result

[20, 40]

1~5까지 순회하며 짝수인 경우 10을 곱해 리스트에 저장하는 코드

연습 문제

- 1) while문을 사용해 1부터 1000까지의 자연수 중 3의 배수의 합을 구하세요.
- 2) while문을 사용해 다음과 같이 *를 출력해보세요.

```
*  
**  
***  
****  
*****
```

- 3) 아래 코드를 리스트 내포를 이용해 한 줄로 구현해보세요.

```
numbers = [1, 2, 3, 4, 5]  
result = []  
for n in numbers:  
    if n % 2 == 0:  
        result.append(n + 2)
```

11 화면 입출력

◈ 화면 출력

print() 함수

```
print("Hello World!")
```

Result

Hello world!

(쌍)따옴표 안의 문자열은 자유롭게 작성 가능

11 화면 입출력

◈ 화면 출력

print() 함수

```
print("Hello World!", end = " ")  
print("This is beautiful!")
```

Result

Hello world! This is beautiful!

end: 출력의 맨 끝 문자를 정의함 (기본 값은 \n)

11 화면 입출력

◈ 화면 출력

print() 함수

```
print("Hello", "World!")  
print("Hello", "World!", sep = "#")
```

Result

```
Hello world!  
Hello#world!
```

sep: 개체 출력 값 사이의 문자 정의 (기본 값은 공백 ' ')

11 화면 입출력

◈ 화면 출력

Input() 함수

```
a = input()  
print("당신의 이름은 ", a)
```

Result (입력: kim)

당신의 이름은 kim

input 함수로 입력받은 문자열을 변수 a에 저장하여 print 함수로 출력

❖ 화면 출력 - 이스케이프 문자

₩(역슬래시)와 특정 문자가 합쳐진 형태로 출력 시 특별한 의미를 나타냄

print() 함수

```
print('\\', \", \\, \t, \n')
```

Result

' , " , \ , ,

₩' : 홑따옴표, ₩" : 쌍따옴표, ₩n : 줄바꿈, ₩t : 탭문자 (일정 간격), ₩₩ : 역슬래시

11 화면 입출력

◈ 화면 출력 - 자료형별 출력 서식

출력 문자열에 **자료형에 대응하는 서식**을 순서대로 넣고, 이후 %() 에 원래 변수 작성

print() 함수

```
a = 100
b = 200
c = 0.5
print("%d" %a)
print("%d %d %f" %(a,b,c))
```

Result

```
100 100 200 0.500000
```

%d : 정수, %f : 실수, %c : 1글자 문자, %s : 2글자 이상 문자열

11 화면 입출력

❖ 화면 출력 - 자료형별 출력 서식

출력 문자열에 **자료형에 대응하는 서식**을 순서대로 넣고, 이후 %() 에 원래 변수 작성

print() 함수

```
a = "아메리카노"  
b = 2000  
  
print("%s 한 잔의 가격은 %d원입니다." %(a, b))
```

Result

아메리카노 한 잔의 가격은 2000원입니다.

%d : 정수, %f : 실수, %c : 1글자 문자, %s : 2글자 이상 문자열

11 화면 입출력

❖ 화면 출력 - 여백을 맞추어 출력하기 (정수)

