제09장
Python Programming

산기대학교
컴퓨터공학부
뇌리공학 연구소

학습 목표

- **Python 가상환경 구성**
- **Special data types**
  - list
  - dictionary
- **Object oriented programming in python**
- **How to import package, module**
  - matlab
  - graphics
- **Synchronization between threads**
- **Socket programming**

python 가상 개발 환경

Python에서의 가상 환경이란?

- 파이썬에서는 한 라이브러리에 대해 하나의 버전만 설치가 가능
- 여러 개의 프로젝트 진행 시, 각각의 프로젝트 작업을 바꿀 때마다 다른 버전의 라이브러리 필요 가능
- 이를 방지하기 위한 격리된 독립적인 가상 환경을 제공
- 일반적으로 프로젝트마다 다른 하나의 가상 환경을 생성한 후 작업을 시작
- 가상 환경의 대표적인 모듈 3가지
  - ✓ venv : Python 3.3 버전 이후 부터 기본 모듈에 포함됨
  - ✓ virtualenv : Python 2 버전부터 사용해오던 가상환경 라이브러리, Python 3에서도 사용가능
  - ✓ conda : Anaconda Python을 설치했을 시 사용할 수 있는 모듈
  - ✓ pyenv : pyenv의 경우 Python Version Manger임과 동시에 가상환경 기능을 플러그인 형태로 제공

pip - 패키지 매니저

- 파이썬의 각종 라이브러리들을 설치 및 관리해주는 패키지 매니저

- **pip 업그레이드**

```
$ pip install --upgrade pip
```

- **pip 패키지 검색**

```
$ pip search 검색
```

- **pip 설치 리스트 확인**

현재 추가된 가상환경 내에 라이브러리 목록을 보여주는 명령어

```
$ pip list
```

```
$ pip freeze
```

또는

```
$ pip freeze > requirements.txt
```

pip - 패키지 매니저

•일반적인 외부라이브러리 설치
```
$ pip install 패키지명
```

•파일에 기록된 패키지리스트 모두 설치하는 방법
```
$ pip install -r requirements.txt
```

•특정 버전을 설치
•기본적으로 버전정보가 입력되지 않으면 최신버전이 설치
```
$ pip install 패키지명==버전넘버(예:2.3.0)
```

•특정 버전 이상을 설치
```
$ pip install 패키지명>=버전넘버(예:2.3.0)
```

•특정라이브러리를 업그레이드
```
$ pip install --upgrade 패키지명
```

•특정라이브러리 삭제
```
$ pip uninstall 패키지
```

•라이브러리 정보 확인
```
$ pip show 패키지명
```

virtualenv

- pip 명령어로 모듈을 설치

```
$ pip install virtualenv
```

- virtualenv로 가상환경 생성

```
$ virtualenv 가상환경명
```

- 가상환경 구동
  - Mac 또는 리눅스에서 가상환경 구동 방법

```
$ source 가상환경명/bin/activate
```

  - Windows 가상환경 구동방법

```
가상환경명/Scripts/activate
```

  - 가상환경이 구동되면 터미널창의 프롬프트가 아래와 같이 변경 됨

```
(가상환경명) $
```

- 현 상태에서 pip 명령어로 필요한 패키지를 install 및 프로그램 실행
- 가상환경을 빠져나오는 명령어

```
$ deactivate
```

Anaconda

- **conda는 가상환경만을 구성 + 부가기능**
  - anaconda python의 정보를 확인하고,
  - pip와 마찬가지로 패키지 인스톨
  - 가상 환경을 설치

- **conda 명령어로 가상 환경 구성 방법**
  - create 가 가상 환경 구성

    $ conda create -n 가상환경명

    또는 pip 명령 사용 구성 가능 – 혼용 가능
  - 가상 환경을 활성화

    $ source activate 가상환경명
  - 가상 환경을 비활성화

    $ source deactivate

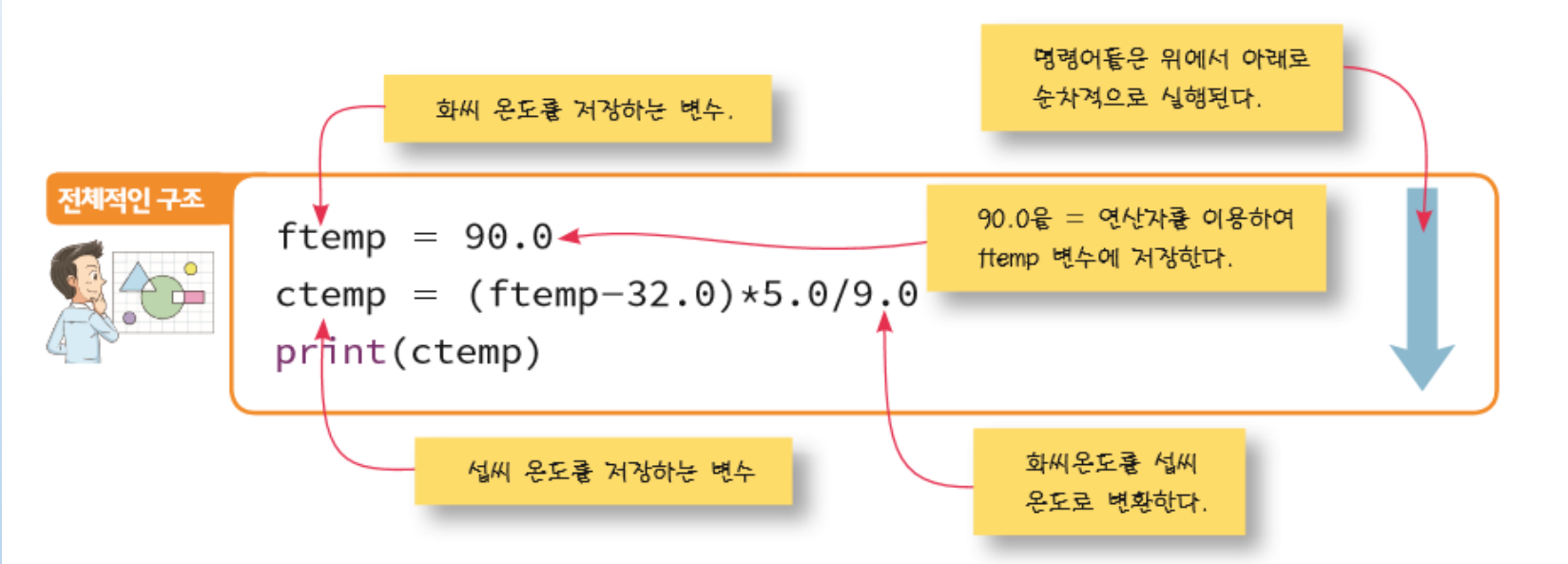
Python 언어란?

- **파이썬은 문법이 쉬워서 코드를 보면 직관적으로 알 수 있는 부분이 많다.**

```
if "사과" in ["딸기", "바나나", "포도", "사과"] :
        print("사과가 있습니다")
```

- **파이썬은 다양한 플랫폼에서 사용**
- **라이브러리가 풍부**
- **애니메이션이나 그래픽을 쉽게 사용**
- **파이썬을 설치**
  - http://www.python.org/에 접속하여 Download 메뉴에서 최신 버전 선택

Python 프로그램 예



print() 함수: 자료형 서식

print( ) 함수를 사용한 다양한 출력

표 3-1 print( ) 함수에서 사용할 수 있는 서식

서식	값의 예	설명
%d, %x, %o	10, 100, 1234	정수(10진수, 16진수, 8진수)
%f	0.5, 1.0, 3.14	실수(소수점이 붙은 수)
%c	"b", "한"	한 글자
%s	"안녕", "abcdefg", "a"	두 글자 이상인 문자열

표 3-2 이스케이프 문자

이스케이프 문자	역할	설명
\n	새로운 줄로 이동	Enter를 누른 효과
\t	다음 탭으로 이동	Tab을 누른 효과
\b	뒤로 한 칸 이동	Backspace를 누른 효과
\\	\ 출력	
\'	' 출력	
\"	" 출력	

```
print("%d / %d = %d" % (100, 200, 0.5))
```

결과는 100/200=0이다.

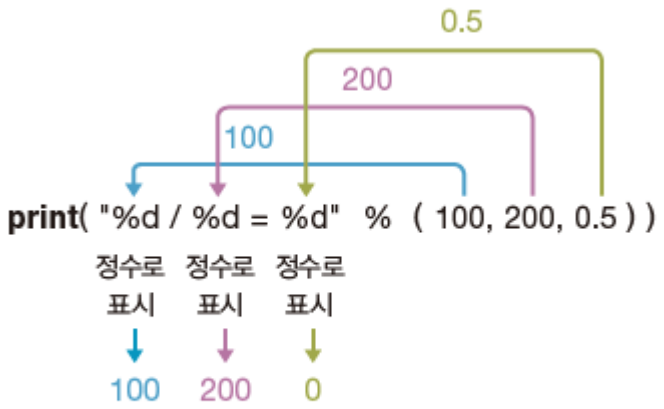


그림 3-2 서식과 숫자의 불일치 상황

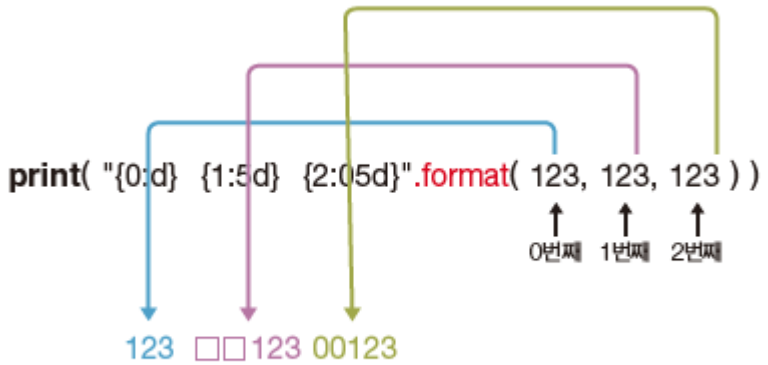


그림 3-6 format( ) 함수의 사용

```
print("%d  %5d  %05d" % (123, 123, 123))
print("{0:d}  {1:5d}  {2:05d}".format(123, 123, 123))
```

