03 제어문과 함수

목차

- 조건에 따라 실행 순서를 조정하거나 반복하는 제어문 사용방법
- 사용자 정의 함수 및 내장 함수 사용방법, 클래스와 객체 개념

if 문

● 조건에 따라 코드를 실행하거나, 실행하지 않도록 만들고 싶을 때 사용하는 구문

- 조건문과 블럭으로 구성
- 조건문이란? 참과 거짓을 판단하는 문장
- 조건문이 참인 경우 if조건문 블럭 수행, 조건문이 거짓인 경우 else문 블록 실행
- 같은 레벨의 들여쓰기로 블럭을 구분
  - 같은 수준으로 들여 쓴 연속된 문장들을 블럭이라 함
  - 블럭 중간에 공백 라인도 가능

```
if 조건문 :
        수행할 코드 1
        수행할 코드 2
else :
        수행할 코드 3
        수행할 코드 4
```

조건문이 참일 때 수행되는 블록

조건문이 거짓일 때 수행되는 블록

if 문

● 비교 연산자

비교 연산자	문 법	설 명
==	x == y	x와 y가 같다
!=	x != y	x와 y가 같지 않다
>	x > y	x가 y보다 크다
<	x < y	x가 y보다 작다
>=	x >= y	x가 y보다 크거나 같다
<=	x <= y	x가 y보다 작거나 같다

➢ x가 y보다 큰지 비교(예시)

```
x = 10          # 변수 x에 10을 입력
y = 5           # 변수 y에 5를 입력
x > y           # x가 y보다 큰 값인지 비교
```

----------출력결과----------

```
True
```

if 문

● 논리 연산자

논리 연산자	문 법	설 명
and	x and y	x, y 모두 참일 때 참
or	x or y	x, y 중 하나라도 참이면 참
not	not x	x가 거짓일 때 참

➢ or 연산 실행(예시)

```
x = “True”                  # 변수 x에 True 입력
y = “False”                 # 변수 y에 False 입력
x or y                      # x와 y 값 중 하나라도 참이면 참을 출력
----------------------------------------------------출력결과----------------------------------------------------
True
```

➢ 조건문에 in, not in 사용하기

```
"x" not in ("x", "y", "z")          # 튜플 ("x", "y", "z")에 "x"가 없으면 참. "x" 가 있으므로 거짓
----------------------------------------------------출력결과----------------------------------------------------
False
```

if 문

● 짝수/홀수 판단 예제

➢ 짝수이면 짝수 출력, 홀수이면 홀수를 출력하는 코드

➢ % 연산자 이용

```
x = 3                    # 변수 x에 3을 대입
if x % 2 == 0 :          # x를 2로 나눈 나머지 값과 0을 비교
    print("짝수")        # 조건문이 참일 때 실행
else:
    print("홀수")        # 조건문이 거짓일 때 실행
```

----------출력결과----------

홀수

if 문

● 리스트 자료형 활용 예제

```
pass_list = ["이순신", "을지문덕", "연개소문", "강감찬"]          #리스트 자료형
name = "강감찬"
if name in pass_list :
    print (name + "님은 합격입니다.")
else :
    print (name + "님은 불합격입니다.")
```

----------출력결과----------

강감찬님은 합격입니다.