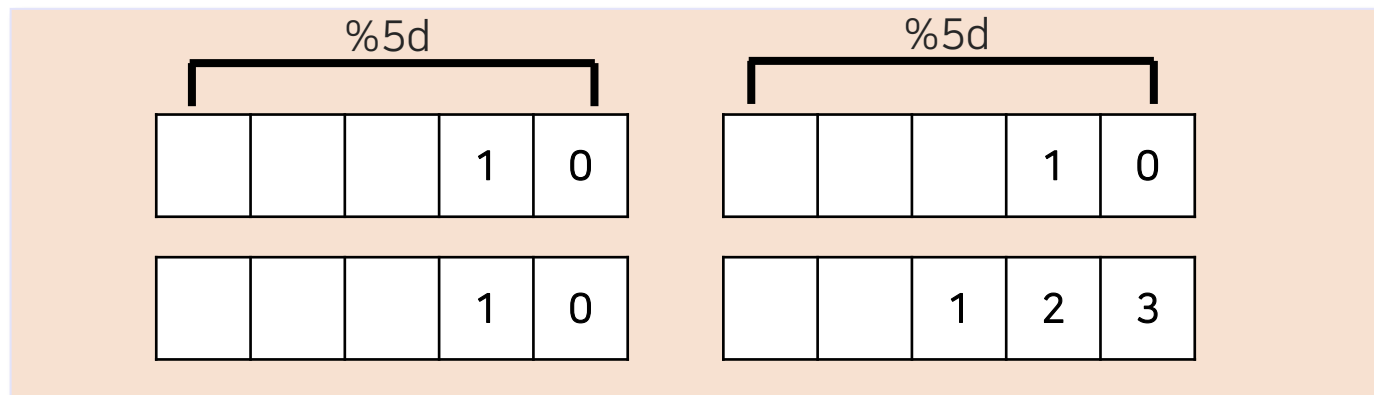
%와 문자 사이에 숫자를 입력하여 출력값 앞쪽으로 여백 생성

print() 함수

```
a = 10  
b = 123  
print("%5d %5d" %(a, a))  
print("%5d %5d" %(a, b))
```

Result

```
10    10  
    10  123
```



11 화면 입출력

❖ 화면 출력 - 여백 및 소수점 맞추어 출력하기 (실수)

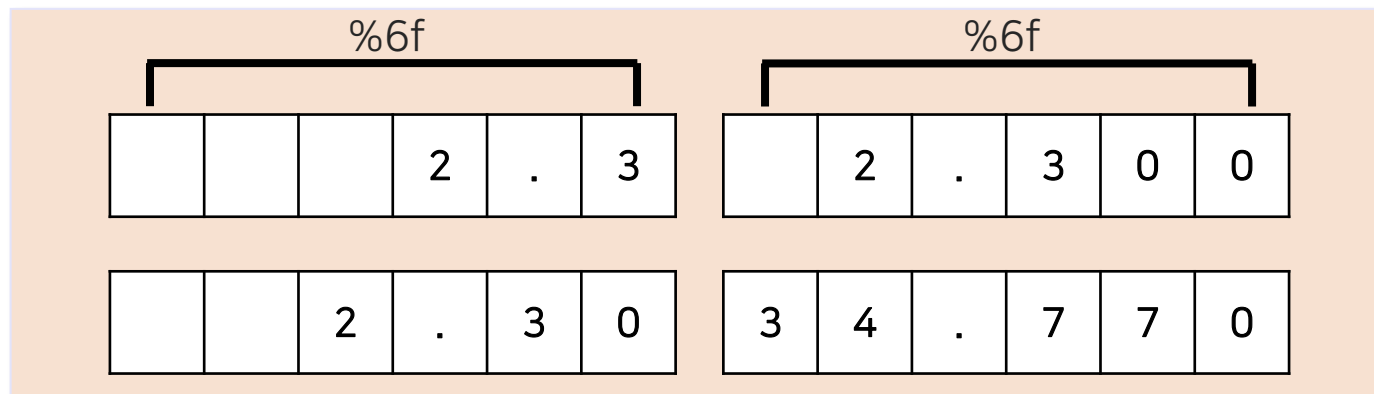
%와 문자 사이에 숫자를 입력하여 출력값 앞으로 여백 생성

print() 함수

```
a = 2.3  
b = 34.77  
print("%6.1f %6.3f" %(a, a))  
print("%6.2f %6.3f" %(a, b))
```

Result

```
2.3 2.300  
2.3034.770
```



11 화면 입출력

❖ 화면 출력 - format()을 이용한 출력

format 함수는 중괄호 {, } 안에 인덱스를 지정하고 인자로 값들을 삽입

format() 함수

```
a = 2
b = 3
s = "구구단 {0} x {1} = {2}".format(a, b,
a*b)
print(s)
```

```
{0} ← a
{1} ← b
{2} ← a*b
```

Result

구구단 2 x 3 = 6

11 화면 입출력

❖ 화면 출력 - format()을 이용한 출력

format 함수 인자로 문자, 변수, 직접 지정 등 사용 가능

format() 함수 -1

```
s1 = 'name: {0}'.format('Deeplearning')  
print(s1)
```

Result

name: Deeplearning

format() 함수 -2

```
age = 55  
print('age: {0}'.format(age))
```

Result

age: 55

format() 함수 -3

```
s1 = 'number: {num}, gender:  
{gen}'.format(num=111, gen="여")  
print(s1)
```