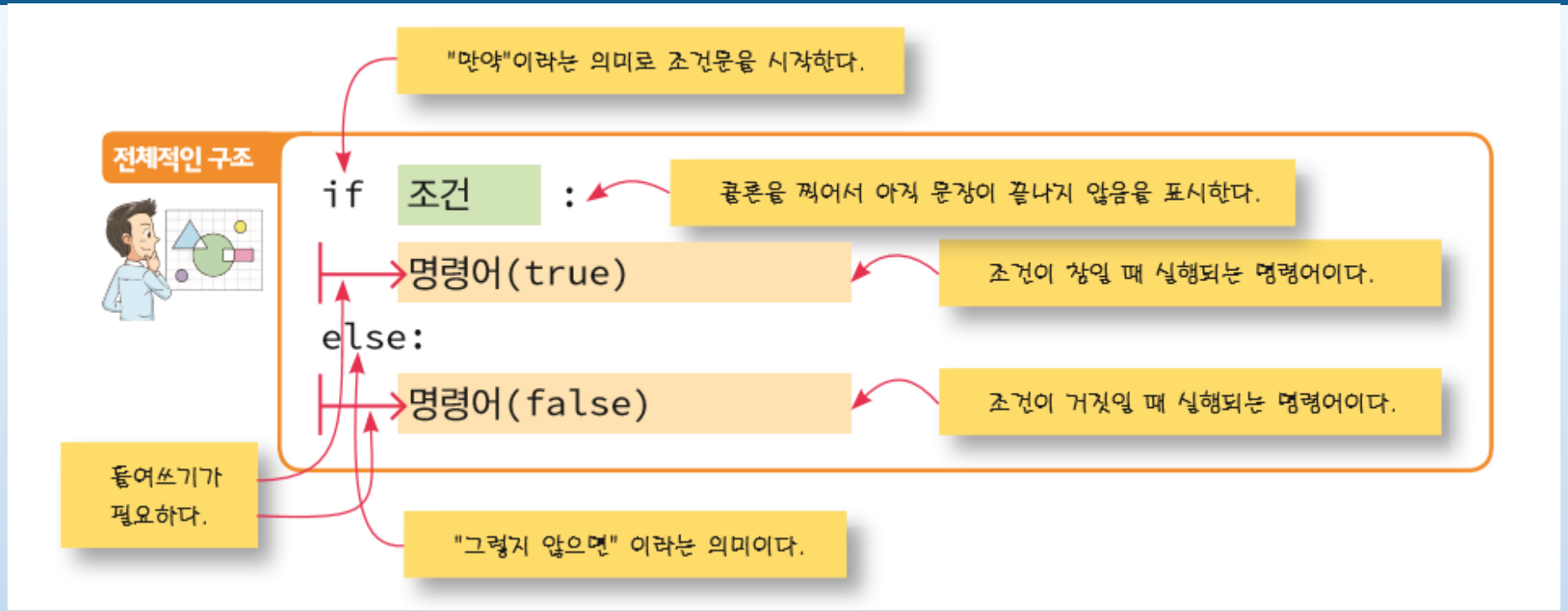
따라서 코드를 다음과 같이 수정

```
print("%d / %d = %5.1f" % (100, 200, 0.5))
```

표준 입력 함수

```
def calGrade (score) :
    .
    .
    .

kor_score = int (input ('국어 점수를 입력 하시오: '))
math_score = int (input ('수학 점수를 입력 하시오: '))
eng_score = int (input ('영어 점수를 입력 하시오: '))

kor_grade = calGrade (kor_score)
math_grade = calGrade (math_score)
eng_grade = calGrade (eng_score)

print ('국어 학점은 ', kor_grade, '입니다')
print ('수학 학점은 ', math_grade, '입니다')
print ('영어 학점은 ', eng_grade, '입니다')
```

조건문


"만약"이라는 의미로 조건문을 시작한다.
전체적인 구조
if 조건 :
콜론을 찍어서 아직 문장이 끝나지 않음을 표시한다.
명령어(true)
조건이 참일 때 실행되는 명령어이다.
else:
명령어(false)
조건이 거짓일 때 실행되는 명령어이다.
들여쓰기가 필요하다.
"그렇지 않으면" 이라는 의미이다.

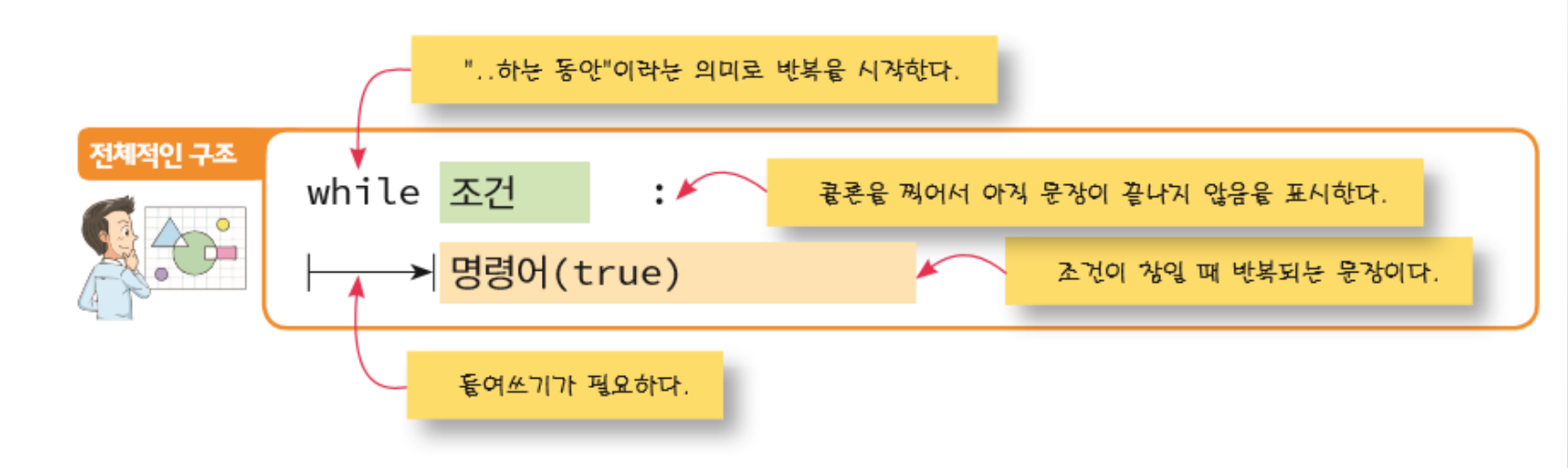
예

```
score = int(input("성적을 입력하시오: "))

if score >= 90 :
        print("학점 A")
elif score >= 80 :
        print("학점 B")
elif score >= 70 :
        print("학점 C")
elif score >= 60 :
        print("학점 D")
else :
        print("학점 F")
```