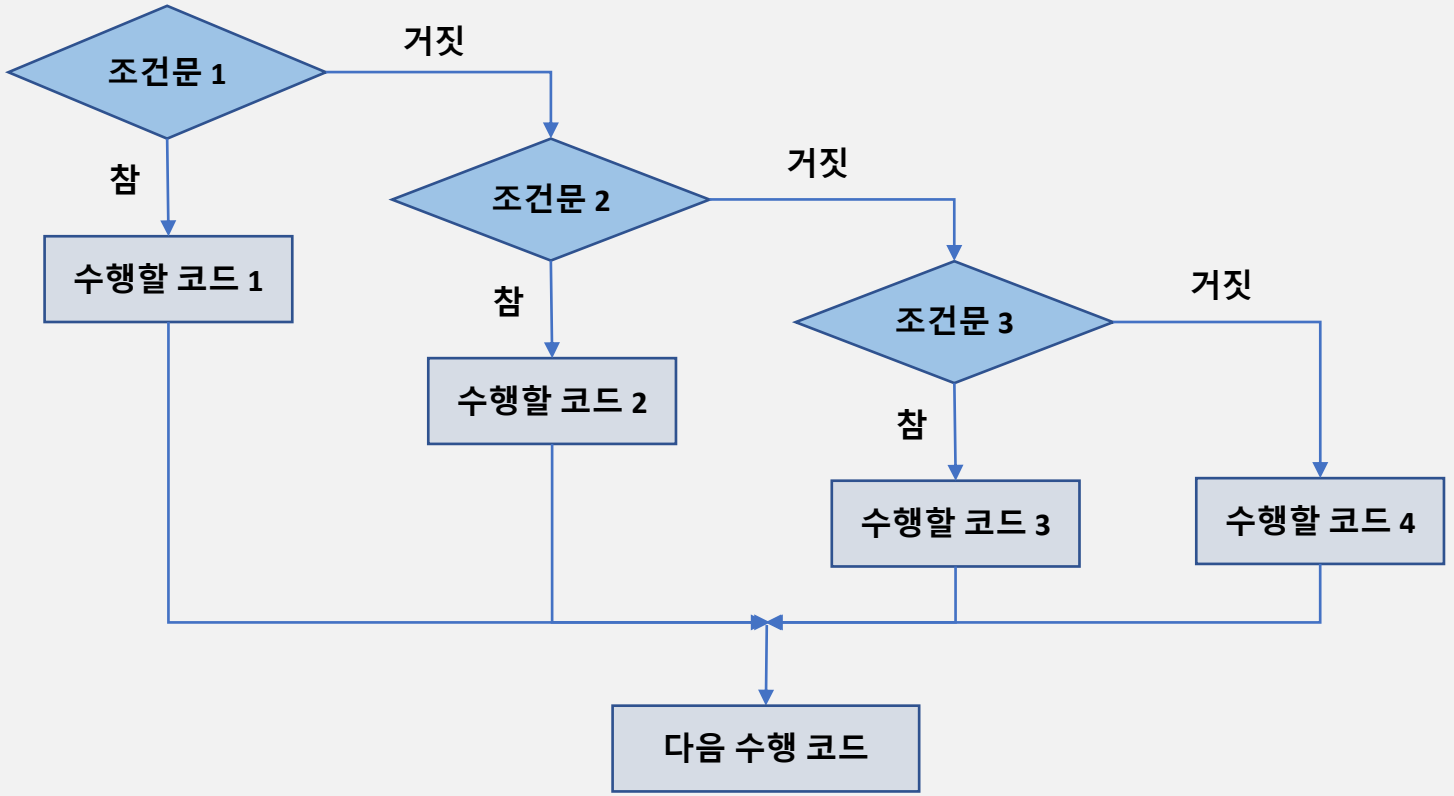
if 문

● elif 문

➢ 다양한 조건을 판단하기 위해서 if ~ else 문에 elif 문을 추가

➢ elif는 개수에 제한 없이 사용 가능

```
if 조건문 1 :
        수행할 코드 1
elif 조건문 2 :
        수행할 코드 2
elif 조건문 3 :
        수행할 코드 3
else :
        수행할 코드 4
다음 수행 코드
```



if 문

● elif 문

➢ 활용 예시

```
x= 55
if x >= 70 :                # x가 70보다 작으므로 조건문의 값은 거짓
    print ("양호")
elif x >= 40 :              # x가 40보다 크므로 조건문의 값은 참
    print ("보통")
else :
    print ("불량")
```

----------출력결과----------

보통

while 문

● 반복해서 코드를 수행해야 할 경우 사용하는 구문

➢ 조건문이 참인 동안 while 아래의 코드를 반복해서 수행

```
while 조건문 :
        코드 1
        코드 2      조건문이 참일 때 수행되는 블록
        ...
```

➢ 1부터 100까지 더하는 코드(예시)

```
i = 1
sum = 0
while i < 101 :                    # i 가 101 보다 작을 동안 반복
    sum = sum + i
    i = i + 1
print("1 + 2 + .....+ 99 + 100 =", sum)
```

----------출력결과----------

```
1 + 2 + ... + 99 + 100 = 5050
```

while 문

● break 문

➢ 수행중인 코드를 멈추고 while문을 빠져나와야 하는 경우 활용

```
egg = 2
while True:                                    # 무한 반복
    egg = egg - 1
    print("남은 계란은 %d개입니다." % egg)
    if egg == 0:                               # egg가 0일 때 print() 실행
        print("계란이 없습니다.")
        break                                  # while 문 탈출
----------------------------------------------출력결과----------------------------------------------
남은 계란은 1개입니다.
남은 계란은 0개입니다.
계란이 없습니다
```

while 문

● continue 문

➢ 코드 실행 중에 어떤 특정 조건이 되면 나머지는 실행하지 않고 **while 문을 처음부터 다시 실행**해야 할 경우 사용

```
i = 0
while i < 10:
    i = i + 1
    if i % 2 == 0: continue        # i 가 짝수일 경우 while 문의 처음 코드로 돌아감
    print(i)
```