Result

number : 111, gender : 여

11 화면 입출력

❖ 화면 출력 - format()을 이용한 출력

인덱스가 없으면 순서대로 포매팅. 순서가 바뀌거나 중복도 허용

format() 함수 -1

```
s4 = 'name: {}, city: {}'.format('Kim', 'seoul')
print(s4)
```

Result

```
name: Kim, city: seoul
```

format() 함수 -2

```
s5 = 'song1: {1}, song2: {0}'.format('love yourself', 'shape of you')
print(s5)
```

Result

```
song1: shape of you, song2: love yourself
```

format() 함수 -3

```
s6 = 'test1: {0}, test2: {1}, test3: {0}'.format('인덱스0', '인덱스1')
print(s6)
```

Result

```
test1: 인덱스0, test2: 인덱스1, test3: 인덱스0
```

11 화면 입출력

❖ 화면 출력 - format()을 이용한 출력

'<': 왼쪽 정렬, '>': 오른쪽 정렬, '^': 가운데 정렬

format() 함수 -왼쪽정렬

```
s9 = "this is {0:<10} | done {1:<5} |".format('left', 'a')  
print(s9)
```

Result

```
this is left      | done a      |
```

❖ 화면 출력 - format()을 이용한 출력

'<': 왼쪽 정렬, '>': 오른쪽 정렬, '^': 가운데 정렬

format() 함수 -오른쪽정렬

```
s10 = "this is {0:>10} | done {1:>5} |".format('right', 'b')  
print(s10)
```

Result

```
this is      right | done      b |
```


11 화면 입출력

❖ 화면 출력 - format()을 이용한 출력

'<': 왼쪽 정렬, '>': 오른쪽 정렬, '^': 가운데 정렬

format() 함수 - 가운데정렬

```
s11 = "this is {0:^10} | done {1:^5} |".format('center', 'c')  
print(s11)
```

Result

```
this is   center   | done   c   |
```

❖ 화면 출력 - f-string을 이용한 출력 (>=python3.6)

문자열 맨 앞에 f를 붙이고, 출력할 변수와 값을 중괄호 안에 삽입

f-string

```
s = 'study'
n = 5
result1 = f'{s}를 좋아합니다. 하루 {n}시간 합니다.'
print(result1)
```

Result

study를 좋아합니다. 하루 5시간 합니다.

11 화면 입출력

❖ 화면 출력 - f-string을 이용한 출력 (>=python3.6)

변수 뒤에 :<, :^, :> 를 붙여서 정렬

f-string (왼쪽 정렬)

```
s1 = 'left'
result1 = f'|{s1:<10}|'
print(result1)
```

Result

```
|left          |
```

f-string (오른쪽 정렬)

```
s2 = 'right'
result1 = f'|{s2:>10}|'
print(result1)
```

Result

```
|          right|
```

f-string (가운데 정렬)

```
s3 = 'mid'
result1 = f'|{s3:^10}|'
print(result1)
```

Result

```
|      mid      |
```

12 파일 입출력

◆ 파일 입력

- open() 함수와 close() 함수를 이용하여 파일을 열고 닫음
- read() 함수를 이용하여 파일 전체를 읽어옴

읽을 파일 (txtio.txt)

```
Hello, world!  
My name is Hee-Soo Kim.  
Nice to meet you.
```

r: 읽기 모드

open() / read() 함수

```
f = open(".txtio.txt", "r", encoding='utf-8')  
line = f.read()  
print(line)  
f.close()
```

Result

```
Hello, world!  
My name is Hee-Soo Kim.  
Nice to meet you.
```

12 파일 입출력

◆ 파일 입력

- open() 함수와 close() 함수를 이용하여 파일을 열고 닫음
- readline() 함수를 이용하여 파일 한 문장을 읽어옴

읽을 파일 (txtio.txt)

```
Hello, world!  
My name is Hee-Soo Kim.  
Nice to meet you.
```

r: 읽기 모드

open() / readline() 함수

```
f = open(".txtio.txt", "r", encoding='utf-8')  
line = f.readline()  
print(line)  
f.close()
```

Result

Hello, world!