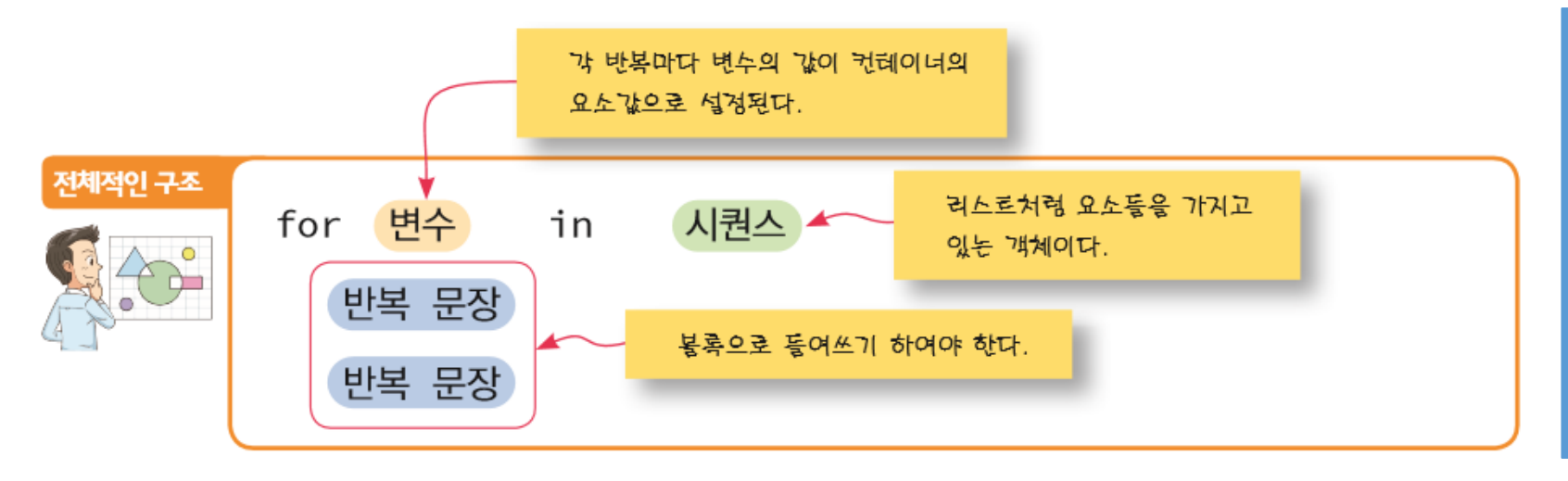
```
성적을 입력하시오: 90
학점 A
```

반복문

- while 문



- for 문



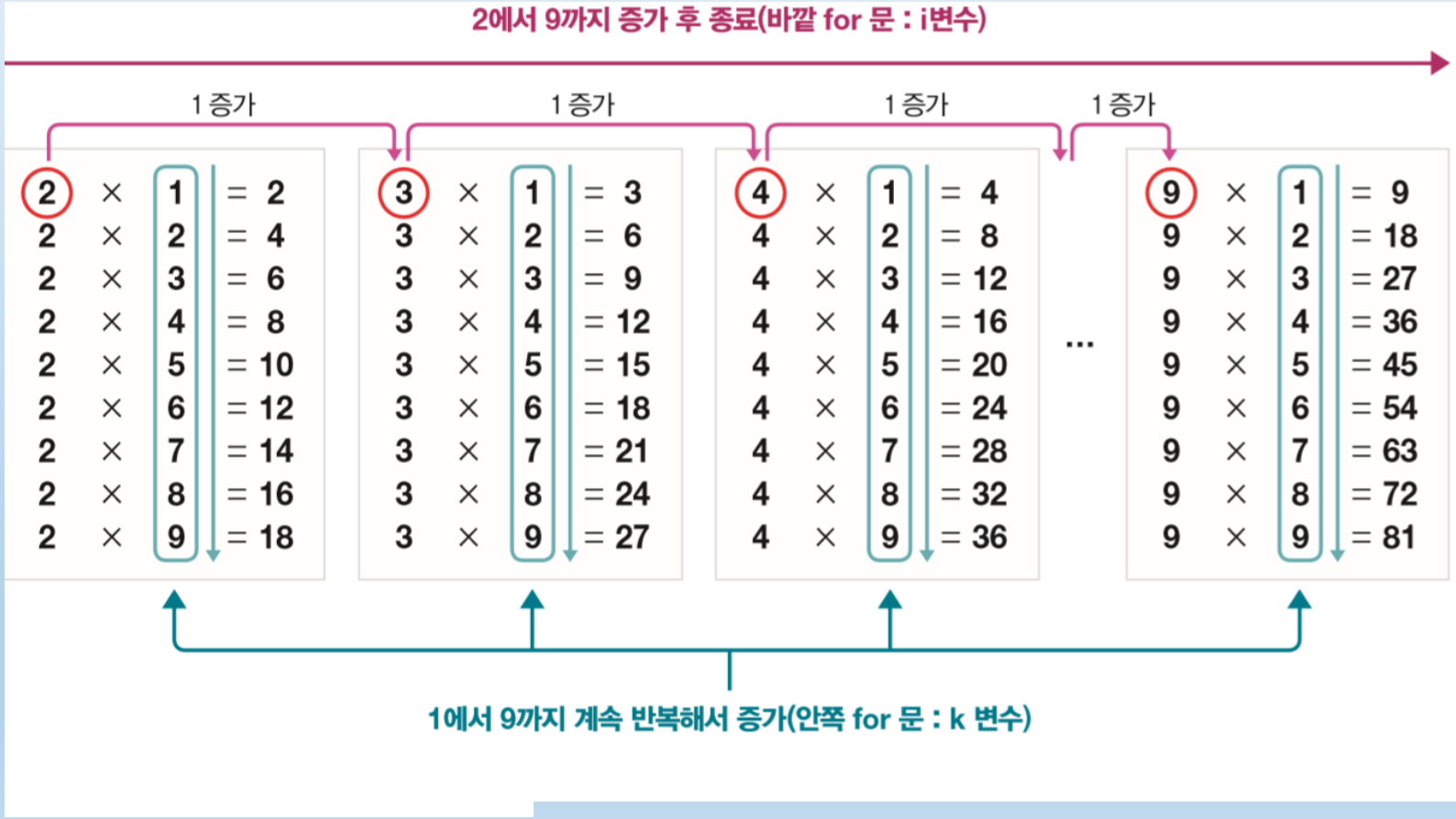
시퀀스 자료형:
순서를 가진 요소들의 집합
문자열
바이트 시퀀스
바이트 배열
리스트
튜플
range() 객체

```
i = 1
sum = 0;

while i <= 10:
        sum = sum + i
        i = i + 1

print("합계=", sum)
```

```
# 반복을 이용한 팩토리얼 구하기
fact=1.0
n = int (input ("정수를 입력하시오:  "))

for i in range(1, n+1) :
    fact = fact * i

print(n,"!은",  fact, "이다.")
```

실습:

1. 중첩 for 문의 활용: 2단부터 5단까지 구구단 출력 하시오.
2. 중첩 while 문을 사용하시오.

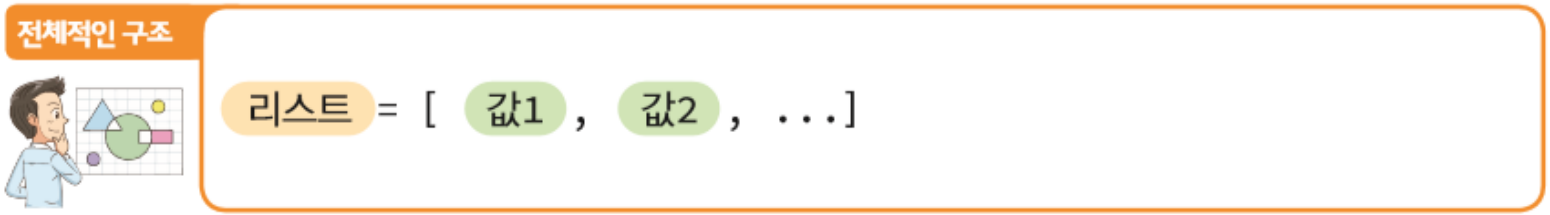


Special Data Types

- List
- Dictionary
- Tuple
- set

리스트란?

- 리스트(list)는 여러 개의 데이터가 저장되어 있는 장소이다.




```
scores  = [ 32, 56, 64, 72, 12, 37, 98, 77, 59, 69]
```

```
scores = [ 32, 56, 64, 72, 12, 37, 98, 77, 59, 69]

scores[0] = 80;
scores[1] = scores[0];

scores[i] = 10;                 // i는 정수 변수
scores[i+2] = 20;               // 수식이 인덱스가 된다.

if i >= 0 and i < len(scores) :
        scores[i] = number
```

```
>>> squares = [0, 1, 4, 9, 16, 25, 36, 49]
>>> squares[3:6]   # 슬라이싱은 새로운 리스트를 반환
[9, 16, 25]
```

리스트 관련 메소드

■ 리스트 조작 메소드

표 7-1 **리스트 조작 함수**

함수	설명	사용법
append()	리스트 맨 뒤에 항목을 추가한다.	리스트명.append(값)
pop()	리스트 맨 뒤의 항목을 빼낸다(리스트에서 해당 항목이 삭제된다).	리스트명.pop()
sort()	리스트의 항목을 정렬한다.	리스트명.sort()
reverse()	리스트 항목의 순서를 역순으로 만든다.	리스트명.reverse()
index()	지정한 값을 찾아 해당 위치를 반환한다.	리스트명.index(찾을값)
insert()	지정된 위치에 값을 삽입한다.	리스트명.insert(위치, 값)
remove()	리스트에서 지정한 값을 삭제한다. 단 지정한 값이 여러 개면 첫 번째 값만 지운다.	리스트명.remove(지울값)
extend()	리스트 뒤에 리스트를 추가한다. 리스트의 더하기(+) 연산과 기능이 동일하다.	리스트명.extend(추가할리스트)
count()	리스트에서 해당 값의 개수를 센다.	리스트명.count(찾을값)
clear()	리스트의 내용을 모두 지운다.	리스트명.clear()
del()	리스트에서 해당 위치의 항목을 삭제한다.	del(리스트명[위치])
len()	리스트에 포함된 전체 항목의 개수를 센다.	len(리스트명)
copy()	리스트의 내용을 새로운 리스트에 복사한다.	새리스트=리스트명.copy()
sorted()	리스트의 항목을 정렬해서 새로운 리스트에 대입한다.	새리스트=sorted(리스트)

리스트의 기초 연산

- 두개의 리스트를 합칠 때는 연결 연산자인 + 연산자를 사용할 수 있다.

```
>>> marvel_heroes  = [ "스파이더맨", "헐크", "아이언맨" ]
>>> dc_heroes  = [ "슈퍼맨", "배트맨", "원더우먼" ]
>>> heroes =  marvel_heroes + dc_heroes
>>> heroes
['스파이더맨', '헐크', '아이언맨', '슈퍼맨', '배트맨', '원더우먼']
```

- 리스트의 반복

```
>>> values = [ 1, 2, 3 ] * 3
>>> values
[1, 2, 3, 1, 2, 3, 1, 2, 3]
```

리스트의 기초 연산

- **len() 연산은 리스트의 길이를 계산하여 반환**

```
>>> letters = ['a', 'b', 'c', 'd']
>>> len(letters)
4
```

- **append()를 사용하여서 리스트의 끝에 새로운 항목을 추가**

```
>>> shopping_list = [ ]
>>> shopping_list.append("두부")
>>> shopping_list.append("양배추")
>>> shopping_list.append("딸기")

>>> shopping_list
['두부', '양배추', '딸기']
```

리스트의 기초 연산

- 어떤 요소가 리스트에 있는지/없는지만 알려면 in 연산자를 사용

```
heroes  = [ "스파이더맨", "슈퍼맨", "헐크", "아이언맨", "배트맨" ]
if "배트맨" in heroes :
        print("배트맨은 영웅입니다. ")
```

- 어떤 요소의 리스트 안에서의 위치를 알려면 index()을 사용

```
heroes  = [ "스파이더맨", "슈퍼맨", "헐크", "아이언맨", "배트맨" ]
index = heroes.index("슈퍼맨")      # index는 1이 된다.
```

- 리스트 비교: 연산자 ==, !=, >, <를 사용하여서 2개의 리스트를 비교

```
>>> list1 = [ 1, 2, 3 ]
>>> list2 = [ 1, 2, 3 ]
>>> list1 == list2
True
```

리스트의 기초 연산

- **pop() 메소드는 특정한 위치에 있는 항목을 삭제**

```
>>> heroes  = [ "스파이더맨", "슈퍼맨", "헐크", "아이언맨", "배트맨" ]
>>> heroes.pop(1)
'슈퍼맨'
>>> heroes
['스파이더맨', '헐크', '아이언맨', '배트맨']
```

```
>>> heroes  = [ "스파이더맨", "슈퍼맨", "헐크", "아이언맨", "배트맨", "조커" ]
>>> heroes.remove("조커")
>>> heroes
['스파이더맨', '슈퍼맨', '헐크', '아이언맨', '배트맨']
```

동적으로 2차원 리스트 생성

- **중첩에 의한 2차원 리스트 생성**

```
# 동적으로 2차원 리스트를 생성한다.
rows = 3
cols = 5

s = [ ]
for row in range(rows):
    s += [[0]*cols]

print("s =", s)
```

```
s = [[0, 0, 0, 0, 0], [0, 0, 0, 0, 0], [0, 0, 0, 0, 0]]
```

```
s = [
        [ 1, 2, 3, 4, 5 ] ,
        [ 6, 7, 8, 9, 10 ],
        [11, 12, 13, 14, 15 ]
]

# 행과 열의 개수를 구한다.
rows = len(s)
cols = len(s[0])

for r in range(rows):
    for c in range(cols):
        print(s[r][c], end=",")
    print()
```

```
1,2,3,4,5,
6,7,8,9,10,
11,12,13,14,15,
```

