Chapter

3

# Python 둘러보기

# 1

# 입출력

# print 함수

```
print(*objects,
      sep=' ', end='₩n',
      file=sys.stdout,      flush=False)
```

- ❖ **sep : iterable형태의 인수들을 출력할 때, 그 사이에 들어간 문자열**
  - print('우리', '학교', sep='->') # 우리->학교
- ❖ **end : 문자열의 끝을 지정하는 문자열**
  - print('Hello', end=' ') # 출력후 줄바꿈 없음
- ❖ **file : 출력스트림**
- ❖ **flush : 버퍼 비우기**

# 사용 예

```
a = 123
print(a)

a = "Python"
print(a)

a = [1,2,3]
print(a)

print("Life" "is" "too short")

print("Life"+"is"+"too short")

print("Life","is","too short")
```

```
123
Python
[1, 2, 3]
Lifeistoo short
Lifeistoo short
Life is too short
```

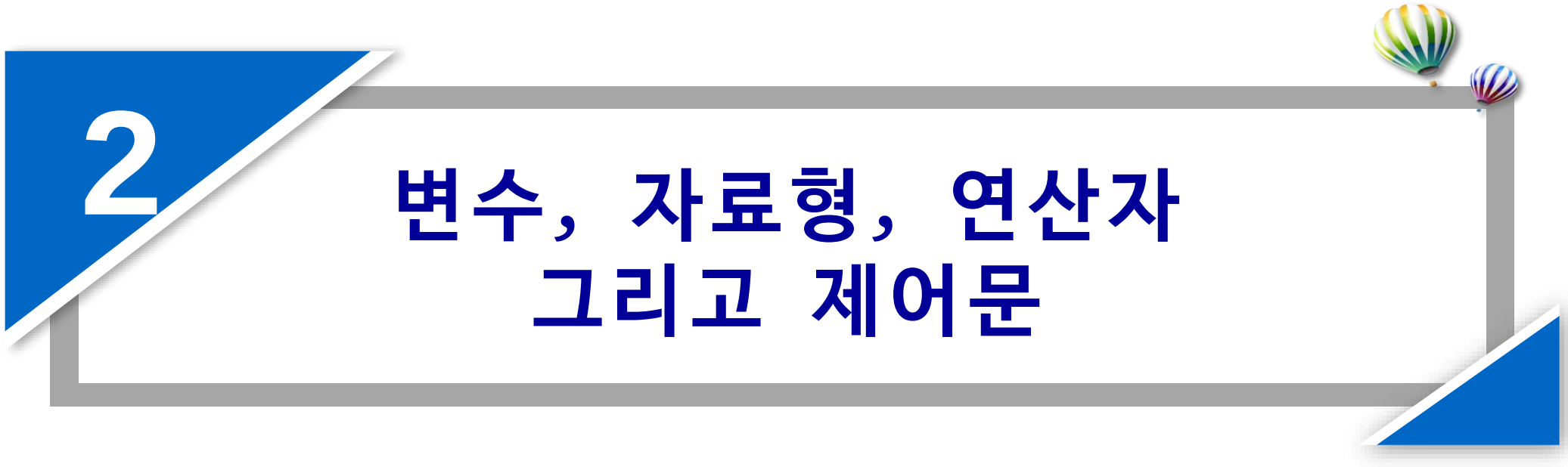
# input 함수

- **x = input("메시지")** # 매개변수로 문자열 출력 가능
  - 모든 입력은 문자열로 입력됨
  - 숫자는 변환해서 사용(eval()사용 가능)
- **(사용 예)**
  - ```
    x = input()
    print(x)

    x = int(input())
    x = pow(x, 2)
    print("x = ", x)

    x = float(input())
    x = x * 2.0
    print("x = ", x)
    ```

```
Kim Bong Keun
Kim Bong Keun
5
x =  25
2.4
x =  4.8
```
```



# 2 변수, 자료형, 연산자 그리고 제어문

# 변수(variable)

## ❖ 변수는 어떤 값(object)을 가리키는(reference) 이름(identifier)

- 식별자(identifier) : 변수, 함수 등에 사용되는 이름
  - 작성규칙 :
    - 영어 대소문자, 숫자, _(밑줄문자)로만 구성된다.
    - 첫글자는 숫자가 올 수 없다.
- 대소문자를 구분(즉, 대문자와 소문자를 다른 글자로 취급)

## ❖ 변수의 생성

- 변수이름 = 값  (예) a = 123

## ❖ 변수의 타입

- 대입연산자(=)의 뒤에 오는 값의 유형에 따라 자동으로 결정됨

# 변수

- **변수는 객체를 가리키는 것(reference name)**
- **여러 개의 변수 만들기**
  - a, b = 'python', 'life'
    print(a, b)
    a = b = 'python'
    print(a, b)
- **변수값 교환**
  - a = 3
    b = 5
    a, b = b, a
    print(a, b)
- **변수 제거**
  - a = 3
    **del**(a)

# Data Type의 종류

- **Number**
  - 숫자형태로 이루어진 자료형
- **String**
  - 문자열
- **List [...]**
  - 리스트 : 추가, 수정, 삭제 가능
- **Tuple (...)**
  - 투플 : 추가, 수정, 삭제 불가
- **Dictionary {k1:v1, ...}**
  - key와 value의 쌍으로 이루어진 자료
- **Sets**
  - 집합 : 요소의 중복 불가

- **변수Type : 명시적인 타입은 없음**

# Number

| 항목 | 사용 예 |
|---|---|
| 논리 | True, False |
| 정수 | 123, -345, 0, 123L |
| 2진수 | 0b110101111010 |
| 8진수 | 0o34, 0o25 |
| 16진수 | 0x2A, 0xFF |
| 실수 | 123.45, -1234.5, 3.4e10 |
| 복소수 | 1 + 2j, -3j<br><복소수 내장함수><br>.real(실수) .imag(허수)<br>.conjugate(켤레복소수) |

# String

## ❖ 문자열 만들기

- "Hello World"
- 'Python is fun'
- """Life is too short, You need python"""
- '''Life is too short, You need python'''

```
'\n'      라인 피드(Line feed)
'\r'      캐리지 리턴(Carriage return)
'\t'      탭(Tab)
'\''      작은따옴표(Literal single quote)
'\"'      큰따옴표(Literal double quote)
'\\'      백슬래시(Literal backslash)
```

## ❖ 사용 예

```
food1 = "Python's favorite food is perl"
say1  = '"Python is very easy." he says.'
food2 = 'Python₩'s favorite food is perl'
say2  = "₩"Python is very easy.₩" he says."
multiline1 = "Life is too short₩nYou need python"
multiline2 = """Life is too short
You need python"""
print(food1)
print(say1)
print(food2)
print(say2)
print(multiline1)
print(multiline2)
```

```
Python's favorite food is perl
"Python is very easy." he says.
Python's favorite food is perl
"Python is very easy." he says.
Life is too short
You need python
Life is too short
You need python
```

# String Operations

- + : 문자열 더하기
- * : 문자열 곱하기
  - print("hello" * 2)  # hellohello
- indexing
  - str = "Hello world"
    print(str[0])  # H
    print(str[-1])  # d
- slicing
  - str = "Hello world"
    print(str[0:5])  # Hello
    print(str[:5])  # Hello
    print(str[6:11])  # world
    print(str[6:])  # world

# String Functions

- upper, lower
- count
- find, index
- join
- lstrip, rstrip, strip
- split
- replace
- (예)

```
 Python is Good                    1
 PYTHON IS GOOD                    2
1                                  3
3                                  4
3                                  5
 ,P,y,t,h,o,n, ,i,s, ,G,o,o,d,     6
Python is Good                     7
['Python', 'is', 'Good']           8
Other                              9
```

```
str = " Python is Good "
print(str)                                  1
print(str.upper())                          2
print(str.count('P'))                       3
print(str.find('t'))                        4
print(str.index('t'))                       5
ch = ','
print(ch.join(str))                         6
print(str.strip())                  7
print(str.split())                  8
print(str.replace(str, "Other"))    9
```

# r-string

## ❖ 백슬래시를 해석하지 않는 문자열

```
rs = 'c:\\newdata\\test'
print(rs)

rs = r'c:\newdata\test' # 원시(백슬래시를 해석하지 않음)
print(rs)
```

```
c:\newdata\test
c:\newdata\test
```

# C-Style String Formatting

```
str = "I eat %d apples." % 3
print(str)

str = "I eat %s apples." % "five"
print(str)

number = 10
day = "three"
str = "a = %d b = %s" % (number, day)
print(str)
```

```
I eat 3 apples.
I eat five apples.
a = 10 b = three
```

## ❖ format

- 플래그
  - 0 - + : leading zero, left align, force sign
- 형식지정자
  - %d, %i : 10진 정수
    - print("%5d" % 123)
  - %o : 8진 정수
  - %x, %X : 16진 정수
  - %f, %e, %E, %g, %G : 실수
    - print("%10.2f" % 123.456)
  - %c : 문자
  - %s : 문자열
  - %% : %문자

# format() 함수

```
str1 = "name: {}, age: {}".format("Kim", 45)
str2 = "name: {1}, age: {0}".format(45, "Kim")
str3 = "name: {name}, age: {age}".format(name="Kim", age=45)
str4 = "name: {0}, age: {1}, name2: {0}".format("Kim", 45)

print(str1)
print(str2)
print(str3)
print(str4)
```

```
name: Kim, age: 45
name: Kim, age: 45
name: Kim, age: 45
name: Kim, age: 45, name2: Kim
```