12 파일 입출력

◆ 파일 입력

- readline() 함수와 반복문을 이용하여 파일 전체를 한 문장씩 읽어옴

읽을 파일 (txtio.txt)

```
Hello, world!  
My name is Hee-Soo Kim.  
Nice to meet you.
```

r: 읽기 모드

open() / readline() 함수

```
f = open(".txtio.txt", "r", encoding='utf-8')  
line = f.readline()  
while line:  
    print(line)  
    line = f.readline()  
f.close()
```

Result

Hello, world!

12 파일 입출력

◆ 파일 입력

- readlines() 함수로 **파일의 모든 line**을 리스트화하여 읽어옴

읽을 파일 (txtio.txt)

```
Hello, world!  
My name is Hee-Soo Kim.  
Nice to meet you.
```

r: 읽기 모드

open() / readlines() 함수

```
f = open(".txtio.txt", "r", encoding='utf-8')  
line = f.readlines()  
print(line)  
f.close()
```

Result

```
['Hello, world!\n', 'My name  
is Dohyun Kim.\n', 'Nice to  
meet you.\n']
```

12 파일 입출력

◆ 파일 입력

- 'w': **쓰기 모드** - 파일에 내용을 쓸 때 사용
- 'a': **추가모드** - 파일의 마지막에 새로운 내용을 추가할 때 사용

open() / write() 함수

```
music = ['Circle of life', 'Be prepared',  
         'The lion sleeps tonight', 'Hakuna Matata']  
  
with open('./Lion_king.txt', 'w') as f:  
    for a in music:  
        f.write(a+'\n')
```

출력 파일 (Lion_king.txt)

```
Circle of life  
Be prepared  
The lion sleeps tonight  
Hakuna Matata
```


12 파일 입출력

파일 입력

format(), f-string 모두 사용 가능

open() / write() 함수

```
music = ['Circle of life', 'Be prepared',  
         'The lion sleeps tonight', 'Hakuna Matata']  
  
with open('./Lion_king.txt', 'w',  
          encoding='utf-8') as f:  
    for a in music:  
        data = f'{a}\n'  
        f.write(a+'\n')
```

출력 파일 (Lion_king.txt)

```
Circle of life  
Be prepared  
The lion sleeps tonight  
Hakuna Matata
```

13 함수 (Function)

◆ 함수 (Function)의 정의

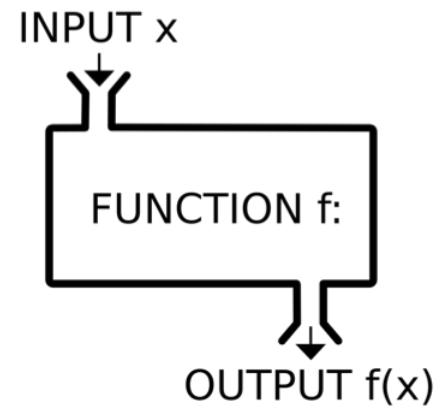
특정 작업을 수행하기 위해 독립적으로 설계된 코드의 집합

함수 선언

```
def data_science():  
    print('python')  
  
data_science()
```

Result

python



13 함수 (Function)

❖ 파라미터 (Parameter)와 아규먼트 (Argument)

파라미터 (parameter): 함수 호출 시 전달하는 데이터를 받아서 함수 내에서 사용되는 변수명

아규먼트 (Argument): 함수를 호출할 때의 인수

함수 선언 및 실행

```
def data_science(print_sentence):  
    print(print_sentence)  
  
string = "매개변수로 전달된 문자열 출력"  
data_science(string)
```

print_sentence: 파라미터 **string**: 아규먼트

Result

매개변수로 전달된 문자열 출력

13 함수 (Function)

❖ 디폴트 파라미터 (Default Parameter)

함수 호출시 파라미터에 대한 **아규먼트가 없으면 디폴트로 설정된 값이 파라미터 변수로 입력됨**

함수 선언 및 실행

```
def minus(num1=3, num2=2):  
    print(num1-num2)  
  
minus(2)
```