----------출력결과----------

```
1
3
5
7
9
```

● 무한 루프

➢ 특정 값이 입력될 때까지 어떤 작업을 반복할 필요가 있을 때 : **실행하지 말것!!! – 못 빠져나올 수 있어요...**

```
while True:                        # 조건문이 True이므로 항상 참
    print("Ctrl+C를 눌러야 빠져나갈 수 있습니다.")
```

for 문

● 지정한 횟수만큼 반복할 때 사용하는 구문

➢ "범위 또는 배열"의 변수에서 값을 하나씩 끄집어 내어 "변수" 에 할당하고 for 의 몸체 블록을 수행

```
for 변수 in 범위 또는 배열의 변수(리스트, 튜플, 문자열):
        코드 1
        코드 2      반복 수행되는 블록
        ...
```

➢ 리스트를 활용하여 반복문 수행(예시)

```
web_list = ["네이버", "구글", "다음"]
for a in web_list :                  # 변수 a 에 "네이버", "구글", "다음"이 차례로 대입됨
    print(a)
```

----------출력결과----------

```
네이버
구글
다음
```

for 문

➢ 튜플을 활용하여 반복문 수행(예시)

```
a = [(3, 6), (4, 5), (3, 8)]
for (x, y) in a:              # 변수(x, y)에 (3, 6), (4, 5), (3, 8)이 차례로 대입됨
    print(x * y)              # x와 y를 곱한 값을 출력
----------------------------------------------------------출력결과----------------------------------------------------------
18
20
24
```

for 문

● range() 활용

➢ range() : 파이썬 내장 함수, (시작숫자, 끝숫자, 숫자간격)을 지정하면 끝 숫자를 제외한 숫자 리스트를 생성

```
a = list(range(1, 10))                  # 1부터 9까지 숫자 리스트 생성
b = list(range(1, 10, 2))               # 1 ~ 9에서 2씩 증가하는 숫자 리스트 생성
c = list(range(10, 1, -1))              # 10 ~ 2에서 -1씩 감소하는 숫자 리스트 생성
print(a, b, c)
```

----------출력결과----------

```
[1, 2, 3, 4, 5, 6, 7, 8, 9] [1, 3, 5, 7, 9] [10, 9, 8, 7, 6, 5, 4, 3, 2]
```

➢ for 문과 range() 함수를 함께 사용(예시)

```
sum = 0
for a in range(1, 100, 2):          # 1 ~ 99에서 2씩 증가, 홀수 리스트를 생성
    sum = a + sum                   # sum 변수에 홀수를 차례로 더함
print(sum)
```

----------출력결과----------

```
2500
```

for 문

● 중첩 for 문

➢ for 문은 중첩해서 사용가능

- end="" 줄을 바꾸지 않도록
- sep="" 띄어쓰기 안되도록 (separator)

```
for x in range(2, 8):                                    # x에 2부터 7까지 대입
    for y in range(2, 7):                                # y에 2부터 6까지 대입
        print("%d X %d = %d" %(x, y, x*y), "₩t", end="")  # x, y, x*y 출력, tab으로 간격 조정,
    print()                                              # 줄 바꾸기
```

출력결과

```
2 X 2 = 4    2 X 3 = 6    2 X 4 = 8    2 X 5 = 10   2 X 6 = 12
3 X 2 = 6    3 X 3 = 9    3 X 4 = 12   3 X 5 = 15   3 X 6 = 18
4 X 2 = 8    4 X 3 = 12   4 X 4 = 16   4 X 5 = 20   4 X 6 = 24
5 X 2 = 10   5 X 3 = 15   5 X 4 = 20   5 X 5 = 25   5 X 6 = 30
6 X 2 = 12   6 X 3 = 18   6 X 4 = 24   6 X 5 = 30   6 X 6 = 36
7 X 2 = 14   7 X 3 = 21   7 X 4 = 28   7 X 5 = 35   7 X 6 = 42
```

for 문

● continue 문

➢ for 문에서도 continue 문 사용가능

```
scores = [(1, 83), (2, 66), (3, 55), (4, 70), (5, 50)]
for num, score in scores:                    # num 과 score 에 튜플 데이터를 대입
    if score < 60:                           # score 가 60 보다 작을 경우 continue 실행
        continue                             # for 문의 처음으로 돌아감
    print("%d번 학생 축하합니다." %num)        # score 가 60 보다 크거나 같을 경우 print 실행
----------------------------------------------------출력결과----------------------------------------------------
1번 학생 축하합니다.
2번 학생 축하합니다.
4번 학생 축하합니다.
```

함수 만들기

● 함수 : 입력값을 가지고 어떤 일을 수행한 다음에 그 결과값을 돌려주는 코드의 집합

- 함수명은 변수처럼 임의로 만들고, **def** 로 정의
- 매개변수(parameter)는 이 함수에 입력으로 전달되는 값을 받는 변수이며 함수 안에서 활용
- 함수의 마지막에는 **return** 문을 사용하며, 함수의 결과값을 함수를 호출한 곳으로 전달

```
def 함수명 (매개변수):
        <코드 1>
        <코드 2>
          ...
        return  결과값
```