## ❖ format spec.

```
format_spec ::= [[fill]align][sign][#][0][width][grouping_option][.precision][type]
fill ::= <any character>
align ::= "<" | ">" | "^"
sign ::= "+" | "-"
width ::= digit
grouping_option ::= "_" | ","
precision ::= digit
type ::= "b"|"c"|"d"|"e"|"E"|"f"|"F"|"g"|"G"|"n"|"o"|"s"|"x"|"X"|"%"
```

- 정렬 : 왼쪽(<), 오른쪽(>), 가운데(^)
- 부호 : 반드시표시(+), 음수만표시(-)
- 천단위 구분문자 : grouping_option(보통 ,로 지정)
- %b: 이진수, %n: Number(%d와 유사)

## ❖ format()

```
str = "I eat {0} apples.".format(3)
print(str)
str = "I eat {0} apples."
print(str.format(3))
print(str.format("five"))
str = "x={0} and y={1}"
print(str.format(4, 5))
print("'{0:<10}'".format("hi"))
print("'{0:>10}'".format("hi"))
print("'{0:^10}'".format("hi"))
print("'{0:=^10}'".format("hi"))
print("{0:0.4f}".format(3.141592653))
print("{0:12.4f}".format(3.141592653))
```

```
I eat 3 apples.
I eat 3 apples.
I eat five
apples.
x=4 and y=5
'hi        '
'        hi'
'    hi    '
'====hi====='
3.1416
      3.1416
```

# f-string

## ❖ 보통 변수를 포함하는 문자열을 표현할 때 사용

- Python 3.6이상만 지원

```
name = 'IBM'
shares = 100
price = 91.1

str = f'{name:>10s} {shares:10d} {price:10.2f}'
print(str)
str = f'Cost = ${shares*price:.2f}'
print(str)
```

```
1234567890 1234567890 1234567890
       IBM        100      91.10
Cost = $9110.00
```

# List

- a = []
  b = [1, 2, 3]
  c = ['Life', 'is', 'too', 'short']
  d = [1, 2, 'Life', 'is']
  e = [1, 2, ['Life', 'is']]

- List접근의 예
  - e = [1, 2, ['Life', 'is']]
    print(e[2][0])

```
Life
```

# List(2차원)

(예)

```
mat1 = [[1, 2, 3],
        [4, 5, 6],
        [7, 8, 9]]

mat2 = [ [0]*4 for i in range(3) ] # 리스트 컴프리핸션

mat3 = []
for i in range(3):
    mat3.append([1] * 4)

print(mat1)
print(mat2)
print(mat3)
```

```
[[1, 2, 3], [4, 5, 6], [7, 8, 9]]
[[0, 0, 0, 0], [0, 0, 0, 0], [0, 0, 0, 0]]
[[1, 1, 1, 1], [1, 1, 1, 1], [1, 1, 1, 1]]
```

# List Functions

- indexing
- slicing
- + : 더하기
- * : 곱하기
- 수정 & 삭제
- (예)

```
a = [1, 2, 3]
b = [4, 5, 6]
print(a + b)          # 더하기
print(a * 3)          # 곱하기
a[2] = 7              # 수정
print(a)
b[0:2] = []           # 삭제   del b[0:2]
print(b)
```

```
[1, 2, 3, 4, 5, 6]
[1, 2, 3, 1, 2, 3, 1, 2, 3]
[1, 2, 7]
[6]
```

## ❖ List 복사

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

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

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

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

```
a = [1, 3, 2]
a.append(4)                 # 단일요소 추가
print(a)                    # (1)
a.extend([4, 8])            # 복수요소 추가
a.sort()                    # 정렬
print(a)                    # (2)
a.reverse()          # 뒤집기
print(a)                    # (3)
a.insert(2, 7)              # index 2에 7을 삽입
print(a)                    # (4)
print(a.index(3))    # (5) 3의 index 알아내기
print(a.count(4))    # (6) 4의 개수 알아내기
```

```
[1, 3, 2, 4]                    (1)
[1, 2, 3, 4, 4, 8]              (2)
[8, 4, 4, 3, 2, 1]              (3)
[8, 4, 7, 4, 3, 2, 1]           (4)
4                               (5)
2                               (6)
```

## ❖ 리스트 정렬

```
s = [10, 1, 7, 3]
s.sort()  # s는 자가수정됨(in-place : 제자리에서)
print(s)

s = [10, 1, 7, 3]
s.sort(reverse=True) # reverse()함수와 동일
print(s)

s = ['foo', 'bar', 'spam']
s.sort()
print(s)

s = ['foo', 'bar', 'spam']
t = sorted(s)  # s는 바뀌지 않으며, t는 정렬된 값을 가짐
print(s)
print(t)
```

```
[1, 3, 7, 10]
[10, 7, 3, 1]
['bar', 'foo', 'spam']
['foo', 'bar', 'spam']
['bar', 'foo', 'spam']
```

# Tuple

- List와 비슷함
- ()로 둘러싸고, 수정, 삭제가 불가능
- (예)
  - t1 = ()
    t2 = (1,)
    t3 = (1,2,3)
    t4 = 1,2,3
    t5 = ('a', 'b', ('ab', 'cd'))
- 연산
  - indexing
  - slicing
  - +(더하기) *(곱하기)

## ❖ packing

- Tuple은 관련 항목을 단일 entity로 묶을 때 사용
- (예)

```
t = ('KIM', 50, 'Seoul’)
```

## ❖ unpacking

- 전달받은 tuple은 풀어서 사용가능
- (예)

```
name, age, address = t
print(name, age, address)
```

# Dictionary

- **Associative array, Hash의 개념**
- **key와 value의 쌍으로 이루어진 데이터**
  - { }안에 데이터를 ,로 구분하여 열거
  - dic = {'name':'pey', 'birth': '1118'}
    print(dic)
    print(dic['name'])  # pey
  - print(dic.get('name'))  # pey

- **주의사항**
  - key의 중복 **(X)**
  - List를 key로 사용 **(X)**

# Dictionary Functions

## ❖ keys()

- dict_keys 객체를 리턴

```
dic = {'name':'pey', 'birth': '1118'}
for i in dic.keys() :
    print(i)                          # key를 출력
```

  - 출력의 순서는 임의

- dict_keys 객체를 List로 변환

```
dic = {'name':'pey', 'birth': '1118'}
lst = list(dic.keys())
for i in lst :
    print(i)
```

## ❖ values()

- dict_values 객체를 리턴

## items()

- dict_items 객체를 리턴

```
dic = {'name':'pey', 'birth': '1118'}
for i in dic.items() :
    print(i)                    # 각 item(키,값)을 출력
```

## clear()

- 모두 지우기

## get()

- key로 value 얻기

```
dic = {'name':'pey', 'birth': '1118'}
print(dic.get('name'))
```

## in

- 해당 key의 존재유무 확인

```
dic = {'name':'pey', 'birth': '1118'}
print('name' in dic)
```

# Sets

❖ 집합을 쉽게 처리하기 위한 것

❖
```
s1 = set([1,2,3])
print(s1)
s2 = set("Hello")
print(s2)          # {'e', 'l', 'o', 'H'}
```

❖ 주의사항

- 중복 불가
- 순서 무시(unordered)
  - indexing 불가 (List나 Tuple로 변환 후에 가능)

# True & False

| 값 | 참 or 거짓 |
|---|---|
| “Python” | True |
| “” | False |
| [1,2,3] | True List |
| [] | False List |
| () | False Tuple |
| {} | False Dictionary |
| 1 | True |
| 0 | False |
| None | False |

# Data Type Conversion

| Function | Description |
|---|---|
| **int(x [,base])** | 정수로 변환 |
| **long(x [,base] )** | Long정수로 변환 |
| **float(x)** | 실수로 변환 |
| **complex(real [,imag])** | 복소수 생성 |
| **str(x)** | 객체 x를 문자열로 변환 |
| **tuple(s)** | Converts s to a tuple. |
| **list(s)** | Converts s to a list. |
| **set(s)** | Converts s to a set. |
| **dict(d)** | Creates a dictionary. d must be a sequence of (key,value) tuples |
| chr(x) | Converts an integer to a character. |
| unichr(x) | Converts an integer to a Unicode character. |
| ord(x) | Converts a single character to its integer value. |
| hex(x) | Converts an integer to a hexadecimal string. |
| oct(x) | Converts an integer to an octal string. |

# 연산자(Operators)

- **산술연산자**
  - + - * / **(지수) //(몫) %(나머지)
- **비교연산자**
  - == != < > <= >=
- **대입연산자**
  - = += -= *= /= %= **= //=
- **비트연산자**
  - & | ^ ~ << >>
- **논리연산자**
  - and or not
- **멤버연산자**
  - in
- **식별연산자**
  - is is not

✓ **주의: ++, -- : 없음**

# 연산자 우선순위

| Operator | Description |
|---|---|
| ** | 지수 |
| ~ + - | 비트not, 부호(+), 부호(-) |
| * / % // | 곱하기, 나누기, 나머지, 몫 |
| + - | 덧셈과 뺄셈 |
| >> << | 좌우 비트 시프트 |
| & | 비트 'AND' |
| ^ ¦ | 비트 'XOR'와 'OR' |
| <= < > >= | 비교 연산자 |
| <> == != | 등가 연산자 |
| = %= /= //= -= += *= **= | 할당 연산자 |
| is is not | 식별 연산자 |
| in not in | 멤버 연산자 |
| not or and | 논리 연산자 |