Result

0

num1은 argument로 넘긴 2, num2은 **default 값 2**

13 함수 (Function)

❖ 디폴트 파라미터 (Default Parameter)

함수 호출시 파라미터에 대한 **아규먼트가 없으면 디폴트로 설정된 값이 파라미터 변수로 입력됨**

에러 케이스

```
def minus(num1=3, num2=2, num3):  
    print(num1 - num2 - num3)  
  
minus(5)
```

Result

```
File "<ipython-input-14-4e76455cla92>", line 1  
def minus(num1=3, num2=2, num3):  
    ^  
SyntaxError: non-default argument follows default argument
```

디버깅

```
def minus(num3, num1=3, num2=2):  
    print(num1 - num2 - num3)  
  
minus(5)
```

Result

-4

13 함수 (Function)

◆ 키워드 아규먼트 (Keyword Argument)

argument가 들어가는 부분에 (key=value) 형태로 입력

argument의 키 값과 함수 parameter명이 같은 변수에 value 값이 들어감

키워드 아규먼트

```
def minus(num1, num2):  
    print(num1 - num2)  
  
minus(2, 3)  
minus(num2=3, num1=2)
```

Result

-1
-1

순서는 상관없이 매치되는 parameter명에 맞게 value값이 들어감

13 함수 (Function)

◆ 리턴 (Return)

리턴 (return)은 함수를 종료하며 결과 데이터를 반환하는 용도로 사용

함수 return

```
#리턴이 없는 함수
def no_return():
    a = 1 + 2

result = no_return()
print(result)
```

Result

None

함수 return

```
#리턴이 있는 함수
def plus(a, b):
    return a + b

result = plus(1, 2)
print(result)
```

Result

3

13 함수 (Function)



*args

함수를 호출할 때 보내는 **argument**의 개수를 특정할 수 없을 때, 함수의 파라미터를 넣는 영역에 사용

*args

```
def print_args(*args):  
    print(args)  
    print(args[4])  
    print(args[5][1])  
  
print_args(1, 2, 3, 'deep',  
          'learning', ['artificial',  
                      'intelligence'])
```

Result

```
(1, 2, 3, 'deep', 'learning',  
 ['artificial', 'intelligence'])  
learning  
intelligence
```

파라미터로 받는 데이터는 **tuple** 데이터 타입으로 **인덱스를 통한 접근**이 가능함

13 함수 (Function)

*args 활용 예시

함수를 호출할 때 보내는 **argument**의 개수를 특정할 수 없을 때, 함수의 파라미터를 넣는 영역에 사용

***args**

```
def avg_func(*args):  
    return sum(args) / len(args)  
  
a = avg_func(100, 70, 80, 99, 85, 60, 80)  
  
print('avg : {}'.format(round(a, 2)))
```

Result

avg : 82.0

13 함수 (Function)

**kwargs

함수에서 parameter로 키워드 argument를 받아올 수 있음 (key, value로 구성된 dictionary 타입)

**kwargs

```
def avg_func(**kwargs):  
    print(kwargs)  
    total = 0  
    count = 0  
    for subject, point in kwargs.items():  
        print(subject, point)  
        total += point  
        count += 1  
    return total / count  
  
a = avg_func(korean=100, english=70, math=80, science=90)  
print("avg: {}".format(round(a, 2)))
```

Result

```
{'korean': 100,  
'english': 70, 'math':  
80, 'science': 90}  
korean 100  
english 70  
math 80  
science 90  
avg : 85.0
```

13 함수 (Function)

*args, **kwargs

함수에서 *args와 **kwargs를 함께 사용 가능

***args, **kwargs**

```
def test_func(*args, **kwargs):  
    print(args)  
    print(kwargs)  
  
test_func(1, 2, 3, "deepandhigh", 'python',  
korean=100, english=70, math=80)
```

Result

```
(1, 2, 3, 'deepandhigh', 'python')  
{'korean': 100, 'english': 70, 'math': 80}
```

13 함수 (Function)

◈ 범위 (Scope)

변수는 코드 전체에서 사용 가능한 **global (전역 변수)**와 특정 블록에서만 사용 가능한 **local (지역 변수)**로 나뉨

전역변수 gv

```
gv = 10
def print_gv():
    print(gv)

print_gv()
```

Result

10

print_gv() 함수 안에서도 전역 변수인 gv에 접근 가능

13 함수 (Function)