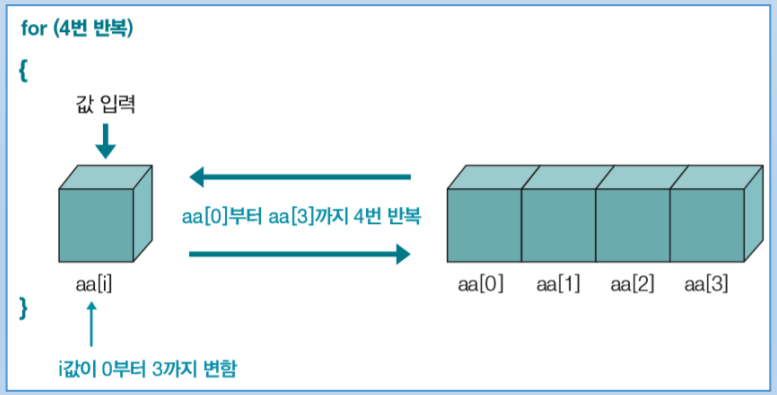
- **함축에 의한 리스트 생성**

```
## m x n list with init. val 0

matrix = [[0 for col in range(n)] for row in range(m)]
```

실습:

- **다음 조건을 만족하는 프로그램을 작성하시오.**

1. main () 정의
   - ① 리스트를 초기값 없이 생성
   - ② 리스트를 인자로 sum_list () 호출
   - ③ 리스트의 값을 화면에 출력
2. sum_list () 정의
   - ① for문을 반복하여 aa[i]의 첨자를 aa[0]에서 aa[3]까지 변하게 하며 값을 키보드로부터 입력
   - ② 리스트의 값을 합산
   - ③ 리스트 반환



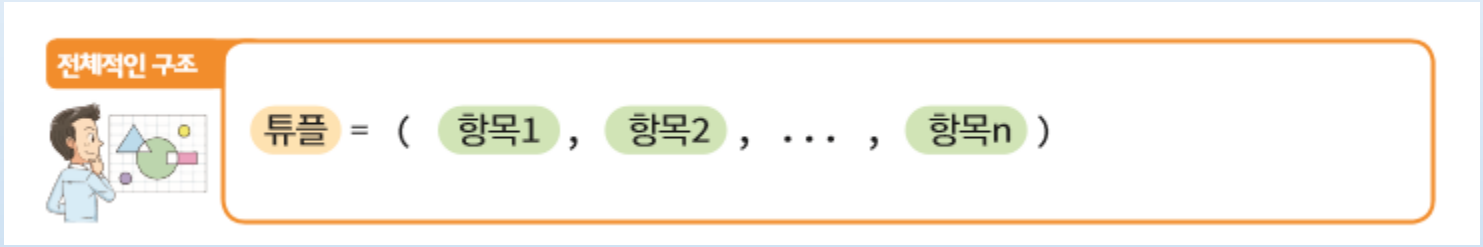

```
def main():
    . . .
    sum_ist( . . .)
    print( . . .)

def sum_list (. . . ) :
    . . .

if __name__ == "__main__":
    main()
```

출력 결과

```
1번째 숫자 : 10
2번째 숫자 : 20
3번째 숫자 : 30
4번째 숫자 : 40
합계 ==> 100
```

튜플

- **튜플(tuple)은 변경될 수 없는 리스트**
  - 추가/삭제 불가
  - 복제 가능



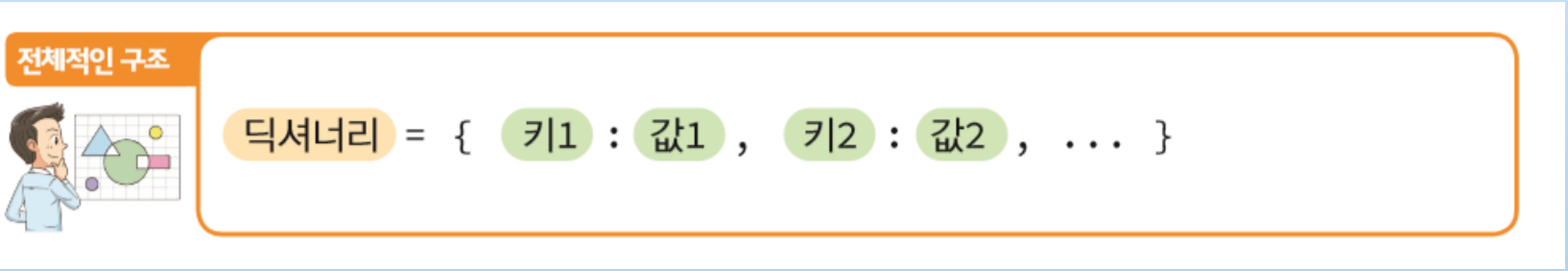
  - 소괄호 생략 가능
    - ✓ t2 = 10, 20, 30

```
>>> colors = ("red", "green", "blue")
>>> colors
('red', 'green', 'blue')

>>> numbers = (1, 2, 3, 4, 5 )
>>> numbers
(1, 2, 3, 4, 5)
>>> fruits = “apple“, “persimmon“, “pear“, “dates“
>>>fruits
 ('apple', 'persimmon', 'pear', 'dates')
```

Dictionary

- 딕셔너리는 키(key)와 값(value)의 쌍을 저장할 수 있는 객체
  - 중복되는 키 사용 불가!

키(key)	값(value)
"Kim"	"01012345678"
"Park"	"01012345679"
"Lee"	"01012345680"




```
>>> contacts = {'Kim':'01012345678',  'Park':'01012345679', 'Lee':'01012345680' }

>>> contacts
{'Kim': '01012345678', 'Lee': '01012345680', 'Park': '01012345679'}
```

항목 접근

```
>>> contacts = {'Kim':'01012345678',  'Park':'01012345679', 'Lee':'01012345680' }

>>> contacts['Kim']
'01012345678’

>>> contacts.get('Kim')
'01012345678'
```

```
>>> if "Kim" in contacts:
        print("키가 딕셔너리에 있음")
```

항목 추가 & 삭제

- dic[key] = value 의 형식으로 추가
- del dic[key] 의 형식으로 삭제

```
>>> contacts['Choi'] = '01056781234'
>>> contacts
{'Kim': '01012345678', 'Choi': '01056781234', 'Lee': '01012345680', 'Park': '01012345679'}
```

```
>>> contacts = {'Kim':'01012345678',  'Park':'01012345679', 'Lee':'01012345680' }

>>> contacts.pop("Kim")
'01012345678'

>>> contacts
{'Lee': '01012345680', 'Park': '01012345679'}
```

항목 순회

- 항목을 튜플로 출력

```
scores = { 'Korean': 80, 'Math': 90, 'English': 80}
for item in scores.items():
    print(item)
    print(item[0],"-", item[1])
```

```
('Korean', 80)
('Korean', '-', 80)
('Math', 90)
('Math', '-', 80)
('English', 80)
('English', '-', 80)
```

- 키를 출력, 값을 출력

```
scores = { 'Korean': 80, 'Math': 90, 'English': 80}

## To get only each "key" in dic
print('keys:')
for key in scores.keys(): ## == for key in scores:
    print(key)

## To get values
print('\nvalues:')
for val in scores.values ():
    print(val)
```

```
$ python dic.py
keys:
Korean
Math
English

values:
80
90
80
```

딕셔너리 함축과 정렬

- **함축**

```
### dic. comprehension
## dic = { item for key in sequence }
triples = { x: x**3 for x in range (6) }
print ('Comprehension: ' + str(triples))
```

```
Comprehension: {0: 0, 1: 1, 2: 8, 3: 27, 4: 64, 5: 125}
```

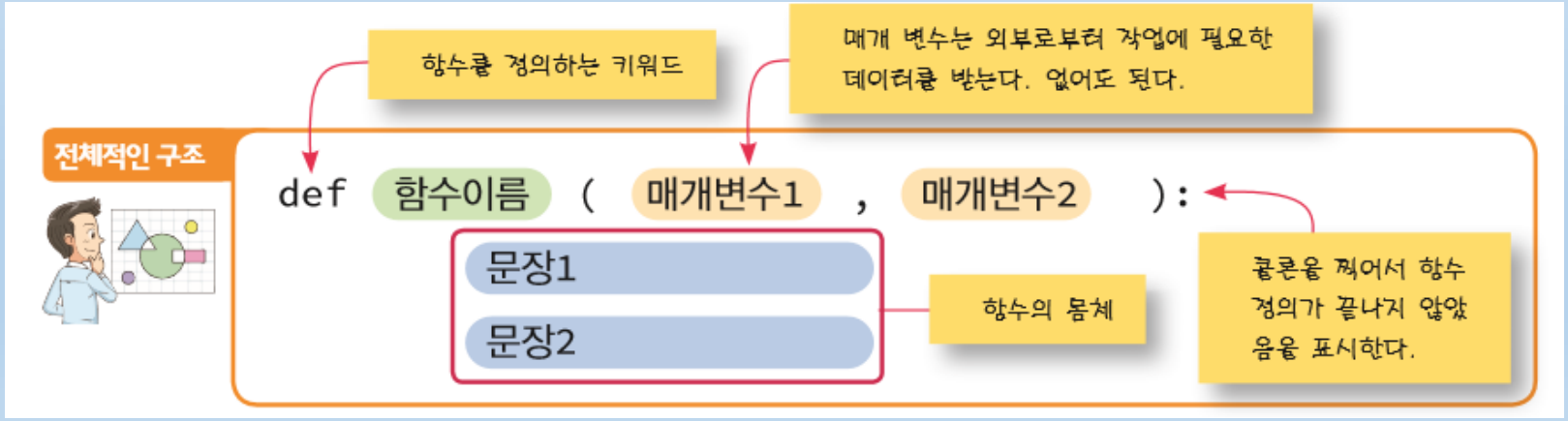
- **정렬**

```
### Sorting of dic
sorted_scores = sorted (scores)
print ('scores: ' + str(scores)) ## No change on org dic.
print ('Sorting keys: ' + str(sorted_scores))

sortedVal_scores = sorted (scores.values ())
print ('Sorting values: ' + str(sortedVal_scores))
```

```
scores: {'Korean': 80, 'Math': 90, 'English': 80}
Sorting keys: ['English', 'Korean', 'Math']
Sorting values: [80, 80, 90]
```

함수 정의

- **함수이름: 의미 있는 식별자 사용**
- **함수 호출 전에 반함수 정의들 간에는 순서 무시 가능**
- **드시 함수 정의**
- **매개변수(parameter):**
  - 의미 있는 식별자 사용
  - 호출자가 호출 된 함수에게 전달하는 데이터들
  - 임의의 개수 매개변수 가능
  - 함수가 호출 될 때 주어지는 변수들은 인자(argument)라 부른다.




```
def power(x, y):
    result = 1
    for i in range(y):
        result = result * x
    return result

print(power(10, 2))
```

```
100
```