# 제어문 간단히 알아보기

## ❖ if, if ~ else, if ~ elif ~ else

```
if 조건문 :                  # 문장은 반드시 단계별로 들여쓰기(Tab)
   문장                      # pass 가능
else :
   문장
```

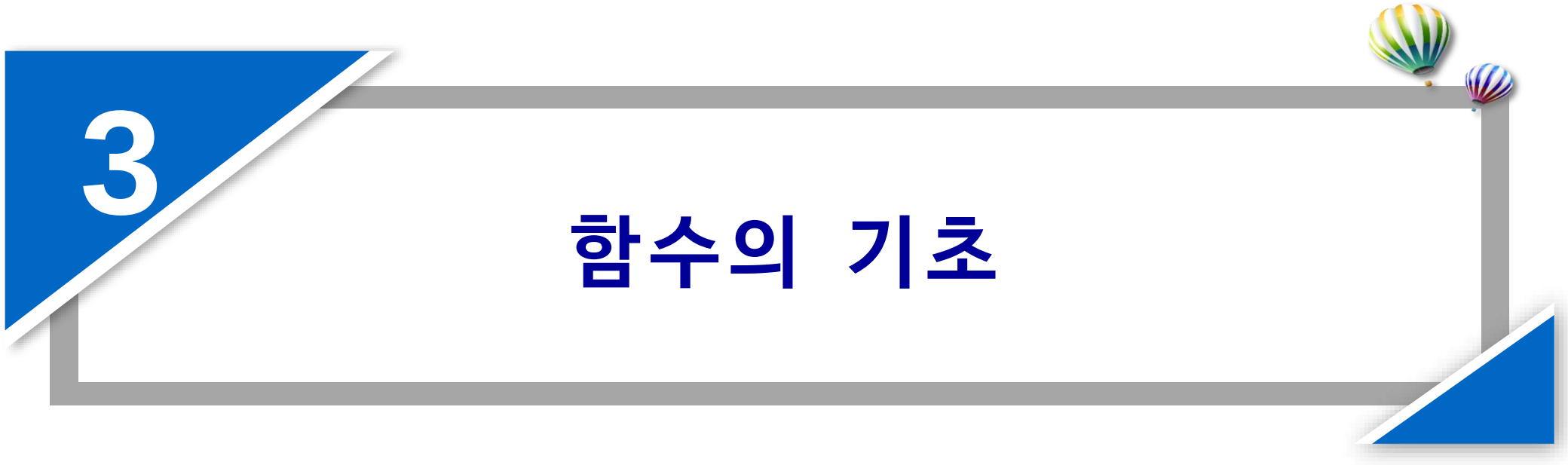
## ❖ while

```
while 조건문 :  # 조건에 True사용 시 무한루프
   문장
```

## ❖ for

```
for 변수 in 리스트(투플,문자열 등) :
   문장
for 변수 in range(시작값, 끝값, 간격) :       # 끝값은 1 더 큰값을 지정
   문장
```

## ❖ break, continue


3
함수의 기초

# Function

## ❖ 함수의 구조

**def** 함수명(입력 인수):

    수행할 문장1

    수행할 문장2

    ...

## ❖ (예)

```
def sum(a, b):
    c = a + b
    return c

m = sum(4, 7)
print('m = ', m)
```

## ❖ Doc Strings

- doc-strings은 함수 이름 바로 뒤에 직접 쓴 문자열
- help(), IDE 등에서 인식

```
def total(n) -> int :
    '''
    정수 1부터 n까지의 합을 반환
    '''
    sum = 0
    while n > 0:
        sum += n
        n -= 1
    return sum
```

## ❖ Type Annotations

- 함수 정의에 타입 힌트(type hint)를 추가한 것
- 연산에 사용되지 않으나, IDE, 코드검사기 등에서 사용

## ❖ 여러 개의 값 반환

```
def divide(a, b):
    q = a // b          # 몫
    r = a % b           # 나머지
    return q, r         # tuple 반환

x, y = divide(37,5) # x = 7, y = 2
t = divide(37, 5)   # t = (7, 2)
print(x, y, t)
```

```
7 2 (7, 2)
```

# global 변수

- 함수의 바깥에서 선언된 변수

```
x = value # 글로벌 변수
def foo():
    y = value # 로컬 변수
```

- 함수내에서 글로벌 변수는 자유롭게 접근이 가능
- 단, 함수내에서 글로벌 변수는 수정이 불가능
  -> 만일, 수정이 필요하면 global로 선언한 후 가능

```
name = 'Dave'

def spam():
    global name
    name = 'Guido' # 글로벌 변수 name을 변경
```

# Function(기본인수)

- 기본인수(default parameters) : 기본값을 가진 인수
- (예)

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

say_myself("박응용", 27)
print()
say_myself("박사라", 27, False)
```

```
나의 이름은 박응용 입니다.
나이는 27살입니다.
남자입니다.

나의 이름은 박사라 입니다.
나이는 27살입니다.
여자입니다.
```

# Function(가변인수)

- 가변인수 : 인수의 개수가 가변적인 인수
- (예)

```
def sum_many(*args):
    sum = 0
    for i in args:
        sum += i
    return sum
```

```
m = sum_many(1, 2, 3)
print('m = ', m)
m = sum_many(1, 2, 3, 4)
print('m = ', m)
```

```
m =  6
m =  10
```

# Function(키워드인수)

## ❖ 인수는 기본적으로 순서대로 전달됨

- 위치 매개변수(positional argument) 전달 방식

```
def person(name, age, address):
    print(f"이름: {name}")
    print(f"나이: {age}")
    print(f"주소: {address}")

person("Kim", 50, "Chungju")
```

- 반드시 순서를 기억하여야 함 -> 매우 불편

## ❖ 키워드 인수

- 인수에 이름을 붙이는 기능(키워드=값)

```
person(name="Kim", address="Chungju", age=50)
```

# 가변인수와 키워드 인수의 혼용

❖ 가변인수, 키워드인수(기본인수)의 순으로 작성

```
def sum_many(*args, times=2):
    sum = 0
    for i in args:
        sum += i * times
    return sum

print(sum_many(1, 2, 3))
print(sum_many(1, 2, 3, times=3))
```

```
12
18
```

# Function(키워드 가변인수)

- 키워드 가변인수 : 임의 개수의 키워드 인수
- (예)

```
def sum_many(x, y, **kwargs):
    print(x + y)
    print(kwargs)

sum_many(3, 5, one="하나", two="둘", three="셋")
```

```
8
{'one': '하나', 'two': '둘', 'three': '셋'}
```

# 가변인수와 키워드 가변인수의 혼용

## ❖ 가변인수, 키워드 가변인수의 순으로 작성

```
def sum_many(*args, **kwargs):
    sum = 0
    for i in args:
        sum += i
    print(f"sum = {sum}")
    print(kwargs)

sum_many(3, 5, one="하나", two="둘", three="셋")
```

```
sum = 8
{'one': '하나', 'two': '둘', 'three': '셋'}
```

# Function(Lambda 함수)

## ❖ 구문

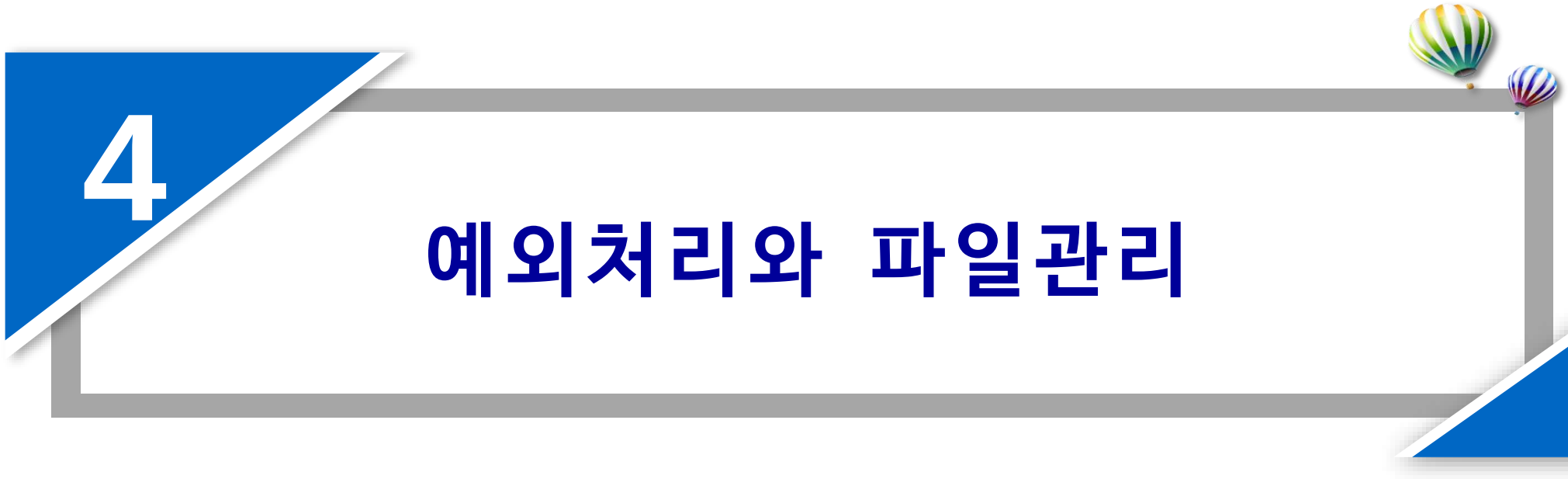
- **lambda 인수 : 식**
- 한 줄 짜리 축약형 함수, 자동 return
- if, while 등은 허용되지 않음

## ❖ (예1)