❖ 범위 (Scope)

변수는 코드 전체에서 사용 가능한 global (전역 변수)와 특정 블록에서만 사용 가능한 local (지역 변수)로 나뉜다

전역/지역 변수 gv1, 2

```
gv1, gv2 = 1, 2
def print_variable():
    gv1 = 10
    gv2 = 20
    print(gv1, gv2)
    return gv1, gv2

print_variable()
print(gv1, gv2)
```

Result

```
10 20
1 2
```

동일한 변수명의 전역/지역 변수가 있을 경우, 지역변수의 우선순위가 더 높음

13 함수 (Function)

◈ 범위 (Scope)

global 예약어를 사용하면 함수 내에서 전역 변수의 값 변경이 가능

global 변수 접근

```
gv = 12
def change_gv(data):
    global gv
    gv=data

print(gv)
change_gv(100)
print(gv)
```

Result

12
100

global 사용은 자칫 프로그램이 꼬일 수 있으므로 유의해야 함

13 함수 (Function)

◆ 람다 함수 (Lambda function)

파라미터를 간단한 계산으로 리턴하는 함수는 람다 함수 이용

일반적 함수 선언

```
def sum_func(x, y):  
    return x + y  
  
sum_func(5, 6)
```

Result

11



람다 함수 선언

```
sum_func2 = lambda x, y : x + y  
sum_func2(5, 6)
```

Result

11

lambda 활용시 코드가 간결해지고 메모리 절약 가능

14 객체지향 (object-oriented)

프로그래밍 패러다임

[명령형 프로그래밍 (Imperative programming)]

- 프로그래밍의 상태와 상태를 변경시키는 구문의 관점에서 연산을 설명하는 프로그래밍 패러다임
- 대부분의 컴퓨터 하드웨어의 구현 방식
- 절차적 프로그래밍, 객체지향 프로그래밍

[선언형 프로그래밍 (Declarative programming)]

- 프로그램이 어떤 방법으로 해야 하는지를 나타내기보다 무엇과 같은지를 설명하는 프로그래밍 패러다임
- Haskell, LISP, PROLOG 등

14 객체지향 (object-oriented)

명령형 프로그래밍의 종류

[절차적 프로그래밍 (Procedural programming)]

- 루틴, 서브루틴, 메소드, 함수 등을 이용한 프로그래밍 패러다임

[객체지향 프로그래밍 (Object-oriented programming)]

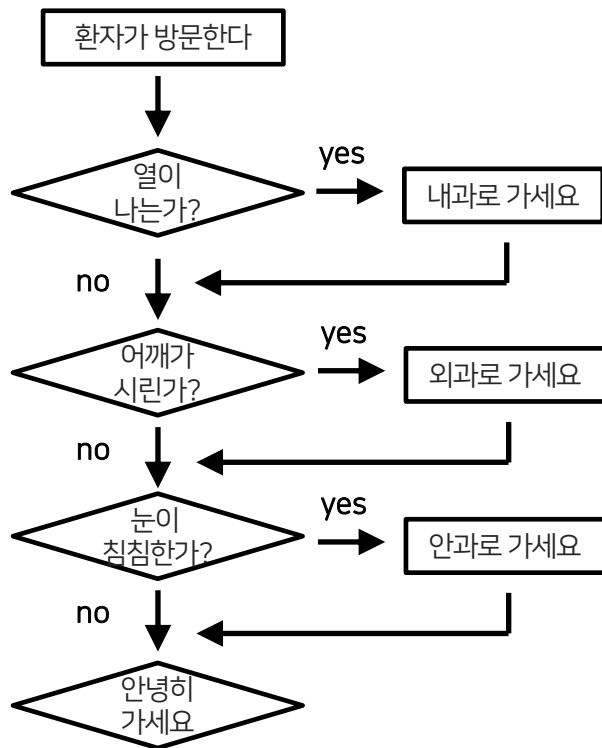
- 프로그램을 수많은 '객체'라는 기본 단위로 나누고 이들의 상호작용으로 서술하는 방식
- 객체(Object): 하나의 역할을 수행하는 '메소드와 변수(데이터)'의 묶음
- 인스턴스(Instance) : 실제 메모리에 할당되어 사용할 수 있는, 실체가 있는 객체