함수 만들기

● 함수의 종류

① 매개변수와 반환값이 있는 함수

```
def mul(x, y):              # mul 함수 정의, 매개 변수 x, y
    res = x * y
    return res              # res 를 결과값으로 전달

print(mul(100, 2))          # mul 함수 호출. 100, 2 는 매개변수(paramenter)로 전달되는 인수(argument)
```

----------출력결과----------

```
200
```

② 매개변수가 없지만 반환값이 있는 함수

```
def good( ):                                # good 함수 정의. 매개 변수 없음
    return "안녕하세요! 반갑습니다."
print(good( ))                              # good 함수 호출
```

----------출력결과----------

```
안녕하세요! 반갑습니다.
```

함수 만들기

● 함수의 종류

③ 매개변수는 있지만 반환값이 없는 함수

```
def test(a):                                    # test 함수 정의
    if a >= 60:                                 # a가 60 이상이면  “합격을 축하드립니다.” 출력
        print(“합격을 축하드립니다.”)
    else:                                       # a가 60 미만이면 “불합격입니다.ㅠㅠ” 출력
        print(“불합격입니다.ㅠㅠ”)
test(55)
```

----------출력결과----------

```
불합격입니다.ㅠㅠ
```

④ 매개변수와 반환값 모두 없는 함수

```
def sayhello( ):                                # sayhello() 함수 정의
   print("안녕하세요! 반갑습니다.")
sayhello()                                      # 함수 호출
```

----------출력결과----------

```
안녕하세요! 반갑습니다.
```

여러 개의 입력값을 받는 함수 만들기

◆ 입력 값이 몇 개가 입력될지 모를 때

```
def 함수명 (*매개변수):
```

```
def add_many(*args):            # 매개변수 앞에 *을 붙이면 입력값을 전부 모아서 튜플로 만들어 준다.
    result = 0
    for i in args:
        result = result + i
    return result

print(add_many(1, 2, 3))
```

----------출력결과----------

```
6
```

```
print(add_many(1, 2, 3, 4, 5, 6, 7))
```

----------출력결과----------

```
28
```

함수 만들기

➢ 함수 사용시 주의할 점
- 변수의 유효 범위
- 함수 안의 변수와 함수 밖의 변수는 변수명이 동일하더라도 **서로 다른 변수임**

```
a = 1                              #-- 함수 밖의 변수

def test(a):                       # test() 함수 정의
    a = a + 1                      #-- 함수 안의 변수
    print(a)                       #-- 함수 안의 변수값 출력
    return a

test(a)                            # test() 함수 호출
print(a)                           #-- 함수 밖의 변수값 출력
----------------------------------------------------출력결과----------------------------------------------------
2
1
```

입력과 출력 함수

● 입력 함수 : input()

➢ 입력되는 값을 저장하기 위해 test라는 변수명에 input() 함수를 사용하여 사용자가 값을 입력하게 함

```
In [1]: test = input()
```

➢ [Shift + Enter] 키를 눌러 실행→아래 output 라인에 입력하는 공간이 생성됨 → Hello Python이라고 입력 [Enter]

```
In [*]: test = input()
        Hello Python
```

➢ 입력한 값이 test 변수에 저장되고, 화면에 출력됨

```
In [2]: test = input()
        Hello Python
```

```
1 print(test)
```

```
Hello Python
```

```
1 type(test)
```

```
str
```

입력과 출력 함수

● 입력 함수 : input()

➢ **화면에 정보를 출력해서 입력 창을 띄울 때**

```
In [6]: 1 input("저장하고 싶은 값을 입력하세요:")
```

➢ **[Shift + Enter] 키를 눌러 실행→아래 output 라인에 문자열과 함께 입력하는 공간이 생성됨**
→ Hello Python이라고 입력 [Enter]

```
저장하고 싶은 값을 입력하세요: Hello Python
```

```
Out[7]: 'Hello Python'
```

입력과 출력 함수

- **출력 함수 : print()**
  - ➢ 출력하고 싶은 변수, 문자, 숫자, 연산식 등을 print() 함수에 입력하면 그 값을 출력
  - ➢ 하나의 print()로 여러 문자열을 출력할 경우, 문자열 사이에 공백( ), 콤마(,), 더하기(+)를 사용

```
print("I" "eat" "5" "eggs.")        # 문자 사이에 공백 사용했지만 출력값 사이에 공백 없이 출력됨
----------출력결과----------
I eat5eggs.

print("I" "eat" 5 "eggs.")          # 공백 사용했지만 콤마가 없어 에러
----------출력결과----------
invalid syntax. Perhaps you forgot a comma?

print("I", "eat", 5, "eggs.")       # 콤마(,) 사용 시 공백 추가됨, 숫자 입력 가능
----------출력결과----------
I eat 5 eggs.

print("I"+"eat"+"5"+"eggs.")        # 더하기(+)는 문자열끼리만(같은 자료형 끼리) 사용 가능
----------출력결과----------
I eat5eggs.
```