```
g = lambda x, y : x * y
print(g(2,3))
```

## ❖ (예2)

```
print((lambda x, y : x + y)(2,3))
```


4
예외처리와 파일관리

# 예외처리

## ❖ 예외처리 문법

- except는 여러 개가 나올 수 있고, finally와 else는 옵션임

```
try:
    실행할 문장
    ...
except [발생오류 [as 변수]]:
    예외처리 문장
[finally:]
    오류의 유무와 관계없이 마지막에 항상 실행할 문장]
[else:]
        오류가 없는 경우에만 실행할 문장]
```

## ❖ 예외처리 방법

```
try:
    ...
except:
    ...
```

```
try:
    ...
except 발생오류:
    ...
```

```
try:
    ...
except 발생오류 as 변수:
    ...
```

```
try:
    ...
finally:
    ...
```

```
try:
    ...
except 발생오류:
    ...
finally:
    ...
```

```
try:
    ...
except 발생오류:
    ...
else:
    ...
```

# 예외처리

## ❖ 예외발생

- 예외를 발생시킬 때, raise 문을 사용

```
if name not in authorized:
    raise RuntimeError(f'{name} not authorized')
```

## ❖ 예외처리

- 발생된 예외는 붙잡아 처리할 수 있음
- Exception은 모든 예외

```
try:
    authenticate(username)
except RuntimeError as e:
    print(e)
```

**Exception**
ArithmeticError
AssertionError
EnvironmentError
EOFError
ImportError
IndexError
KeyboardInterrupt
KeyError
MemoryError
NameError
ReferenceError
RuntimeError
SyntaxError
SystemError
TypeError
ValueError 등

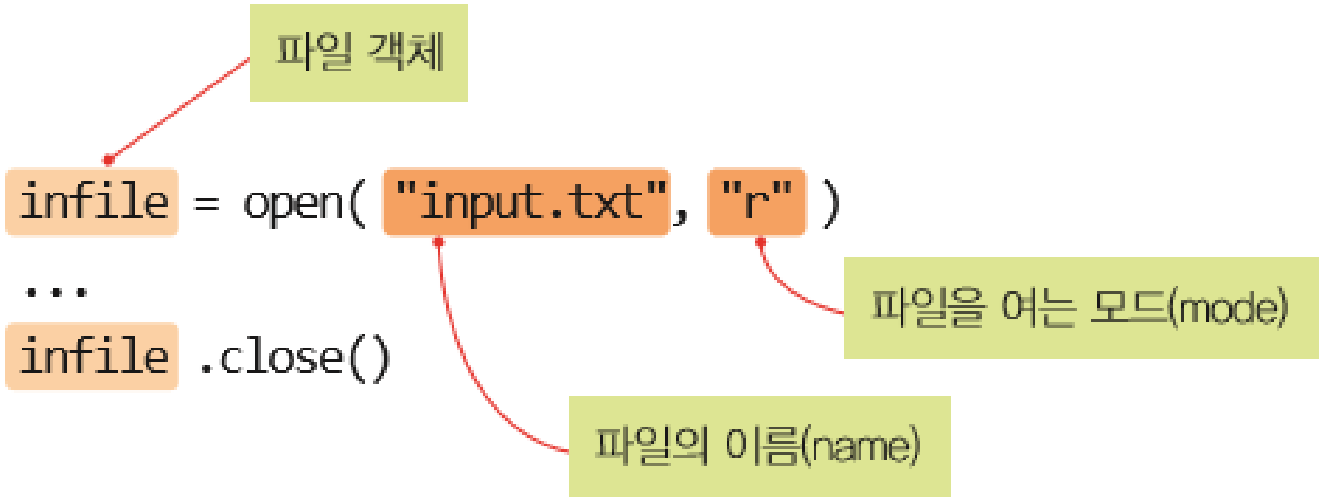
# 5

# 파일 입출력

# 파일 입출력의 기초

## ❖ 파일 열고 닫기

- mode : “r”, “w”, “a”, “r+”, “w+”, “a+” (각각에 t, b를 추가가능)
  - t: text file, b: binary file
  - “r”, “r+” : 파일이 없으면 오류
  - “r+” : 기존파일에 덮어쓰기, “w+” : 기존파일을 지우고 새로 읽고 쓰기
  - “a” : fp가 끝으로 이동(쓰기모드), “a+” : 읽기와 추가모드
- 열려진 파일은 사용 후 반드시 닫아야 함



# 텍스트 파일

## ❖ 파일 쓰기

- write(str)

```
outfile = open("c:/temp/newfile.txt", 'w')
for i in range(1, 11):
    outfile.write(f"{i}번째 라인입니다.\n")
outfile.close()
```

- 경로명
  - '/'문자 또는 r-string으로 표현
    - "c:/temp/newfile.txt"
    - r"c:₩temp₩newfile.txt" 아니면 "c:₩₩temp₩₩newfile.txt"로 해야 함
- 보통 라인의 끝에 '₩n'을 추가

## ❖ 파일 읽기

- read() -> str
  - 통째로 읽어 문자열을 반환

```
f = open("c:/temp/newfile.txt", 'r')
lines = f.read()
print(lines)
f.close()
```

- readline() -> str
  - 한 라인을 읽어 문자열을 반환

```
f = open("c:/temp/newfile.txt", 'r')
while True:
    line = f.readline().rstrip() # '\n'제거
    if not line: break
    print(line)
f.close()
```

- readlines() -> list
  - 라인단위로 읽어 리스트를 반환

```
f = open("c:/temp/newfile.txt", 'r')
lines = f.readlines()
for line in lines:
    line = line.rstrip() # 라인끝의 '\n'제거
    print(line)
f.close()
```

## ❖ with 구문 사용하기

- 파일을 사용한 후 닫는 것을 자동으로 처리해 줌

```
with open("C:/temp/newfile.txt", 'w') as outfile:
    for i in range(1, 11):
        outfile.write(f"{i}번째 라인입니다.\n")
    # outfile.close() 필요없음
```

```
with open("c:/temp/newfile.txt", 'r') as infile:
    lines = infile.readlines()
    for line in lines:
        line = line.rstrip() # 라인끝의 '\n'제거
        print(line)
    # infile.close() 필요없음
```

# 바이너리 파일

## ❖ bytes 클래스로 바이트객체 만들기

```
bytes([10, 20, 30, 40, 50])  # b'\n\x14\x1e(2'
bytes(b'hello')              # b'hello'
```

## ❖ 바이트 자료형과 인코딩

- python의 문자열 기본 인코딩 방법 : UTF-8
- 바이트객체 : ASCII 코드로 저장

```
bstr1 = 'hello'.encode()
bstr2 = '안녕'.encode('euc-kr')
bstr3 = '안녕'.encode('utf-8')
print(bstr1, bstr1.decode())
print(bstr2, bstr2.decode('euc-kr'))
print(bstr3, bstr3.decode('utf-8'))
```

```
b'hello' hello
b'\xbe\xc8\xb3\xe7' 안녕
b'\xec\x95\x88\xeb\x85\x95' 안녕
```

```
data = [1, 2, 3, 4, 5]
with open("c:/temp/data.bin", "wb") as f:
    f.write(bytes(data))
```

```
with open("c:/temp/data.bin", "rb") as f:
    data = f.read()
    for b in data:
        print(b)
```

```
1
2
3
4
5
```

```
with open("c:/temp/data.bin", "rb") as f:
    byte = f.read(1)
    while byte != b"":
        print(byte)
        byte = f.read(1)
```

```
b'\x01'
b'\x02'
b'\x03'
b'\x04'
b'\x05'
```

## ❖ 파일 복사하기

```
infile = open('ABBA.mp3','rb’)
outfile = open('ABBA-copy.mp3','wb’)

data = infile.read() # bytes
outfile.write(data)

infile.close()
outfile.close()
```

## ❖ 문자열 읽기/쓰기

```
mytext = '이 일은 쉬운 일이 아닙니다.'
with open('c:/temp/mydata.bin','wb') as f:
    f.write(mytext.encode())

# reading str from binary file
with open('c:/temp/mydata.bin','rb') as f:
    bdata = f.read()
    mytext = bdata.decode()
    print(mytext)
```

## ❖ 수치데이터 읽기/쓰기

- struct 패키지의 pack(), unpack() 함수 사용
  - struct.pack(format, v1, v2, ...)
  - struct.unpack(format, buffer)
  - format
    - c: char
    - i(I): signed(unsigned) int
    - d: double
    - s: char[]

```
import struct

# packing numerical data into bytes
data = struct.pack("idd", 1, 10.3, -11.3)   # int, float, float
# unpacking bytes to numerical data
(i, x, y) = struct.unpack("idd", data)   # i=1, x = 10.3, y=-11.3
```

```
import struct

age = 59                # int
height = 172.5        # float
weight = 72.8         # float