14 객체지향 (object-oriented)

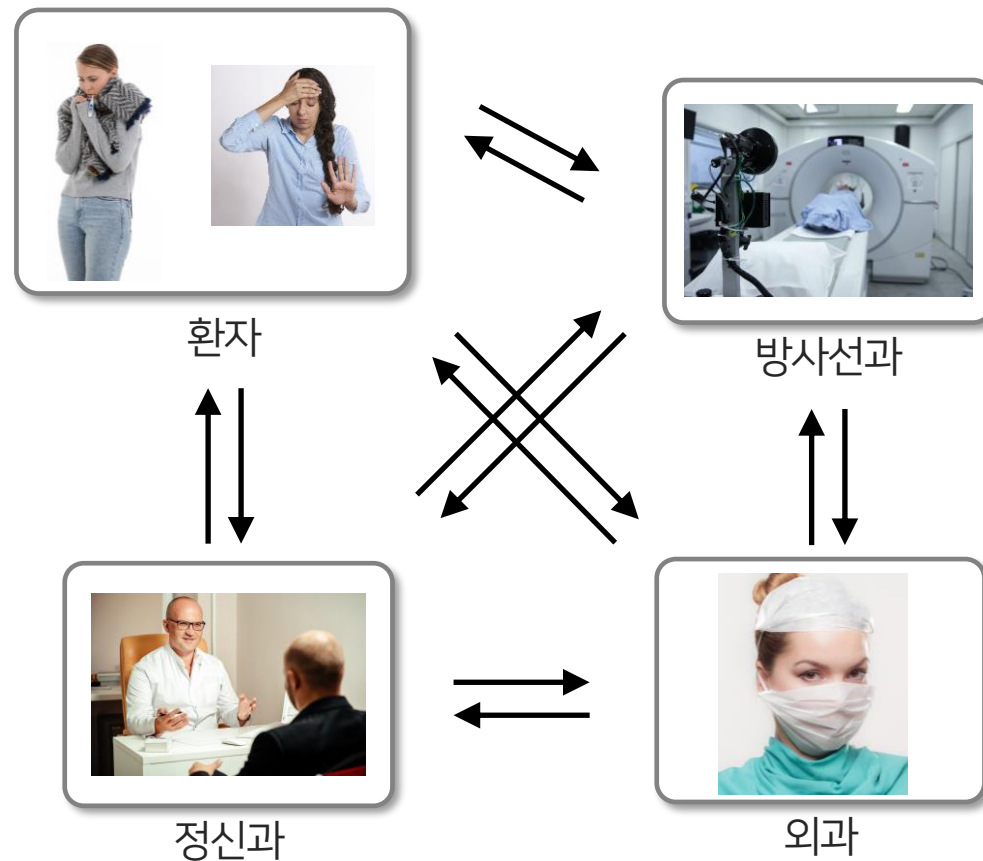
명령형 프로그래밍의 종류



[절차적 프로그래밍]



[객체지향 프로그래밍]



15 클래스 (class)

❖ 클래스 (class)의 정의

변수와 함수를 묶은 사용자 정의 데이터 타입

클래스는 청사진, 설계도, 빵틀

[클래스]



객체는 클래스로 찍어낸 형체

[객체]



15 클래스 (class)

클래스 (class)의 정의

class 명령어를 사용하여 정의

class 정의

```
class Flight:  
    pass  
  
f = Flight()  
print(type(f))
```

Result

```
<class '__main__.Flight'>
```

어떤 변수, 함수도 없는 class도 정의 가능함

15 클래스 (class)

◆ 클래스 (class)의 정의

메소드(method) : 클래스 내 함수. **온점(.)을 통해서 접근** 가능

self : 메소드의 첫번째 파라미터명. 객체 자신을 의미 (**항상 필요**)

class 내부 함수 정의

```
class Flight:
    def number(self):
        return 'KE081'

f = Flight()
print(f.number())
```

Result

KE081

15 클래스 (class)

◈ 생성자 (Constructor)