입력과 출력 함수

● 출력 함수 : print()

➢ 여러 개의 print() 함수를 한 줄로 출력하고 싶다면 자료 입력 후 end=" "를 입력

```
print("I", end=" ")                    # end=" "사이에 공백 입력
print("eat", end=" ")
print(5, end=" ")
print("eggs.", end=" ")
----------------------------------------------------------출력결과----------------------------------------------------------
I eat 5 eggs.
```

➢ end="" 공백이 없다면?

Ieat5eggs.

입력과 출력 함수

● 입력 값을 숫자로 변환하기

➢ input() 함수를 통해 입력된 값은 숫자를 입력해도 전부 **문자열**로 처리됨.

➢ 숫자로 변환하기 위해서는 **형변환**을 해야 함

```
num = input()
print(num)
type(num)
```

----------출력결과----------

```
39
39
str
```

```
num = int(input("숫자를 입력하세요 : "))
print(num)
type(num)
```

----------출력결과----------

```
숫자를 입력하세요 : 39
39
int
```

외부 파일 읽고 쓰기

● 파일 열기 모드

➢ 파일을 열 때 내장 함수 open() 사용

```
open(file, mode, buffering, encoding, errors, newline, closefd, opener)
```

- **file** : **파일 경로**
  - **"파일명"**만 입력하면 Jupyter Notebook이 **실행되는 폴더**에서 파일을 읽거나 저장.
  - 다른 폴더를 이용하려면 **경로를 포함**해서 작성해야 함
- **mode** : 파일이 열리는 옵션, 입력값에 따라 **읽기용, 생성용, 편집용** 등으로 설정
  - **r** : **읽기** 모드, 파일을 읽을 때 사용하며 기존에 파일이 없으면 에러 발생
  - **w** : **쓰기** 모드, 내용을 쓸 때 사용하며 기존에 파일이 있으면 기존 내용 삭제 후 작성,
    파일이 없는데 **w**로 여는 경우는 파일 새로 생성
  - **a** : **추가** 모드, 기존 내용 다음에 추가로 작성하며 기존 내용은 유지
  - **x** : 파일이 없으면 파일을 생성하여 **쓰기** 모드로 열고, 기존에 파일이 있으면 에러 발생

외부 파일 읽고 쓰기

● 파일 닫기

➢ (주의) 파일을 열고 작업이 완료되면 반드시 파일을 닫아야 함

➢ 파일을 닫지 않으면 버퍼링되어 있는 데이터가 기록되지 않고 지워질 수 있음

```
f = open("yonsei summer.txt", "r")      # 현재 경로에서 "yonsei summer.txt"를 읽기 모드로 열기
f.close()                               # 파일 닫기
----------------------------------------------------출력결과----------------------------------------------------
FileNotFoundError : [Errno 2] No such file or directory: 'yonsei summer.txt'
```

외부 파일 읽고 쓰기

● 텍스트 파일 만들고 쓰기

➢ write() : 텍스트를 추가하는 함수

```
변수명.write("내용")
```

➢ 여러 줄을 입력하려면 큰따옴표(혹은 작은따옴표) 3개("""내용""")를 사용

➢ "yonsei summer.txt" 파일에 텍스트 3줄을 입력(예시)

```
f = open("yonsei summer.txt", "w")              # 현재 경로에서 "yonsei summer.txt"를 쓰기 모드로 열기
f.write("""연세대학교 여름
업무자동화와 웹 크롤링 교육에 오신 여러분 환영합니다.
반갑습니다!!!""")
f.close()                                       # 파일 닫기
```

외부 파일 읽고 쓰기

● 파일 읽기

➢ 파일의 내용은 한 줄 또는 전체를 읽을 수 있음

```
변수명.readline()          # 파일 내용을 한 줄 읽기
변수명.read()              # 파일 내용 전부 읽기
변수명.readlines()         # 파일 내용의 각 라인을 데이터로 갖는 리스트로 변환
```

➢ 변수 fr에 “yonsei summer.txt” 파일을 읽기 모드로 연 후, readline() 함수로 한 줄만 읽어오기(예시)

```
fr = open("yonsei summer.txt", "r")        # 파일 읽기 모드로 열기
line = fr.readline()                       # 파일 내용 한 줄 출력
print(line)                                # 한 줄 출력
fr.close()
----------------------------------------------------출력결과----------------------------------------------------
연세대학교 여름
```