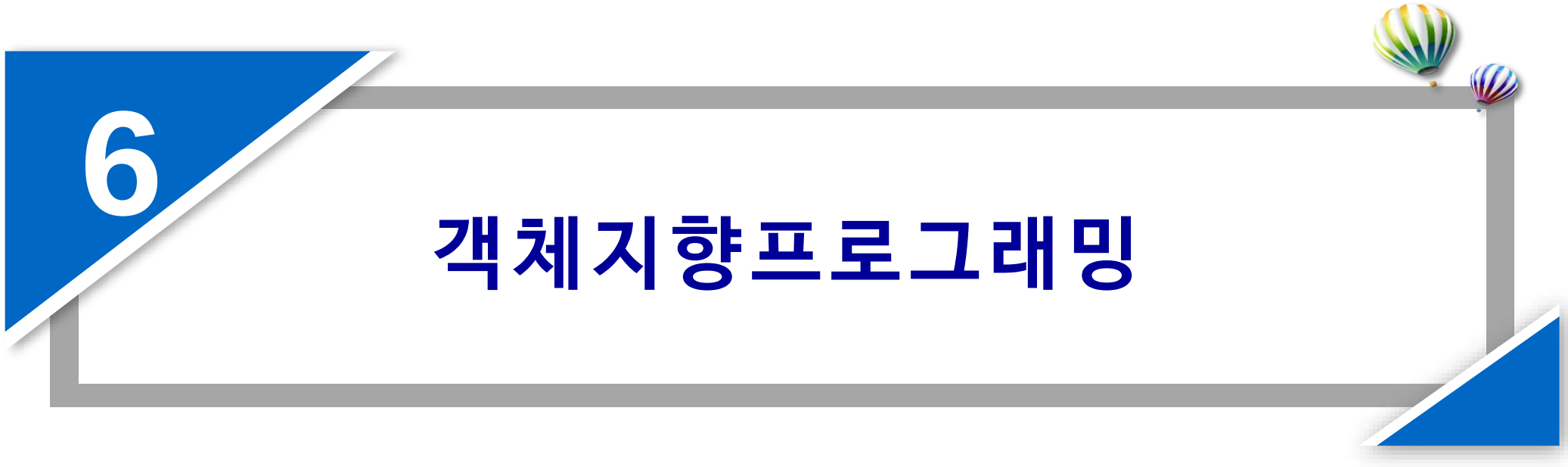
data = struct.pack('idd', age, height, weight)

with open('c:/temp/mydata.bin','wb') as outfile:
    outfile.write(data)
```

```
import struct

with open('c:/temp/mydata.bin','rb') as infile:
    data = infile.read()
    age, height, weight = struct.unpack('idd', data)
    print(age, height, weight)
```

```
59 172.5 72.8
```


6
객체지향프로그래밍

# Object-Oriented Programming

## ❖ 객체지향 프로그래밍

- 프로그램을 만들기 위해 객체들(objects)을 사용하는 프로그래밍 기법

## ❖ 객체

- 객체는 데이터와 함수들의 집합체

  object = attributes + behaviors

  = fields + methods

  = data + functions

  - 객체는 클래스의 생성자에 의해 메모리에 만들어지고 초기화 됨
  - 객체는 클래스의 인스턴스(instance), 즉, object == instance

## ❖ 클래스

- 비슷한 객체들의 형태를 추상화(abstraction)하여 정의한 것

# 이미 우리는 객체를 사용했다!!

❖ **(예)**

```
nums = [1, 2, 3]

nums.append(4)        # nums객체의 append 메소드
nums.insert(1,10)     # nums객체의 insert 메소드
nums.sort()           # nums객체의 sort 메소드

print(nums)
```

❖ **나만의 클래스 정의**

- class 키워드를 사용하여 정의
- class 내에 필드, 메소드, __init__() 생성자 메소드 등을 포함할 수 있음

# 캡슐화(Encapsulation)

```
# 클래스 정의
class Circle:
    def __init__(self, radius=1):
        self.radius = radius
    def getRadius(self):
        return self.radius
    def getArea(self):
        return 3.141592 * self.radius ** 2.0

# 클래스의 객체 생성 및 메소드 접근
circle1 = Circle()
print(f"반지름 {circle1.getRadius()}인 원의 면적은 {circle1.getArea()}")
circle2 = Circle(40)
print(f"반지름 {circle2.radius}인 원의 면적은 {circle2.getArea()}")
```

생성자 메소드(__init__)

self는 객체 자신을 참조하는 매개변수

필드(self.radius)를 정의

필드(self.radius)는 직접 접근이 가능

```
반지름 1인 원의 면적은 3.141592
반지름 40인 원의 면적은 5026.5472
```

```
반지름 1인 원의 면적은 3.141592
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
<ipython-input-46-89ce43c2ed1c> in <module>
     12 print(f"반지름 {circle1.getRadius()}인 원의 면적은 {circle1.getArea()}")
     13 circle2 = Circle(40)
---> 14 print(f"반지름 {circle2.radius()}인 원의 면적은 {circle2.getArea()}")

AttributeError: 'Circle' object has no attribute 'radius'
```

```python
# 클래스 정의
class Circle:
    def __init__(self, radius=1):
        self.__radius = radius
    def getRadius(self):
        return self.__radius
    def getArea(self):
        return 3.141592 * self.__radius ** 2.0

# 클래스의 객체 생성 및 메소드 접근
circle1 = Circle()
print(f"반지름 {circle1.getRadius()}인 원의 면적은 {circle1.getArea()}")
circle2 = Circle(40)
print(f"반지름 {circle2.radius}인 원의 면적은 {circle2.getArea()}")
```

필드(self.__radius)는 직접 접근이 불가능
즉, private 필드임

# 상속(inheritance)

## ❖ 상속은 확장성 있는 프로그램을 작성하는 도구

- 코드의 중복을 최소화하고, 기존 코드의 확장성 제공(코드의 재사용)
- 확장하는 대상
  - 새로운 필드나 메소드 추가
  - 기존 메소드의 재정의 등

## ❖ 상속 방법

```
class Parent:
    ...

class Child(Parent):
    ...
```

parent class / super class / base class

child class / sub class / derived class

# ❖ 상속의 예

```
class Shape:
    def __init__(self, color="black"):
        self.__color = color
    def setColor(self, color):
        self.__color = color
    def getColor(self):
        return self.__color
```

```
class Circle(Shape):
    def __init__(self, radius=1):
        super().__init__()
        self.__radius = radius
    def getRadius(self):
        return self.__radius
    def getArea(self):
        return 3.141592 * self.__radius ** 2.0
```

```
class Rectangle(Shape):
    def __init__(self, width=1, height=1):
        super().__init__()
        self.__width = width
        self.__height = height
    def getWidth(self):
        return self.__width
    def getHeight(slef):
        return self.__height
    def getArea(self):
        return self.__width * self.__height
```

```
class Shape:
    def __init__(self, color="black"):
        self.__color = color
    def setColor(self, color):
        self.__color = color
    def getColor(self):
        return self.__color

class Circle(Shape):
    def __init__(self, radius=1):
        super().__init__()
        self.__radius = radius
    def getRadius(self):
        return self.__radius
    def getArea(self):
        return 3.141592 * self.__radius ** 2.0

class Rectangle(Shape):
    def __init__(self, width=1, height=1):
        super().__init__()
        self.__width = width
        self.__height = height
    def getWidth(self):
        return self.__width
    def getHeight(slef):
        return self.__height
    def getArea(self):
        return self.__width * self.__height
```

```
c = Circle(5)
c.setColor("green")
print(f"{c.getColor()}색상 원의 면적은 {c.getArea()}")
r = Rectangle(20, 30)
r.setColor("blue")
print(f"{r.getColor()}색상 사각형의 면적은 {r.getArea()}")
```

```
green색상 원의 면적은 78.5398
blue색상 사각형의 면적은 600
```

# 다형성(polymorphism)

## ❖ 메소드 오버라이딩(overriding: 재정의)

- 부모 클래스의 메소드를 재정의

## ❖ 동적바인딩(dynamic binding)

- displayObejct(g)는 인스턴스를 인수로 받음
- **g.__str__()**로 어떤 클래스의 __str__()함수가 호출될 것인가?
  - 하위 클래스부터 상위클래스의 순으로 검색하면서 해당 메소드를 검색하여 최초로 찾은 메소드를 호출

```
class Shape:
    ...
class Circle(Shape):
    ...
class Rectangle(Shape):
    ...