함수의 인자 전달

- **함수를 호출할 때, 변수를 전달하는 2가지 방법**
  - 값에 의한 호출**(call-by-value)**
    - ✓ immutable objects
  - 참조에 의한 호출 (call-by-reference)
    - ✓ mutable objects
- Mutable object means you can change their content without changing their identity
  - list, dict, set, bytearray, user-defined classes (unless specifically made immutable)
- Immutable objects can not be changed their content without their identity
  - int, float, decimal, complex, bool, string, tuple, range, frozenset, bytes

● **일반적 프로그래밍 언어 함수 인자 전달 방법**

- ❖ Call by value
  - ✓ 모든 언어 지원
- ❖ Call by reference
  - ✓ Call by address reference – C/C++ 지원
  - ✓ Call by nickname reference – C++, Java, Python 지원

함수의 데이터 전달

- **값에 의한 호출**

```
def modify1(s):
    s += "To You"

msg = "Happy Birthday"
print("msg=", msg)
modify1(msg)
print("msg=", msg)
```

```
msg= Happy Birthday
msg= Happy Birthday
```

참조에 의한 호출

```
def modify2 (li):
    li += [100, 200]

list = [1, 2, 3, 4, 5]
print (list)
modify2 (list)
print (list)
```

```
[1, 2, 3, 4, 5]
[1, 2, 3, 4, 5, 100, 200]
```

다중 반환(튜플반환)

```
def sub():
    return 1, 2, 3

a, b, c = sub()
print(a, b, c)
```

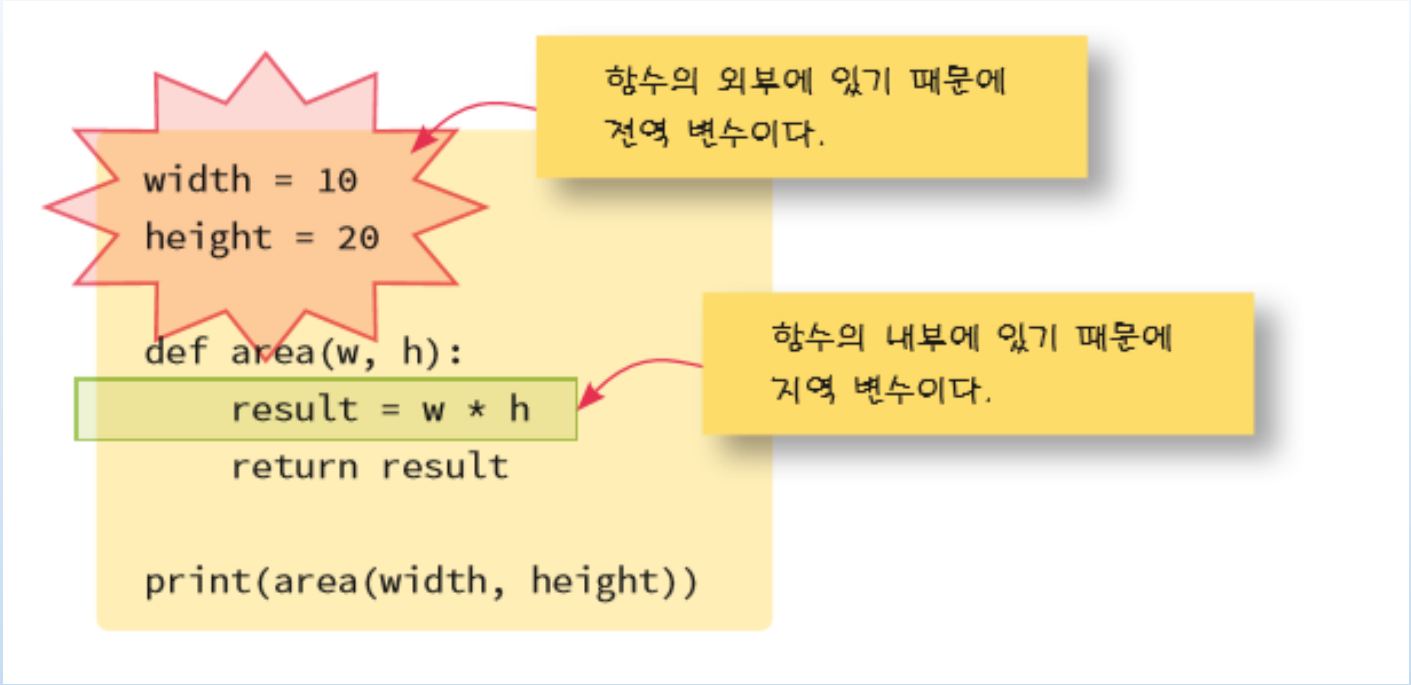
```
1 2 3
```

실습:

- **학점을 계산하는 다음 프로그램을 작성하시오.**

```
def calGrade (score) :
    .
    .
    .

kor_score = int (input ('국어 점수를 입력 하시오: '))
math_score = int (input ('수학 점수를 입력 하시오: '))
eng_score = int (input ('영어 점수를 입력 하시오: '))

kor_grade = calGrade (kor_score)
math_grade = calGrade (math_score)
eng_grade = calGrade (eng_score)

print ('국어 학점은 ', kor_grade, '입니다')
print ('수학 학점은 ', math_grade, '입니다')
print ('영어 학점은 ', eng_grade, '입니다')
```

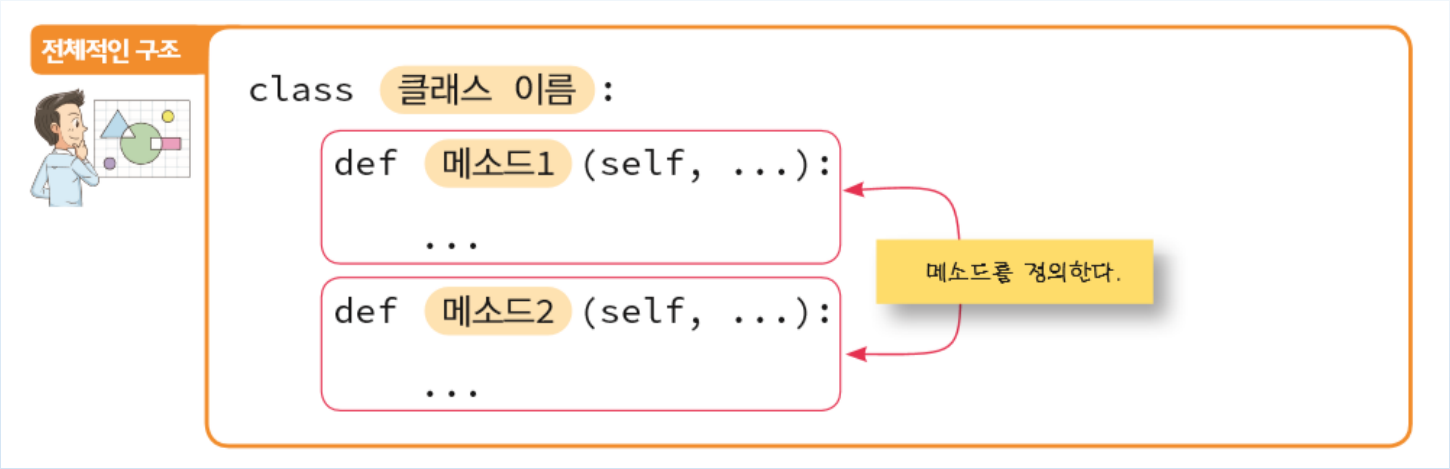
변수의 유효영역(scope): 지역 변수와 전역 변수




```
def sub():
        global s

        print(s)
        s = "바나나가 좋음!"
        print(s)

s = "사과가 좋음!"
sub()
print(s)
```

```
사과가 좋음!
바나나가 좋음!
바나나가 좋음!
```

Object Oriented Programming in Python



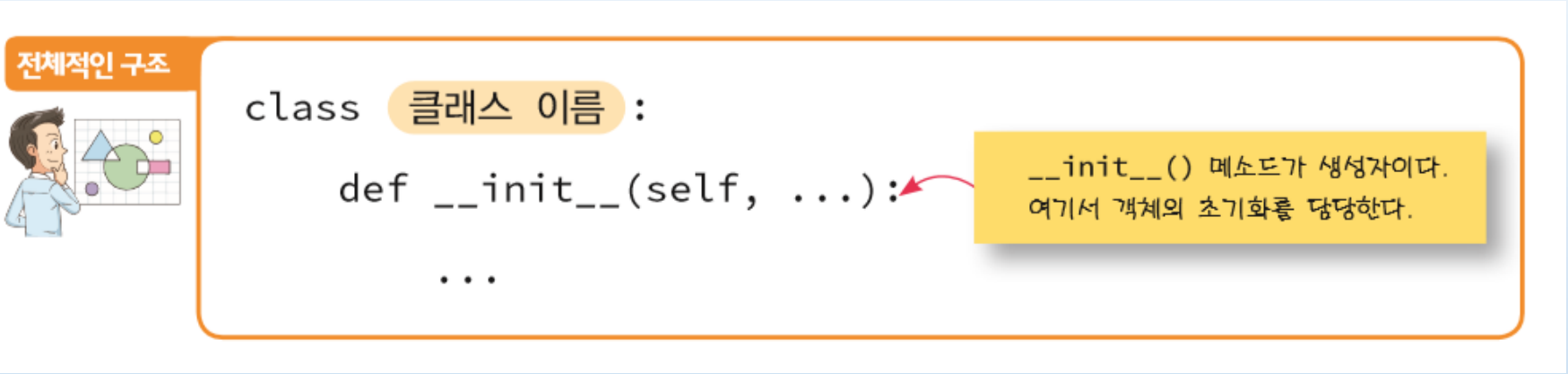

```
class Counter:
    def reset(self):
        self.count = 0
    def increment(self):
        self.count += 1
    def get(self):
        return self.count
```

```
a = Counter()

a.reset()
a.increment()
print("카운터 a의 값은", a.get())
```

```
카운터 a의 값은 1
```

생성자

- 생성자(constructor)는 객체가 생성될 때 객체를 기본값으로 초기화하는 특수한 메소드




```
class Counter:
    def __init__(self) :
        self.count = 0
    def reset(self) :
        self.count = 0
    def increment(self):
        self.count += 1
    def get(self):
        return self.count
```

메소드 정의

- 메소드는 클래스 안에 정의된 함수이므로 함수를 정의하는 것과 아주 유사
- 첫 번째 매개변수는 항상 self

```
class Television:
        def __init__(self, channel, volume, on):
                self.channel = channel
                self.volume = volume
                self.on = on

        def show(self):
                print(self.channel, self.volume, self.on)

        def setChannel(self, channel):
                self.channel = channel

        def getChannel(self):
                return self.channel
```