외부 파일 읽고 쓰기

● 파일 읽기

➢ read()를 이용하면 파일의 전체 내용 읽기 가능

```
fr1 = open("yonsei summer.txt", "r")          # 파일 읽기 모드로 열기
print(fr1.read())                             # 파일 내용 전체 출력
fr1.close()
```

----------출력결과----------

```
연세대학교 여름
업무자동화와 웹 크롤링 교육에 오신 여러분 환영합니다.
반갑습니다!!!
```

➢ 파일을 열 때 a 속성을 사용하면 기존 파일은 유지하고 마지막에 새로운 텍스트 추가 가능

```
fr4 = open("yonsei summer.txt", "a")          # 파일 추가 모드로 열기
fr4.write("""
짧은 기간이지만
알차게 배워 가시기 바랍니다.""")
fr4.close()
```

외부 파일 읽고 쓰기

● 파일 읽기

➢ **"yonsei summer.txt" 파일을 읽기 모드로 열어 보면 앞에서 입력했던 내용이 추가된 것 확인 가능**

```
fr4 = open("yonsei summer.txt", "r")          # 파일 읽기 모드로 열기
print(fr4.read())                             # 파일 내용 전체 출력
fr4.close()
```

----------출력결과----------

```
연세대학교 여름
업무자동화와 웹 크롤링 교육에 오신 여러분 환영합니다.
반갑습니다!!!
짧은 기간이지만
알차게 배워 가시기 바랍니다.
```

내장 함수

- **Python 인터프리터 설치 시 자동으로 설치되는 함수 : 약 70여 가지**

➢ **내장 함수 확인하기**

- **dir** 은 객체가 자체적으로 가지고 있는 변수나 함수를 보여준다.

```
dir(__builtins__)                    # 내장 함수 확인 _ 언더바 2개
```

---출력결과---

…, 'abs', 'aiter', 'all', 'anext', 'any', 'ascii', 'bin', 'bool', 'breakpoint', 'bytearray', 'bytes', 'callable', 'chr', 'classmethod', 'compile', 'complex', 'copyright', 'credits', 'delattr', 'dict', 'dir', 'display', 'divmod', 'enumerate', 'eval', 'exec', 'execfile', 'filter', 'float', 'format', 'frozenset', 'get_ipython', 'getattr', 'globals', 'hasattr', 'hash', 'help', 'hex', 'id', 'input', 'int', 'isinstance', 'issubclass', 'iter', 'len', 'license', 'list', 'locals', 'map', 'max', 'memoryview', 'min', 'next', 'object', 'oct', 'open', 'ord', 'pow', 'print', 'property', 'range', 'repr', 'reversed', 'round', 'runfile', 'set', 'setattr', 'slice', 'sorted', 'staticmethod', 'str', 'sum', 'super', 'tuple', 'type', 'vars', 'zip']

```
dir([1, 2, 3])               # 리스트 객체 관련 함수
```

---출력결과---

… '__subclasshook__', 'append', 'clear', 'copy', 'count', 'extend', 'index', 'insert', 'pop', 'remove', 'reverse', 'sort']

```
dir({'1':'a'})               # 딕셔너리 객체 관련 함수
```

---출력결과---

… '__subclasshook__', 'clear', 'copy', 'fromkeys', 'get', 'items', 'keys', 'pop', 'popitem', 'setdefault', 'update', 'values']

내장 함수

● 파이썬 인터프리터 설치 시 자동으로 설치되는 함수

➢ abs() : 입력 받은 숫자를 절대값을 반환하는 함수

```
abs(-5)
----------출력결과----------
5
```

➢ chr() : 유니코드(Unicode) 값을 입력 받아 그 코드에 해당하는 문자를 출력하는 함수

```
for i in range(97, 123): print(chr(i), end = " ")   # for 문을 활용해 유니코드 97~ 123 문자 출력
----------출력결과----------
a b c d e f g h i j k l m n o p q r s t u v w x y z
```

➢ ord() : 문자의 유니코드 값을 돌려주는 함수 : ordinal value

```
for i in ["a", "b", "c", "d"]: print(ord(i), end = " ")  # "a", "b", "c", "d" 의 유니코드 출력
----------출력결과----------
97 98 99 100
```

내장 함수

● 파이썬 인터프리터 설치 시 자동으로 설치되는 함수

➢ enumerate() : 순서가 있는 자료형(리스트, 튜플, 문자열)을 입력받아 인덱스값을 포함하는 객체를 돌려주는 함수로 for 문과 함께 자주 사용

```
# 리스트 문자열을 입력 받아 index 는 변수 i 에, 문자열은 변수 item 에 대입
for i, item in enumerate(["사과", "참외", "수박"]):
    print(i, item)
```