def displayObject(g):
    print(g.__str__())

c = Circle(5)
r = Rectangle(20, 30)

displayObject(c)
displayObject(r)
```

## ❖ object 클래스

- 모든 클래스들의 부모 클래스
- 아래의 주요메소드는 재정의하여 사용

## ❖ obejct 클래스의 주요메소드

- **`__str__(self)`**
  - 객체에 대한 설명을 문자열로 반환(객체를 출력하는 경우에 자동으로 문자열로 변환되어 출력)

    (예) <__main__.Circle object at 0x0000020A04A33640>
- **`__eq__(self, other)`**
  - 두개의 객체가 서로 동일한지를 판단하여 True 또는 False를 반환

## ❖ 수학을 위한 특수메소드

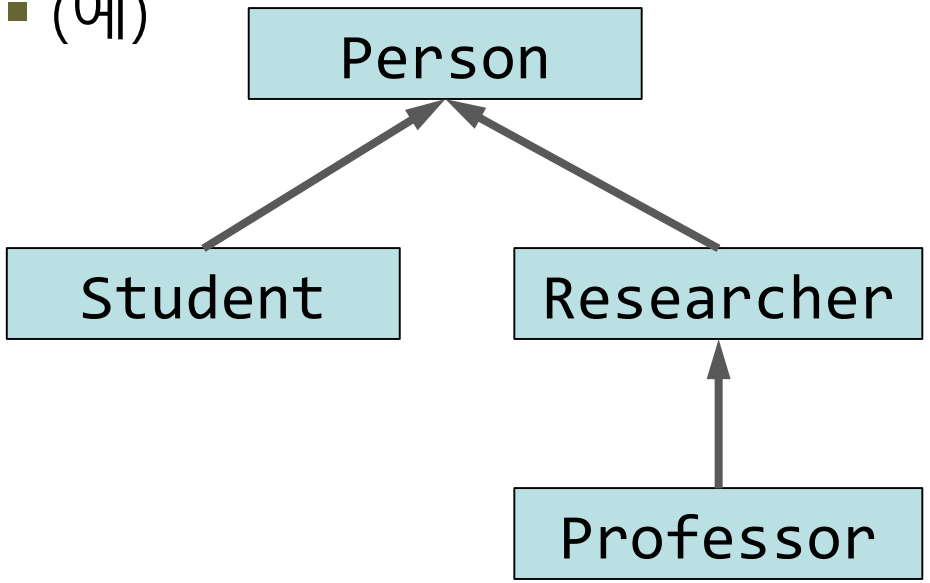
- a + b  a.__add__(b)
- a - b  a.__sub__(b)
- a * b  a.__mul__(b)
- a / b  a.__truediv__(b)
- a % b  a.__mod__(b)
- a & b  a.__and__(b)
- a ¦ b  a.__or__(b)
- a ^ b  a.__xor__(b)

## ❖ 항목접근을 위한 특수메소드

- len(x)  x.__len__()
- x[a]  x.__getitem__(a)
- x[a] = v  x.__setitem__(a,v)
- del x[a]  x.__delitem__(a)

## ❖ isinstance(obj, class) 함수

- 객체가 클래스의 인스턴스인지를 판별
- (예)




```
class Person: pass
class Student(Person): pass
class Researcher(Person): pass
class Professor(Researcher): pass

s = Student()
r = Researcher()
p = Professor()

print(isinstance(s, Student))
print(isinstance(s, Person))
print(isinstance(r, Professor))
print(isinstance(r, Researcher))
print(isinstance(r, Student))
print(isinstance(r, Person))
print(isinstance(p, Professor))
print(isinstance(p, Researcher))
print(isinstance(p, Student))
print(isinstance(p, Person))
```

```
True
True
False
True
False
True
True
True
False
True
```

# 7

# collections

# collections 모듈

- 리스트, 투플, 딕셔너리, 셋 등을 확장하여 대안을 제공
  - from collections import namedtuple
  - from collections import deque 등으로 임포트한 후 사용

| 클래스/함수 | 설명 |
|---|---|
| namedtuple() | 이름 붙은 필드를 갖는 튜플 서브 클래스를 만들기 위한 팩토리 함수 |
| deque | 양쪽 끝에서 빠르게 추가와 삭제를 할 수 있는 리스트류 컨테이너 |
| Counter | 해시 가능한 객체를 세는 데 사용하는 딕셔너리 서브 클래스 |
| OrderedDict | 항목이 추가된 순서를 기억하는 딕셔너리 서브 클래스 |
| defaultdict | 누락된 값을 제공하기 위해 팩토리 함수를 호출하는 딕셔너리 서브 클래스 |

# namedtuple()

- 데이터의 구조체를 저장하는 방법(C언어의 struct)
- 기존 tuple에 위치 인덱스 대신 이름으로 필드에 액세스하는 기능을 추가
  - namedtuple(*typename, field_names, ...*)
    - typename: Tuple의 서브클래스
    - field_name: 문자열의 시퀀스 (예) ['x', 'y']

```
from collections import namedtuple

Point = namedtuple('Point', ['x', 'y'])
p = Point(11, y=22)
print(p)

print(p.x + p.y)    # fields also accessible by name
print(p[0] + p[1])  # indexable like the plain tuple (11, 22)
x, y = p            # unpack like a regular tuple
print(x + y)
```

# deque(double-ended queue: 데크)

- **stack과 queue를 일반화한 클래스**
- **주요메소드**
  - append(x) : 데크의 오른쪽에 x를 추가
  - appendleft(x) : 데크의 왼쪽에 x를 추가
  - pop() : 데크의 오른쪽 요소를 제거하고 반환
  - popleft() : 데크의 왼쪽 요소를 제거하고 반환
  - clear() : 모든요소를 제거하고 길이가 0이 되도록 함
  - insert(i, x) : 데크의 i위치에 x를 삽입
  - rotate(n=1) : 데크를 n단계 오른쪽으로 회전(n이 음수면 왼쪽)
    - 오른쪽 회전: d.appendleft(d.pop())
    - 왼쪽 회전: d.append(d.popleft())

## ❖ 새로운 데크 만들기

```
from collections import deque

d = deque('Hello')
for c in d:
    print(c)
```

```
H
e
l
l
o
```

## ❖ Queue

```
from collections import deque

d = deque()
for i in range(1, 6):
    d.appendleft(i)
for i in range(len(d)):
    print(d.pop())
```

```
1
2
3
4
5
```

## ❖ Stack

```
from collections import deque
d = deque()
for i in range(1, 6):
    d.append(i)
for i in range(len(d)):
    print(d.pop())
```

```
5
4
3
2
1
```

## ❖ 환형 Queue

```
from collections import deque
d = deque()
for i in range(1, 6):
    d.append(i)
d.rotate(1)
print(d)
```

```
deque([5, 1, 2, 3, 4])
```

# Counter

## ❖ 시퀀스 자료형의 데이터값의 개수를 딕셔너리 형태로 반환

- 같은 값들이 몇 번 나타나는지 반환
- 요소가 딕셔너리 키로 저장되고 개수가 딕셔너리값으로 저장

```
text = list('sausages')
print(text)
c = Counter(text)
print(c)
print(c['a']) # 'a'의 개수
```

```
['s', 'a', 'u', 's', 'a', 'g', 'e', 's']
Counter({'s': 3, 'a': 2, 'u': 1, 'g': 1, 'e': 1})
2
```

```
text = "Many of the examples in this manual, even those entered at the intera
ctive prompt, include comments. Comments in Python start with the hash charac
ter, #, and extend to the end of the physical line. A comment may appear at t
he start of a line or following whitespace or code, but not within a string l
iteral. A hash character within a string literal is just a hash character. Si
nce comments are to clarify code and are not interpreted by Python, they may
be omitted when typing in examples.".lower().split()
c = Counter(text)
print(c)
```

```
Counter({'the': 6, 'a': 6, 'of': 3, 'in': 3, 'hash': 3, 'at': 2, 'comments':
2, 'start': 2, 'and': 2, 'to': 2, 'may': 2, 'or': 2, 'not': 2, 'within': 2,
'string': 2, 'are': 2, 'many': 1, 'examples': 1, 'this': 1, 'manual,': 1,
'even': 1, 'those': 1, 'entered': 1, 'interactive': 1, 'prompt,': 1,
'include': 1, 'comments.': 1, 'python': 1, 'with': 1, 'character,': 1, '#,':
1, 'extend': 1, 'end': 1, 'physical': 1, 'line.': 1, 'comment': 1, 'appear':
1, 'line': 1, 'following': 1, 'whitespace': 1, 'code,': 1, 'but': 1,
'literal.': 1, 'character': 1, 'literal': 1, 'is': 1, 'just': 1, 'character.':
1, 'since': 1, 'clarify': 1, 'code': 1, 'interpreted': 1, 'by': 1, 'python,':
1, 'they': 1, 'be': 1, 'omitted': 1, 'when': 1, 'typing': 1, 'examples.': 1})
```

# OrderedDict

## ❖ 순서를 가지 딕셔너리

- 기존 dict

```
d = {}
d['a'] = 100
d['b'] = 200
d['c'] = 300
d['d'] = 400
d['e'] = 500
for k, v in d.items():
    print(k, v)
```

```
a 100
b 200
d 400
e 500
c 300
```

저장된 순서 보장(X)

★ **OrderedDict**

```
from collections import OrderedDict

d = OrderedDict()
d['a'] = 100
d['b'] = 200
d['c'] = 300
d['d'] = 400
d['e'] = 500
for k, v in d.items():
    print(k, v)
```

```
a 100
b 200
c 300
d 400
e 500
```

## ❖ 딕셔너리 정렬(기존방법)

- key에 의한 정렬

```
dict = {'A' :1,'D' :2,'C' :3,'B' :4}
print(sorted(dict.items())) # by key
```

```
[('A', 1), ('B', 4), ('C', 3), ('D', 2)]
```

- value에 의한 정렬

```
import operator

dict = {'A' :1,'B' :4,'C' :3,'D' :2}
print(sorted(dict.items(), key=operator.itemgetter(1))) # by value
```

```
[('A', 1), ('D', 2), ('C', 3), ('B', 4)]
```

## ❖ 딕셔너리 정렬(OrderedDict) – 정렬결과를 유지하고 싶을 때