- 메소드호출

```
t = Television(9, 10, True)

t.show()
t.setChannel(11)
t.show()
```

```
9 10 True
11 10 True
```

Access Modifier (접근 제한자)

- **Private variable**

```
class Student:
    def __init__(self, name=None, age=0):
            self.__name = name
        self.__age = age

obj=Student()
print(obj.__age)
```

```
...
AttributeError: 'Student' object has no attribute '__age'
```

- **Protected variable**

```
class Cup:
    def __init__(self, color):
        self._color = color        # protected variable
        self.__content = None      # private variable

    def fill(self, beverage):
        self.__content = beverage

    def empty(self):
        self.__content = None
```

객체를 함수로 전달할 때

- 우리가 작성한 객체가 전달되면 함수가 객체를 변경 가능
- User defined class
- Mutable object

```
# 사각형을 클래스로 정의
class Rectangle:
    def __init__(self, side=0):
                self.side = side

    def getArea(self):
                return self.side*self.side

# 사각형 객체와 반복횟수를 받아서 변을 증가시키면서 면적을 출력
def printAreas(r, n):
    while n >= 1:
                print(r.side, "\t", r.getArea())
        r.side = r.side + 1
        n = n - 1
```

객체를 함수로 전달할 때

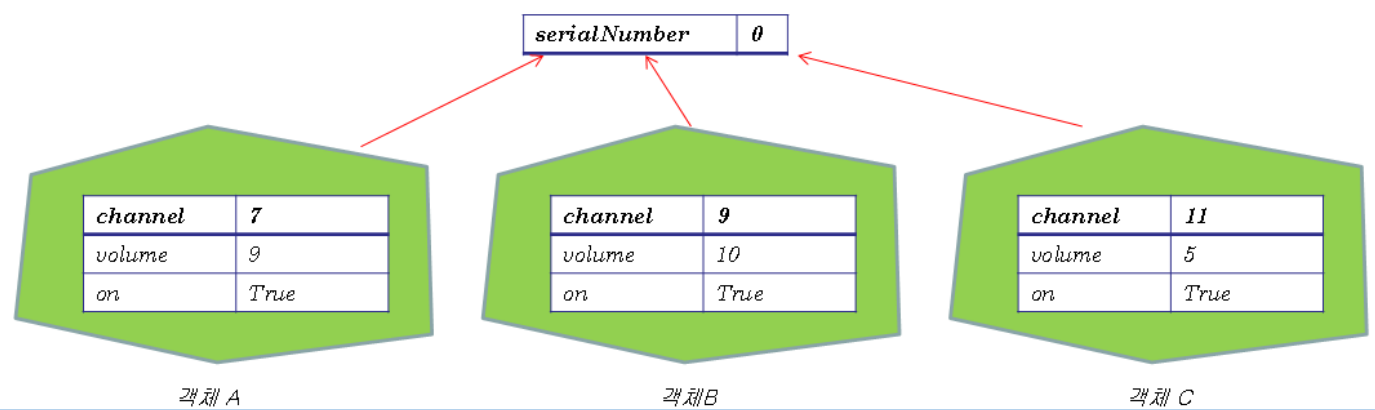
```
# printAreas()을 호출하여서 객체의 내용이 변경되는지를 확인
myRect = Rectangle();
count = 5
printAreas(myRect, count)
print("사각형의 변=", myRect.side)
print("반복횟수=", count)
```

```
0         0
1         1
2         4
3         9
4         16
사각형의 변= 5
반복횟수= 5
```

정적 변수

- 이들 변수는 모든 객체를 통틀어서 하나만 생성되고 모든 객체가 이것을 공유하게 된다. 이러한 변수를 정적 멤버 또는 클래스 멤버(class member)라고 한다.

```
class Television:
    serialNumber = 0     # 정적 변수
    def __init__(self):
        Television.serialNumber += 1
    self.number = Television.serialNumber
                  ...
```



특수 메소드

- 파이썬에는 연산자(+, -, *, /)에 관련된 특수 메소드(special method)가 있다.

```
class Circle:
        ...
        def __eq__(self, other):
                return self.radius == other.radius

c1 = Circle(10)
c2 = Circle(10)
if c1 == c2:
        print("원의 반지름은 동일합니다. ")
```

연산자	메소드	설명
x + y	__add__(self, y)	덧셈
x - y	__sub__(self, y)	뺄셈
x * y	__mul__(self, y)	곱셈
x / y	__truediv__(self, y)	실수나눗셈
x // y	__floordiv__(self, y)	정수나눗셈
x % y	__mod__(self, y)	나머지
divmod(x, y)	__divmod__(self, y)	실수나눗셈과 나머지
x ** y	__pow__(self, y)	지수
x << y	__lshift__(self, y)	왼쪽 비트 이동
x >> y	__rshift__(self, y)	오른쪽 비트 이동
x <= y	__le__(self, y)	less than or equal(작거나 같다)
x < y	__lt__(self, y)	less than(작다)
x >= y	__ge__(self, y)	greater than or equal(크거나 같다)
x > y	__gt__(self, y)	greater than(크다)
x == y	__eq__(self, y)	같다
x != y	__neq__(self, y)	같지않다

예제

```
class Vector2D :
    def __init__(self, x, y):
        self.x = x
        self.y = y

    def __add__(self, other):
        return Vector2D(self.x + other.x, self.y + other.y)

    def __sub__(self, other):
        return Vector2D(self.x - other.x, self.y - other.y)

    def __eq__(self, other):
        return self.x == other.x and self.y == other.y

    def __str__(self):
        return '(%g, %g)' % (self.x, self.y)

u = Vector2D(0,1)
v = Vector2D(1,0)
w = Vector2D(1,1)
a = u + v
print( a)
```

파이썬에서의 변수의 종류

- **지역 변수 – 함수 안에서 선언되는 변수**
- **전역 변수 – 함수 외부에서 선언되는 변수**
- **인스턴스 변수**
  - 클래스 멤버 함수인 메소드 내부에 선언된 변수, 앞에 self. 키워드가 붙는다.
  - private variable
  - public variable
- **정적변수(클래스 변수) ?**
  - 클래스 내부에 선언
  - 같은 클래스로 생성된 객체들은 모두 공통된 하나의 변수 값을 유지

정적 메소드와 클래스 메소드

- **인스턴스 메소드 - 인스턴스(객체)에 속한 메소드**
  - 인스턴스 메소드가 "인스턴스에 속한다"라는 표현은 "인스턴스를 통해 호출가능"라는 뜻
- **정적 메소드와 클래스 메소드는 클래스에 귀속**

To decide whether to use @classmethod or @staticmethod

- **If your method accesses other variables/methods in your class then use @classmethod**.
- On the other hand, if your method does not touches any other parts of the class then use @staticmethod.

When to use what?

- use class method to create factory methods.
  - ✓ Factory methods return class object (similar to a constructor) for different use cases.
- use static methods to create utility functions.

Static Methods

Static Methods:

- Simple functions with no self argument.
- Work on class attributes; not on instance attributes.
- Can be called through both class and instance.
- The built-in function staticmethod() is used to create them.
- A static method is also a method which is bound to the class and not the object of the class.
- A static method can’t access or modify class state.
- It is present in a class because it makes sense for the method to be present in class.
- It’s definition is immutable via inheritance.

Benefits of Static Methods:

- It localizes the function name in the classscope
- It moves the function code closer to where it is used
- More convenient to import versus module-level functions since each method does not have to be specially imported

```
@staticmethod
def some_static_method(*args, **kwds):
   pass
```

Class Methods:

- Functions that have first argument as classname.
- Can be called through both class and instance.
- These are created with classmethod in-built function.

```
@classmethod
def some_class_method(cls, *args, **kwds):
    pass
```

- A class method is a method which is bound to the class and not the object of the class.
- They have the access to the state of the class as it takes a class parameter that points to the class and not the object instance.
- It can modify a class state that would apply across all the instances of the class. For example it can modify a class variable that will be applicable to all the instances.
- Its definition follows Sub class, not Parent class, via inheritance.

정적 메소드

- **정적 메소드**
  - @staticmethod 데코레이터로 수식
  - designed to work on class attributes instead of instance attributes
  - self 키워드 없이 정의
  - 이 메소드가 정의된 클래스로 생성된 모든 객체에 영향
- We can call static method in two ways:
  - ✓call from class
  - ✓call from instances

```
class 클래스이름:
    @staticmethod
    def 메소드이름( 매개변수 ):
        pass
```

@staticmethod데코레이터로 수식합니다.

self 매개변수는 사용하지 않습니다.

정적 메소드

- 예제 : 08/Calculator.py(정적 메소드)

```
class Calculator:

    @staticmethod
    def plus(a, b):
        return a+b

    @staticmethod
    def minus(a, b):
        return a-b

    @staticmethod
    def multiply(a, b):
        return a*b

    @staticmethod
    def divide(a, b):
        return a/b

if __name__ == '__main__':
    print("{0} + {1} = {2}".format(7, 4, Calculator.plus(7, 4)))
    print("{0} - {1} = {2}".format(7, 4, Calculator.minus(7, 4)))
    print("{0} * {1} = {2}".format(7, 4, Calculator.multiply(7, 4)))
    print("{0} / {1} = {2}".format(7, 4, Calculator.divide(7, 4)))
```

- 실행 결과

```
>Calculator.py
7 + 4 = 11
7 + 4 = 3
7 + 4 = 28
7 + 4 = 1.75
```

정적 메소드

- **정적 메소드 호출**

```
# s.py
class S:
    nInstances = 0

    def __init__(self):
        S.nInstances = S.nInstances + 1

        @staticmethod
        def howManyInstances():
            print('Number of instances created: ', S.nInstances)

a = S()
b = S()
c = S()

S.howManyInstances()

a.howManyInstances()
```

```
('Number of instances created: ', 3)
('Number of instances created: ', 3)
```

클래스 메소드

클래스 메소드

- @classmethod 데코레이터로 수식
- cls 매개변수 사용

```
class 클래스이름:
    # ...

    @classmethod
    def 메소드이름(cls):
        pass
```

클래스 메소드를 정의하기 위해서는...
1. @classmethod 데코레이터를 앞에 붙여줍니다.

2. 메소드의 매개변수를 하나 이상 정의합니다.