클래스가 객체가 될 때 실행되는 함수

객체에서 사용할 변수의 초기값 설정 시 사용

생성자 활용

```
class Flight:
    def __init__(self, number):
        self._number = number
    def number(self):
        return self._number

f = Flight('KE082')
print(f.number())
print(f._number)
```

Result

```
KE082
KE082
```

'KE082' 가 Flight 클래스 생성자의 number 변수로 넘어감

15 클래스 (class)

생성자 (Constructor)

생성자 활용 예외처리

```
class Flight:
    def __init__(self, number):
        if not number[:2].isalpha():
            raise ValueError("첫 두글자가 알파벳이 아닙니다.")
        if not number[:2].isupper():
            raise ValueError("첫 두글자가 대문자가 아닙니다.")
        if not number[2:].isdigit():
            raise ValueError("세번째 글자 이후의 글자가 양의  
숫자가 아닙니다.")
        self._number = number

f = Flight('Ke082')
```

Result

```
<ipython-input-38-d6aa9de590f5> in __init__(self, number)
      4     raise ValueError("첫 두글자가 알파벳이 아닙니다.")
      5     if not number[:2].isupper():
----> 6     raise ValueError("첫 두글자가 대문자가 아닙니다.")
      7     if not number[2:].isdigit():
      8     raise ValueError("세번째 글자 이후의 글자가 양의 숫자가 아닙니다.")

ValueError: 첫 두글자가 대문자가 아닙니다.
```

15 클래스 (class)

◆ 더블 언더바 (__)

클래스 내 변수의 외부접근을 막아줌

더블 언더바

```
class Flight:
    def __init__(self, number):
        self._number = number
        self.__number = number

    def number(self):
        return self.__number

f = Flight('KE082')
print(f.number())
print(f._number)
print(f.__number)
```

Result

```
KE082
KE082
```

```
-----
AttributeError                                Traceback (most recent call last)
<ipython-input-6-2e997313a210> in <module>()
      9 print(f.number())
     10 print(f._number)
--> 11 print(f.__number)

AttributeError: 'Flight' object has no attribute '__number'
```


15 클래스 (class)



인스턴스 속성 변경

속성 변경

```
class Flight:
    def __init__(self, number, passenger_num):
        self.__number = number
        self._passenger_num = passenger_num

    def number(self):
        return self.__number

    def add_passenger(self, num):
        self._passenger_num += num #속성 변경

f1 = Flight('KE082', 0)
f2 = Flight('KE081', 0)
f1.add_passenger(2)
f2.add_passenger(3)
print(f1._passenger_num)
print(f2._passenger_num)
```

Result

2
3

15 클래스 (class)

상속 (Inheritance)

새로운 클래스 생성 시, 기존에 정의되어 있던 클래스의 속성과 메소드를 가져오는 기능
기존 클래스 : 부모 클래스, 새로운 클래스 : 자식 클래스

Result

2

클래스 상속

```
class Flight:
    def __init__(self, number, passenger_num):
        self._number = number
        self._passenger_num = passenger_num

    def number(self):
        return self._number

    def add_passenger(self, num):
        self._passenger_num += num
```

클래스 상속

```
class AdvancedFlight(Flight):
    def subtract_passenger(self, num):
        self._passenger_num -= num

f2 = AdvancedFlight('KE081', 0)
f2.add_passenger(3)
f2.subtract_passenger(1)
print(f2._passenger_num)
```

클래스 정의 시 괄호 안에 부모 클래스 이름을 넣어 정의

15 클래스 (class)

Super

상속 시 부모 클래스의 생성자를 재정의하기 위해 사용

클래스 상속

```
class Flight:
    def __init__(self, number, passenger_num):
        self._number = number
        self._passenger_num = passenger_num

    def number(self):
        return self.__number

    def add_passenger(self, num):
        self._passenger_num += num
```

클래스 상속

```
class AdvancedFlight(Flight):
    def __init__(self, number, passenger_num):
        super().__init__('KE083', 10)
    def subtract_passenger(self, num):
        self._passenger_num -= num

f2 = AdvancedFlight('KE081', 0)
f2.add_passenger(3)
f2.subtract_passenger(1)
print(f2._passenger_num)
```

Result

12

super 사용 시 부모 클래스 생성자 형식을 맞춰야 함