```
from collections import OrderedDict

def sort_by_key(e):
    return e[1] # 0:key, 1:value

d = {}
d['a'] = 300
d['b'] = 200
d['c'] = 500
d['d'] = 100
d['e'] = 400
for k, v in OrderedDict(sorted(d.items(), key=sort_by_key)).items():
    print(k, v)
```

operator.itemgetter(1)

# defaultdict

## ❖ 딕셔너리의 key의 value에 기본값을 지정하는 방법

```
from collections import defaultdict

d = defaultdict(lambda : 0) # default : 0
d['first']
d['second']
print(d['first'])
print(d['second'])
```

```
0
0
```

```
rom collections import defaultdict

d = defaultdict(list) # default : list
s = [('yellow', 1), ('blue', 2), ('yellow', 3), ('blue', 4), ('red', 1)]
for k, v in s:
    d[k].append(v)
print(d)
```

```
defaultdict(<class 'list'>, {'yellow': [1, 3], 'blue': [2, 4], 'red': [1]})
```

# 8

# list comprehension

# List Comprehension

- 리스트의 생성을 간편하게 할 수 있는 문법
- (예)

```
arr = []                          # 빈 리스트 생성
for i in range(10):               # 반복
    arr.append(i*2)                   # 빈 리스트에 요소 추가
print(arr)
```

```
[0, 2, 4, 6, 8, 10, 12, 14, 16, 18]
```

```
arr = [i*2 for i in range(10)]   # 한 줄로 리스트 생성 완료
print(arr)
```

```
[0, 2, 4, 6, 8, 10, 12, 14, 16, 18]
```

## ❖ 기본문법

**[ 변수활용 for 변수 in 시퀀스 ]**

(예) arr = [i*2 for i in range(10)]

arr = [n*n for n in arr]

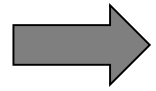
arr = [c*3 for c in "Hello"]

**[[변수활용 for 열변수 in 시퀀스] for 행변수 in 시퀀스]**

(예) arr = [[i+j for i in range(4)] for j in range(3)]

print(arr)

```
[[0, 1, 2, 3], [1, 2, 3, 4], [2, 3, 4, 5]]
```



```
[[0, 1, 2, 3],
 [1, 2, 3, 4],
 [2, 3, 4, 5]]
```

## ❖ 필터링

[ 참인경우 for 변수 in 시퀀스 if 조건 ]

[ 참인경우 if 조건 else 거짓인경우 for 변수 in 시퀀스 ]

(예)

```
arr = [i for i in range(10) if i%2==0]
print(arr)

arr = [i if i%2==0 else -i for i in range(10)]
print(arr)
```

```
[0, 2, 4, 6, 8]
[0, -1, 2, -3, 4, -5, 6, -7, 8, -9]
```

## ❖ 반복의 중첩(1)

```
arr = [c for c in range(4) for r in range(3)]
print("1차원 리스트", arr)
```

```
[0, 0, 0, 1, 1, 1, 2, 2, 2, 3, 3, 3]
```

```
arr = []
for c in range(4):
    for r in range(3):
        arr.append(c)

print("1차원 리스트", arr)
```

## ❖ 반복의 중첩(2)

```
arr = [[c] for c in range(4) for r in range(3)]
print("2차원 리스트(12x1)", arr)
```

```
2차원 리스트(12x1) [[0], [0], [0], [1], [1], [1], [2], [2], [2], [3], [3], [3]]
```

```
arr = []
for r in range(3):
    for c in range(4):
        arr.append(c)

print("2차원 리스트(12x1)", arr)
```

## ❖ 반복의 중첩(3)

```
arr = [[c for c in range(4)] for r in range(3)]
print("2차원 리스트(3x4)", arr)
```

```
2차원 리스트(3x4) [[0, 1, 2, 3], [0, 1, 2, 3], [0, 1, 2, 3]]
```

```
arr = []
for r in range(3):
    arr.append([c for c in range(4)])

print("2차원 리스트(3x4)", arr)
```

## ❖ 2차원 리스트의 편평화

```
arr = [[ 1,  2,  3],
       [ 4,  5,  6],
       [ 7,  8,  9],
       [10, 11, 12]]
flat_arr = [i for row in arr for i in row]
print(flat_arr)
```

```
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12]
```

```
arr = [[ 1,  2,  3],
       [ 4,  5,  6],
       [ 7,  8,  9],
       [10, 11, 12]]
flat_arr = []
for row in arr:
    for i in row:
        flat_arr.append(i)
print(flat_arr)
```

# Set Comprehension

- List Comprehension과 비슷함
- 단지, []를 {}로 바꾼것
- (예)

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

new_set = {i*i for i in arr}
print(new_set)
```

```
{16, 1, 4, 9}
```

# Dictionary Comprehension

## ❖ 문법

- {키:값 for 키:값 in 시퀀스 [if 조건]}

## ❖ 예

```
dict1 = {1:'first', 2:'second', 3:'third'}

dict2 = {val:key for key,val in dict1.items()}
print(dict2)
```

```
{'first': 1, 'second': 2, 'third': 3}
```

```
subjects = ['math', 'history', 'english', 'computer']
scores = [98, 80, 75, 100]
score_dict = {key:val for key,val in zip(subjects, scores)}
print(score_dict)
```

```
{'math': 98, 'history': 80, 'english': 75, 'computer': 100}
```

# 9

# enumerate, zip

# enumerate(iterable)

- 시퀀스(list, tuple, string 등) 데이터를 입력 받아, index를 포함하는 enumerate객체를 반환하는 함수

```
e = enumerate(['A', 'B', 'C'])
print(e)
for i in range(3):
    print(next(e))
```

```
<enumerate object at
0x0000020A04A51580>
(0, 'A')
(1, 'B')
(2, 'C')
```

## ❖ enumerate 객체 요소

```
for i in enumerate(['A', 'B', 'C']):
    print(i)
```

```
(0, 'A')
(1, 'B')
(2, 'C')
```

## ❖ enumerate 객체 요소의 index와 value

```
for i, v in enumerate(['A', 'B', 'C']):
    print(i, v)
```

```
0 A
1 B
2 C
```

## ❖ enumerate 객체 요소의 시작 index 지정하기

```
for i, v in enumerate(['A', 'B', 'C'], start=1):
    print(i, v)
```

```
1 A
2 B
3 C
```

# zip(*iterable)

- 여러 개의 시퀀스(iterable 객체)를 인자로 받고 각 객체가 담고 있는 원소를 튜플 형태로 차례로 접근할 수 있는 시퀀스를 반환
- 시퀀스들의 길이가 다를 경우 가장 짧은 객체를 기준으로 zip 객체 만들어짐

```
nums = [1, 2, 3]
letters = ['A', 'B', 'C' , 'D']
for pair in zip(nums, letters):
    print(pair)
```

```
(1, 'A')
(2, 'B')
(3, 'C')
```

## ❖ 병렬처리

```
for number, upper, lower in zip("12345", "ABCDE", "abcde"):
    print(number, upper, lower)
```

```
1 A a
2 B b
3 C c
4 D d
5 E e
```

## ❖ 사전변환

```
keys = [1, 2, 3]
values = ['A', 'B', 'C']
d = dict(zip(keys, values))
print(d)
```

```
{1: 'A', 2: 'B', 3: 'C'}
```

## ❖ unzipping

- zip()함수를 이용하되 unpacking 연산자(*)를 붙임

```
pairs = zip([1, 2, 3], ['A', 'B', 'C’])

# unzip
nums, letters = zip(*pairs)
print(nums)
print(letters)
```

```
(1, 2, 3)
('A', 'B', 'C')
```

# 10

# map, filter, reduce

# Lambda

## ❖ 구문

- **lambda 인수 : 식**
- 한 줄 짜리 축약형 함수, 자동 return
- if, while 등은 허용되지 않음

## ❖ (예1)

```
g = lambda x, y : x * y
print(g(2,3))
```

## ❖ (예2)

```
print((lambda x, y : x + y)(2,3))
```

# map(f, iterable)

- 시퀀스(iterable객체)의 각 요소에 함수(f)를 적용한 결과를 map 객체로 반환

```
arr = [1, 2, 3, 4]
f = lambda x: x*2
print(list(map(f, arr)))
```

```
[2, 4, 6, 8]
```

```
arr1 = [1, 2, 3, 4]
arr2 = [10, 20, 30, 40]
f = lambda x, y: (x*2, y*2)
print(list(map(f, arr1, arr2)))
```

```
[(2, 20), (4, 40), (6, 60), (8, 80)]
```

# filter(f, iterable)

- 시퀀스(iterable객체)의 각 요소를 함수(f)에 적용한 후, 반환값이 True인 것만 묶어서 반환

```
arr = [1, 2, 3, 4, 5, 6, 7, 8, 9]
f = lambda x: x%2==0
print(list(filter(f, arr)))
```

```
[2, 4, 6, 8]
```

```
arr = [1, 2, 3, 4, 5, 6, 7, 8, 9]
f = lambda x: True if x%2==0 else False
print(list(filter(f, arr)))
```

```
[2, 4, 6, 8]
```