실습 1

```
>>> class TestClass:
        @classmethod
        def print_TestClass(cls):
            print(cls)


>>> TestClass.print_TestClass()
<class '__main__.TestClass'>
>>> obj = TestClass()
>>> obj.print_TestClass()
<class '__main__.TestClass'>
```

클래스를 통한 클래스 메소드 호출

인스턴스를 통한 클래스 메소드 호출

클래스 메소드

- 예제 : 08/InstanceCounter.py(클래스 메소드)

```
class InstanceCounter:
    count = 0
    def __init__(self):
        InstanceCounter.count += 1

    @classmethod
    def print_instance_count(cls):
        print(cls.count)

if __name__ == '__main__':
    a = InstanceCounter()
    InstanceCounter.print_instance_count()

    b = InstanceCounter()
    InstanceCounter.print_instance_count()

    c = InstanceCounter()
    c.print_instance_count()
```

print_instance_count() 메소드는 InstanceCounter의 클래스 변수인 count를 출력합니다.

- 실행 결과

```
>InstanceCounter.py
1
2
3
```

Instance, static, and class methods

```
class Methods:
   def i_method(self,x):
      print(self,x)

   @staticmethod
   def s_method(x):
      print(x)

   @classmethod
   def c_method(cls,x):
      print(cls,x)

obj = Methods()

obj.i_method(1)
Methods.i_method(obj, 2)
obj.s_method(3)
Methods.s_method(4)
obj.c_method(5)
Methods.c_method(6)
```

```
class Methods:
   def i_method(self,x):
      print(self,x)

   def s_method(x):
      print(x)

   def c_method(cls,x):
      print(cls,x)

   s_method = staticmethod(s_method)
   c_method = classmethod(c_method)

obj = Methods()

obj.i_method(1)
Methods.i_method(obj, 2)
obj.s_method(3)
Methods.s_method(4)
obj.c_method(5)
Methods.c_method(6)
```

```
(<__main__.Methods instance at 0x7f7052d75950>, 1)
(<__main__.Methods instance at 0x7f7052d75950>, 2)
3
4
(<class __main__.Methods at 0x7f7052d6d598>, 5)
(<class __main__.Methods at 0x7f7052d6d598>, 6)
```

Dynamic programming language - Monkey patching

- A way for a program to extend or modify supporting system software locally
- Affecting only the running instance of the program
- The dynamic replacement of attributes at runtime
- Dynamic modifications of a class or module at runtime

```
class Car:
    def drive(self):
        self.speed = 10

myCar = Car()
myCar.color = "blue"
myCar.model = "E-Class"

myCar.drive()                # 객체 안의 drive() 메소드가 호출된다.
print(myCar.speed)           # 10이 출력된다.
```

Dynamic programming language - Monkey patching

```
class Car:
    def drive(self):
        self.speed = 60

myCar = Car()
myCar.speed = 0
myCar.model = "E-Class"
myCar.color = "blue"
myCar.year = "2017"

print("자동차 객체를 생성하였습니다.")
print("자동차의 속도는", myCar.speed)
print("자동차의 색상은", myCar.color)

print("자동차의 모델은", myCar.model)
print("자동차를 주행합니다.")
myCar.drive()
print("자동차의 속도는", myCar.speed)
```

```
자동차 객체를 생성하였습니다.
자동차의 속도는 0
자동차의 색상은 blue
자동차의 모델은 E-Class
자동차를 주행합니다.
자동차의 속도는 60
```

Dynamic programming language - Monkey patching

```
class Car:
    def __init__(self, speed, color, model):
        self.speed = speed
        self.color = color
        self.model = model

    def drive(self):
        self.speed = 60

myCar = Car(0, "blue", "E-class")

print("자동차 객체를 생성하였습니다.")
print("자동차의 속도는", myCar.speed)
print("자동차의 색상은", myCar.color)
print("자동차의 모델은", myCar.model)
print("자동차를 주행합니다.")
myCar.drive()
print("자동차의 속도는", myCar.speed)
```

```
자동차 객체를 생성하였습니다.
자동차의 속도는 0
자동차의 색상은 blue
자동차의 모델은 E-class
자동차를 주행합니다.
자동차의 속도는 60
```

실습:

- Person class 를 정의
  - name, id, age, height, weight
  - __str__(self) 메서드 재정의(overriding)

1. 각 항목에 Person class 의 객체를 갖는 리스트 정의
2. 각각 객체의 필드 name, 에 “을지문덕”, “계백”, “김유신”, “강감찬”, “이순신”을 대입
3. 각각의 age, height, weight 대입
4. 각 인물의 이름과 나이 키 몸무게 순서로 5명의 데이터를 출력

실습:

1. 원을 나타내는 Circle라는 클래스를 설계,

- 반지름 radius와
- 중심 좌표 cx와 cy를 통하여
- 원의 넓이를 반환하는 메소드 area()와
- 원의 중심을 반환하는 메소드 center () 를 구현

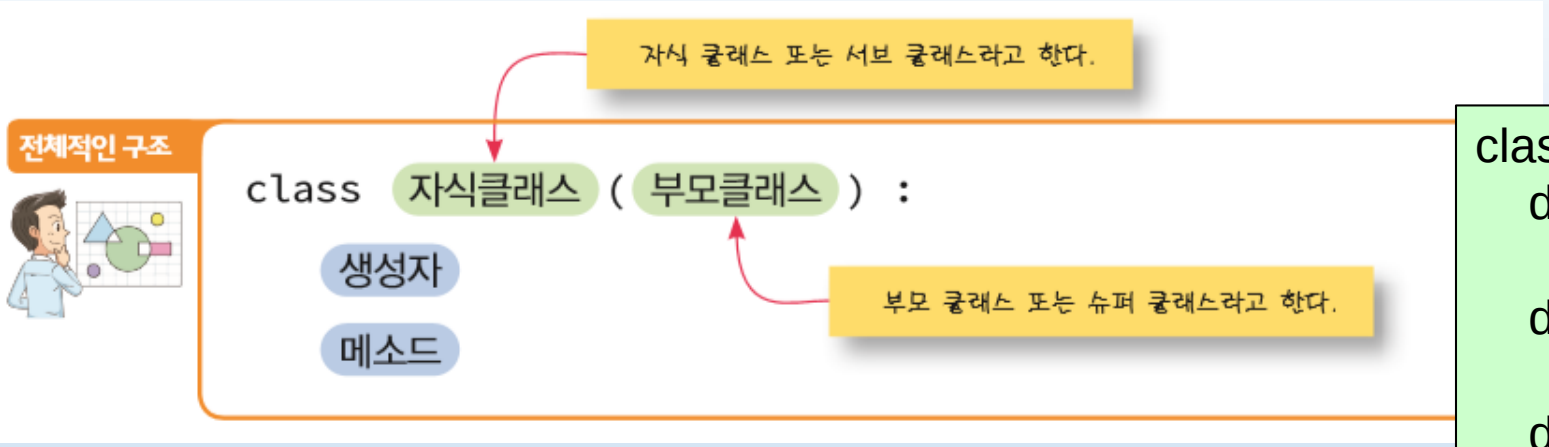
클래스를 완성하고 객체를 생성하여 임의의 값을 대입하여 원의 넓이와 그 중심 좌표값을 화면에 출력하시오.

2. 주사위 클래스 Dice를 설계하는데 메소드에는 roll()을 구현하여 주사위를 던져 나온 숫자를 화면에 출력하는 프로그램을 작성하시오.

-단, 난수의 발생은 다음을 이용하라.

-face = random.randint(1, 6)

상속 구현하기




```
class Car :
    def __init__(self, speed):
        self.speed = speed
    def setSpeed (self, speed):
        self.speed = speed
    def getDesc (self):
        return "차량 = (“ + str(self.speed) + ")"

class SportsCar (Car) :
    def __init__ (self, speed, turbo):
        super().__init__ (speed)
        self.turbo=turbo

    def setTurbo (self, turbo):
        self.turbo=turbo

obj = SportsCar (100, True)
print (obj.getDesc())
obj.setTurbo (False)
```

부모 클래스의 생성자를 명시적으로 호출

- **super(). 사용**

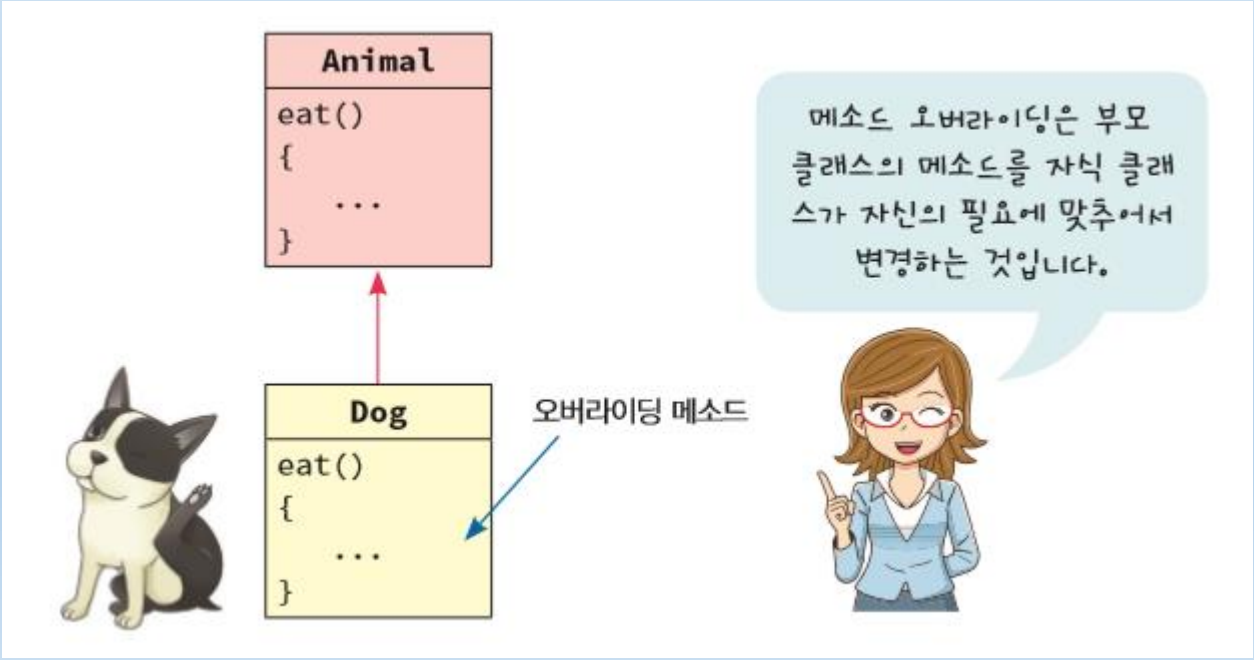
```
class ChildClass(ParentClass) :
    def __init__(self):
        super().__init__()
                                ...
```

- **부모 클래스의 메서드를 자식 클래스에서 호출?**

```
class ParentClass :
   def method1 ():
      pass

class ChildClass(ParentClass) :
    def __init__(self):
        super().__init__()

   def method2 ():
      super(). method1 ()
                                ...
```

메소드 오버라이딩

- **"자식 클래스의 메소드가 부모 클래스의 메소드를 오버라이드 (재정의) 한다"고 말한다.**
  - 명수반 – 메소드명, 매개변수 개수, 반환형
  - 즉, 부모의 클래스에서 사용되던 기능과 자식 클래스에서 사용되는 메소드의 이름은 같지만 기능은 다름
- **Operator overloading – 연산자와 연관된 메소드를 이미 정의 되어 있지만 한번 더 중복해서 정의하여 사용**



예제

```
class Animal:
    def __init__(self, name=""):
        self.name=name

    def eat(self):
        print("동물이 먹고 있습니다. ")

class Dog(Animal):
    def __init__(self):
            super().__init__()

    def eat(self):
        print("강아지가 먹고 있습니다. ")

d = Dog();
d.eat()
```

강아지가 먹고 있습니다.