출력결과

```
0 사과
1 참외
2 수박
```

➢ int() : 문자열 형식의 숫자나 소수점이 있는 숫자를 정수로 돌려주는 함수

```
print(int ("8"))          # 문자열 8을 숫자 8로 반환
print(int (2.3))          # 숫자 2.3을 2로 반환
```

출력결과

```
8
2
```

내장 함수

● 파이썬 인터프리터 설치 시 자동으로 설치되는 함수

➢ len() : 입력 변수의 길이(요소의 수)를 반환하는 함수

```
len("대한민국")                    # 문자열의 길이를 반환
----------------------------------------------------출력결과----------------------------------------------------
4

len([1, 2, 3, 4, 5, 6, 7])         # 리스트의 요소 수를 반환
----------------------------------------------------출력결과----------------------------------------------------
7
```

➢ max(), min() : 최대값, 최소값을 반환하는 함수

```
test = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
max(test), min(test)               # 변수 test의 최대값과 최소값을 반환
----------------------------------------------------출력결과----------------------------------------------------
(10, 1)
```

내장 함수

● 파이썬 인터프리터 설치 시 자동으로 설치되는 함수

➢ pow() : pow(x, y)는 x를 y제곱한 결과를 반환하는 함수

```
pow(3, 2)        # 3의 2제곱
----------------------------------------------------출력결과----------------------------------------------------
9
```

➢ round() : round(x, y)는 실수 x를 y자리까지 반올림하여 결과를 반환하는 함수

```
round(32.3132, 2)        # 소수점 2자리까지 반올림하여 반환
----------------------------------------------------출력결과----------------------------------------------------
32.31
```

➢ sum() : 입력 받은 리스트나 튜플의 모든 요소 합을 계산하여 반환하는 함수

```
sum([1, 2, 3, 4, 5, 6, 7, 8, 9, 10])        # 리스트 요소를 모두 더 함
----------------------------------------------------출력결과----------------------------------------------------
55
```

내장 함수

● 파이썬 인터프리터 설치 시 자동으로 설치되는 함수

➢ sorted() : 입력값을 오름차순으로 정렬한 후, 결과를 리스트로 반환

```
sorted((3, 1, 4, 5, 2)), sorted((3, 1, 4, 5, 2), reverse=True  )       # 튜플을 오름차순과 내림차순으로 정렬
----------출력결과----------
([1, 2, 3, 4, 5], [5, 4, 3, 2, 1])


sorted([3, 1, 4, 5, 2])                                                # 리스트를 오름차순으로 정렬
----------출력결과----------
[1, 2, 3, 4, 5]


sorted("대한민국"),  sorted(["b", "c", "a"])        #문자열은 분리하여 정렬, 리스트 문자열은 요소로 정렬
----------출력결과----------
(['국', '대', '민', '한'], ['a', 'b', 'c'])
```

클래스와 객체

◆ 클래스 구조

- 클래스(붕어빵 틀)와 객체(붕어빵)의 관계
- 동일한 기능을 수행하는 코드를 필요할 때마다 작성한다면 수많은 중복과 낭비 발생
- 붕어빵 틀처럼 동일한 목적을 가진 함수와 변수를 한데 묶어 놓고 필요할 때마다 가져와서 사용하면 코드를 효율적으로 작성할 수 있다.
- 클래스에 포함된 변수 : **멤버 변수**(member variable)
- 클래스에 포함된 함수 : **메소드**(method)

클래스와 객체

● 클래스 구조

➢ 클래스는 동일한 목적을 가진 함수들과 변수들을 한데 묶어 놓은 곳

➢ 클래스 구조 : 클래스명 아래 멤버변수 들을 선언하고, 메소드는 함수처럼 선언

```
class 클래스명 :          # 클래스 선언
    <코드 1>              # 멤버변수 선언
    <코드 2>

    def 함수명 ( ) :      # 메소드 선언
        <코드 3>
        <코드 4>
```