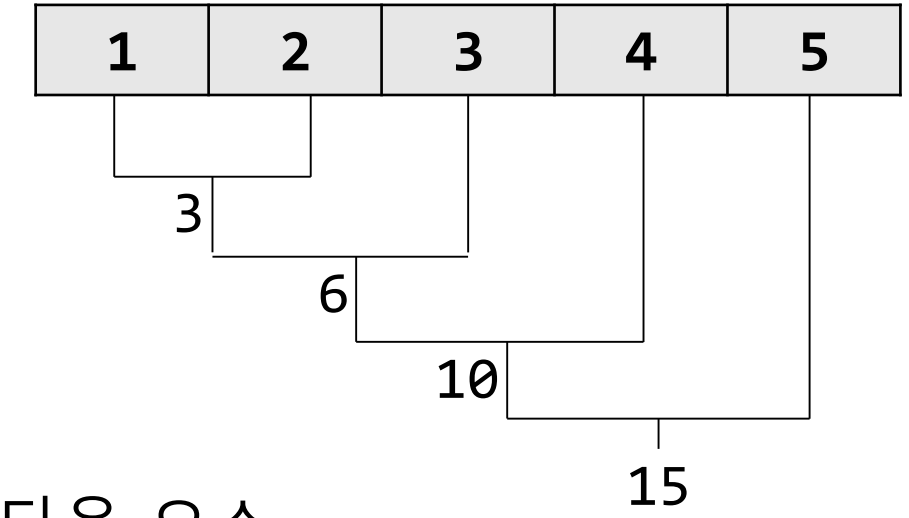
주의: lambda에 조건절을 넣을 때의 문법

# reduce(f, iterable[,initializer])

- **functools 패키지에 포함됨(버전3)**
  - from functools import reduce
- **시퀀스(iterable객체)의 각 요소를 차례대로 함수(f)에 적용하여한 후, 하나의 값으로 통합하여 반환**

```
from functools import reduce

arr = [1, 2, 3, 4, 5]
f = lambda x, y: x+y
print(reduce(f, arr))   # 15
```




  - lambda함수 인수 x는 이전계산결과, y는 다음 요소
  - initializer는 최초의 연산에 필요한 초기값(옵션)

# 11

# Generator

# 참고: Iteration

## ❖ 시퀀스 객체들은 iteration을 제공

```
a = 'hello’
for c in a: # a의 문자를 루핑
    ...

b = { 'name': 'Dave', 'password':'foo’}
for k in b: # 딕셔너리 키를 루핑
    ...

c = [1,2,3,4]
for i in c: # 리스트/튜플의 항목을 루핑
    ...

f = open('foo.txt’)
for x in f: # 파일의 행을 루핑
    ...
```

# Iteration의 작동방식

❖ (예)

```
for x in obj:
    # 문장
```

❖ 내부작동 방식

```
iter = obj.__iter__()           # 이터레이터 객체를 얻음
while True:
    try:
        x = iter.__next__()     # 다음 항목을 얻음
    except StopIteration:       # 남은 항목이 없음
        break
    # 문장 ...
```

# Generator 만들기

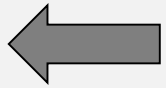
- **iterator를 생성해주는 함수(함수안에 yield 키워드 사용)**
  - yield : 값을 반환하면서, 코드의 실행을 함수의 바깥에 양보(중지)
    - 한 번 더 실행되면 실행되었던 yield의 다음코드가 실행
    - 시퀀스(iteration) 객체를 yield할 때에는 'yield from 시퀀스' 사용
- **(예)**

```
def countdown(n):
    while n > 0:
        yield n
        n = n - 1
```

```
for x in countdown(10):
    print(x, end=' ')
```

```
10 9 8 7 6 5 4 3 2 1
```

```
def countdown(n):
    while n > 0:
        yield n
        n = n - 1
```



generator

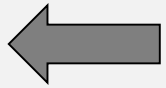
```
x = countdown(10)
print(x)
```

```
<generator object countdown at 0x0000023041B04648>
```

```
print(x.__next__())
print(x.__next__())
print(x.__next__())
print(x.__next__())
print(x.__next__())
print(x.__next__())
print(x.__next__())
print(x.__next__())
print(x.__next__())
print(x.__next__())
```

```
10
9
8
7
6
5
4
3
2
1
```

```
def countdown(n):
    while n > 0:
        yield n
        n = n - 1
```



generator

```
x = countdown(10)
print(x)
```

```
<generator object countdown at 0x0000023041B04648>
```

```
print(next(x))
print(next(x))
print(next(x))
print(next(x))
print(next(x))
print(next(x))
print(next(x))
print(next(x))
print(next(x))
print(next(x))
```

```
10
9
8
7
6
5
4
3
2
1
```

## ❖ yield from

```
def multi_generator():
    arr = [1, 2, 3, 4]
    yield from arr


g = multi_generator()
print(list(g))
```

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

```
def multi_generator():
    arr = [1, 2, 3, 4]
    for i in arr:
        yield i
```

## ❖ list comprehension방식의 generator

- 리스트는 생성하지 않음
- 재사용 불가(1회만 사용가능)

```
arr = [1, 2, 3, 4]
g = (2*x for x in arr)
print(g)

for i in g:
    print(i, end=' ')

for j in g:
    print(j, end=' ')
```

```
<generator object <genexpr> at 0x0000023041DB6248>
2 4 6 8
```

## ❖ Iteration을 지원하는 class 만들기

```
class MyIterCls:
    def __init__(self):
        self.data = []

    def __iter__(self):
        return self.data.__iter__()
    ...
```

```
seq = MyIterCls()
for e in seq:
    ...
```

# 12

# Decorator

# 참고: Closure

- **함수의 함수?**
- **클로저(closure)**
  - 외부함수에서 내부함수를 정의
  - 외부함수는 내부함수 객체를 반환

```
def outer(num):
    def inner():
        print(num)
    return inner

f1 = outer(3)
f2 = outer(4)
f1()
f2()
```

```
3
4
```

# Decorator

- 어떤 함수의 앞 또는 뒤에 로그를 달거나 어떤 조작을 하고 싶은 경우에 사용하는 재사용 가능한 Wrapper 메소드나 클래스
- @로 시작하는 구문

```
@decorator
def func(x, y):
    return x + y
```

## ❖ 어떤 함수의 실행에 걸린 시간을 측정하고 싶다면?

```
import time

def decorated_myfunc():
    start = time.time()
    print("Hello")
    end = time.time()
    print(f"Elaspe time : {end-start} seconds")

decorated_myfunc()
```

```
Hello
Elaspe time : 0.0009987354278564453 seconds
```

## ❖ 만일, 여러 개의 함수에 실행에 걸린 시간을 측정하고 싶다면?

- 방법1 : 위의 문장을 일일히 복사해서 넣어준다.
- 방법2 : 데코레이터를 사용한다.(좀 더 간편함)

```
import time

def time_decorator(func):
    def decorated():
        start = time.time()
        func()
        end = time.time()
        print(f"Elaspe time : {end-start} seconds")
    return decorated

@time_decorator
def myfunc():
    print("Hello")

myfunc()
```

```
Hello
Elaspe time : 0.0009987354278564453 seconds
```

## ❖ Decorator의 일반적인 형태

```
def decorator(func):
    def wrapper(*args, **kwargs):
        # 전처리 코드
        func(*args, **kwargs)
        # 후처리 코드
    return wrapper
```

```
@decorator
def myFunc():
    ...
```

```python
import time

def time_decorator(func):
    def decorated(*args, **kwargs):
        start = time.time()
        func(*args, **kwargs)
        end = time.time()
        print(f"Elaspe time : {end-start} seconds")
    return decorated

@time_decorator
def myfunc():
    print("Hello")

myfunc()
```

```
Hello
Elaspe time : 0.0009987354278564453 seconds
```

## ❖ class로 만들기

```
import time

class TimeDecorator:
    def __init__(self, func):
        self.func = func

    def __call__(self, *args, **kwargs):
        start = time.time()
        self.func(*args, **kwargs)
        end = time.time()
        print(f"Elaspe time : {end-start} seconds")
```

```
class MyClass:
    @TimeDecorator
    def func1():
        ...

    @TimeDecorator
    def func2():
        ...

obj = MyClass()
obj.func1()
obj.func2()
```

# 13

# assert

# assert

- 어떤 조건을 테스트하는 디버깅 보조 도구
- `assert <표현식> [, '진단 메시지']`
  - 참이 아니면 AssertionError 예외가 발생
  - (예1: 조건 검사)

```
assert price >= 0
assert isinstance(10, int), 'Expected int'
```

  - (예2: inline test)

```
def add(x, y):
    return x + y

assert add(2,2) == 4
```

# ❖ Contract Programming

- S/W 설계자가 S/W 구성요소의 정확한 인터페이스 사양을 정의

```
def add(x, y):
    assert isinstance(x, int), 'Expected int'
    assert isinstance(y, int), 'Expected int'
    return x + y

print(add(2, 3))
print(add(2.0, 3.0)) # AssertionError
```

```
AssertionError: Expected int
---------------------------------------------------------------------------
AssertionError                            Traceback (most recent call last)
<ipython-input-53-97791eed1081> in <module>
      5
      6 print(add(2, 3))
----> 7 print(add(2.0, 3.0))

<ipython-input-53-97791eed1081> in add(x, y)
      1 def add(x, y):
----> 2     assert isinstance(x, int), 'Expected int'
      3     assert isinstance(y, int), 'Expected int'
      4     return x + y
      5

AssertionError: Expected int
```