➢ 인스턴스(instance) : 클래스를 기반으로 생성하는 객체

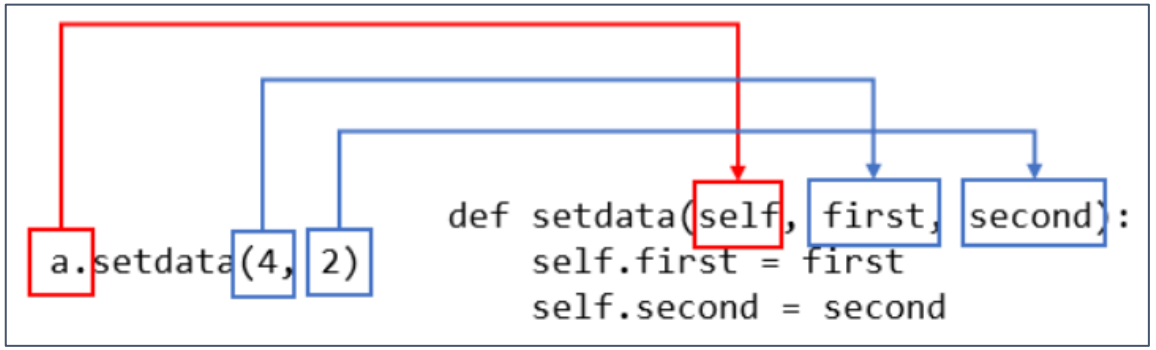
```
인스턴스명 = 클래스명 ( )        # 인스턴스 생성
인스턴스명.멤버변수              # 인스턴스에서 클래스에 정의된 멤버변수를 사용
인스턴스명.메소드( )             # 인스턴스에서 클래스에 정의된 메소드(함수)를 사용
```

클래스와 객체

- self : 메서드를 호출한 객체 자신이 전달되는 통로
  - 파이썬 클래스에서 첫 번째 매개변수 이름은 관례적으로 self 를 사용

```
class FourCal:
    def setdata(self, first, second):       # 메서드의 매개변수
        self.first = first                  # 메서드의 수행문
        self.second = second                # 메서드의 수행문

a = FourCal()                   # 인스턴스 생성
FourCal.setdata(a, 4, 2)        # 클래스이름.메서드 형태로 호출. 객체 a 를 첫번째 매개변수 self 에 반드시 전달
print(a.first)                  # 4
a.setdata(4, 2)                 # 객체.메서드 형태로 호출. self는 반드시 생략.
print(a.second)                 # 2
```



클래스와 객체

➢ __init__ ()

- 객체에 초기값을 설정해야 할 필요가 있을 때는 setdata 메서드를 호출해서 초기값을 설정하기 보다는 생성자를 구현하는 것이 안전하다.
- 다른 메서드와 마찬가지로 첫 번째 매개변수 self 에 생성되는 객체가 자동으로 전달된다.

```
class FourCal:
    def __init__(self, first, second):
        self.first = first
        self.second = second
    def setdata(self, first, second):        # 메서드의 매개변수
        self.first = first                   # 메서드의 수행문
        self.second = second                 # 메서드의 수행문

#a = FourCal()                               # 에러 발생. 생성자 __init__ 이 호출될 때 매개변수 값이 전달되지 않았다.
a = FourCal(4, 2)                            # 생성자 __init__ 이 호출될 때 매개변수를 전달해서 객체를 생성해야 한다.
#FourCal.setdata(a, 4, 2)
print(a.first)                               # 4
#a.setdata(4, 2)
print(a.second)                              # 2
```

클래스와 객체

● 클래스 활용 예제

- ➢ 사무실에 들어가고 나갈 때 인사하는 클래스 만들기(예시)
- ➢ 멤버변수 : sayhello, goodbye,
- ➢ 메소드 : __init__, hello, bye
  - • __init__ : 객체가 생성될 때 자동으로 실행되는 메소드

```
class message :                                  # 클래스 선언
    sayhello = "안녕하세요. 반갑습니다."          # 클래스 멤버변수
    goodbye = "감사합니다. 안녕히 가십시오."      # 클래스 멤버변수
    def __init__(self, name):                    # 객체가 생성될 때 자동으로 호출, 초기화
        self.name = name
    def hello(self):                             # hello 메소드 선언
        print(self.name, "님," , self.sayhello)  # 전달받은 변수와 intro 메시지 출력
    def bye(self) :                              # bye 메소드 선언
        print(self.name, "님," , self.goodbye)   # 전달받은 변수와 bye 메시지 출력
```

클래스와 객체

● 클래스 활용 예제

➢ kim, lee 객체를 생성하고 hello(), bye() 메소드 사용

➢ 객체를 생성할 때, name 멤버변수에서 사용할 값인 "김온달", "이평강"도 지정

```
kim = message("김온달")          # kim 객체 생성, name 변수에 "김온달" 저장
lee = message("이평강")          # lee 객체 생성, name 변수에 "이평강" 저장

kim.hello()                      # kim 객체에서 message 클래스의 hello, bye 메소드 호출
kim.bye()
----------------------------------------------------출력결과----------------------------------------------------
김온달 님, 안녕하세요. 반갑습니다.
김온달 님, 감사합니다. 안녕히 가십시오.

lee.hello()                      # lee 객체에서 message 클래스의 hello, bye 메소드 호출
lee.bye()
----------------------------------------------------출력결과----------------------------------------------------
이평강 님, 안녕하세요. 반갑습니다.
이평강 님, 감사합니다. 안녕히 가십시오.
```