접근지정자(Access Modifier)

- **Python의 접근 지정자 3 가지**
  - public, protected, private
  - 상속 관계에서 주의할 접근 지정자는 private와 protected
  - Protected: 변수명 접두사로 하나의 underscore, "_", 사용
    - 예, self._protected_variable
  - Private: 변수명 접두사로 두개의 underscore, "__", 사용
    - 예, self.__private_var

- **슈퍼 클래스의 private 멤버**
  - 슈퍼 클래스의 private 멤버는 다른 모든 클래스에 접근 불허
  - 슈퍼 클래스의 private 멤버는 상속되지만 서브 클래스에서 직접 접근 불가
  - 슈퍼 클래스 내부의 변수나 내부 메서드에서만 접근 가능
- **슈퍼 클래스의 protected 멤버**
  - 동일 패키지 내의 모든 클래스에서 접근 가능
  - 상속 관계일 때만 다른 패키지에 있는 클래스의 protected 멤버 접근 가능

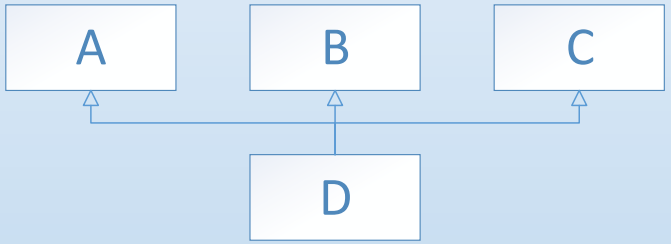
```
class Base :
    def __init__ (self, str):
        self.str = str
        self.__private_var = "PRIVATE VARIABLE IN BASE CLASS"
        self._protected_var="PROTECTED VARIABLE IN BASE CLASS"

    def show_accessModifier (self) :
        print ("str: {}".format(self.str))
        print ("Private: {}".format(self.__private_var))
        print ("Protected: {}".format(self._protected_var))

class Derived (Base) :
    def __init__ (self, str, name) :
        super ().__init__ (str)
        self.name = name
        self.__private_Derived = "PRIVATE VAR IN DERIVED CLASS"
        self._protected_Derived = "PROTECTED VAR IN DERIVED CLASS"

    def print_accessModifier (self) :
        print ("str in Derived: {}".format(self.str))
        ## Wrong!: private var should be used only in the class, not even in the derived class!
        # print ("Private in Derived: {}".format(self.__private_var))
        print ("Protected in Derived: {}".format(self._protected_var))

        print ("name in Derived: {}".format(self.name))
        print ("Private in Derived: {}".format(self.__private_Derived))
        print ("Protected in Derived: {}".format(self._protected_Derived))

if __name__ == "__main__":
    bs = Base ("BASE")
    dv = Derived ("DERIVED", "KPU")

    bs.show_accessModifier ()
    dv.show_accessModifier ()
    dv.print_accessModifier ()
```

다중 상속

- **다중상속은 자식 클래스가 여러 부모 클래스로부터 상속을 받는 것**
  - 파생 클래스의 정의에 기반 클래스의 이름을 콤마(,)로 구분해서 쭉 적어주면 다중상속이 이루어짐.

```
class A:
    pass

class B:
    pass

class C:
    pass

class D(A, B, C):
    pass
```

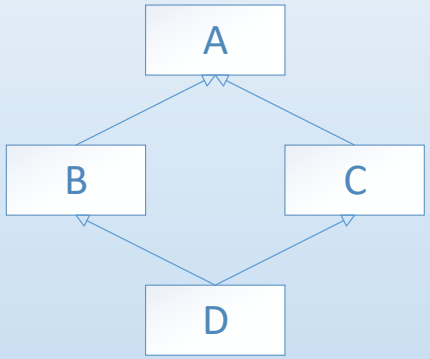
클래스 D는 클래스 A, B, C를 부모로 부터 상속



다중 상속

- **다이아몬드 상속 : 다중 상속이 만들어 내는 곤란한 상황**
  - D는 B와 C 중 누구의 method()를 물려받게 되는 걸까?
  - 부모 클래스 목록에서 맨 앞 클래스의 메서드 상속

```
class A:
    def method(self):
        print("A")

class B(A):
    def method(self):
        print("B")

class C(A):
    def method(self):
        print("C")

class D(B, C):
    pass
```




```
>>> class A:
        def method(self):
            print("A")

>>> class B(A):
        def method(self):
            print("B")

>>> class C(A):
        def method(self):
            print("C")

>>> class D(B, C):
        pass

>>> obj = D()
>>> obj.method()
B
```

D는 B의 method()를 물려받았습니다.

추상 메서드

- 서브 클래스에서 메서드를 오버라이딩 : 슈퍼 클래스에서는 빈 껍질의 메서드만 만들어 놓고 내용은 키워드 **pass** 로 채움

Code12-10.py

```
1  ## 클래스 선언 부분 ##
2  class SuperClass :
3      def method(self) :
4          pass
5
6  class SubClass1 (SuperClass) :
7      def method(self) :          # 메서드 오버라이딩
8          print('SubClass1에서 method()를 오버라이딩함')
9
10 class SubClass2 (SuperClass) :
11     pass
12
13 ## 메인 코드 부분 ##
14 sub1 = SubClass1()
15 sub2 = SubClass2()
16
17 sub1.method()
18 sub2.method()
```

2 ~ 11행 : SuperClass 상속받은 SubClass1 과 SubClass2 만듬
14 ~ 15행 : 각 인스턴스 sub1 과 sub2 생성
17 ~ 18행 : 오버라이딩한 method ( ) 호출

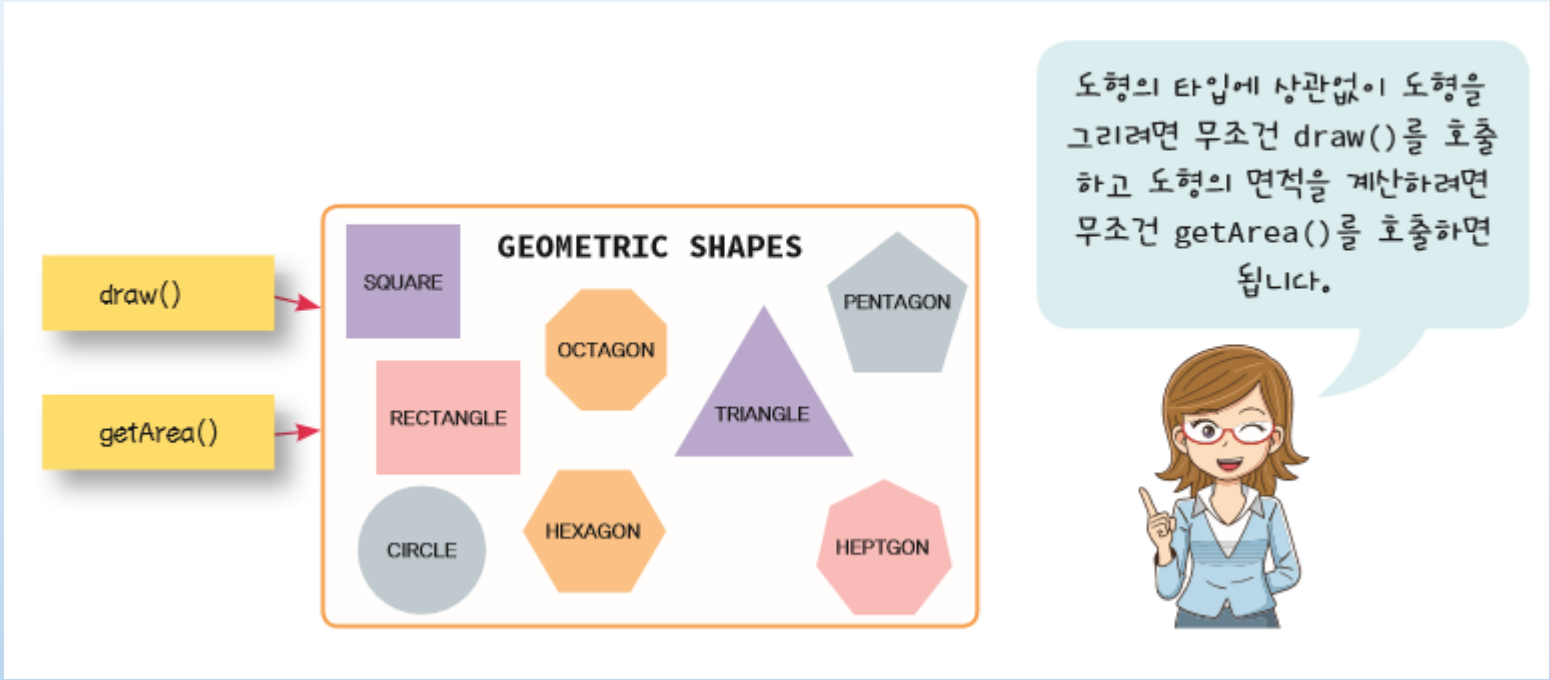
출력 결과

```
SubClass1에서 method()를 오버라이딩함
```

다형성

- 다형성(polymorphism)은 "많은(poly)+모양(morph)"이라는 의미로서 주로 프로그래밍 언어에서 하나의 식별자로 다양한 타입(클래스)을 처리하는 것을 의미한다.

다형성의 예


draw()
getArea()
GEOMETRIC SHAPES
SQUARE
OCTAGON
PENTAGON
RECTANGLE
TRIANGLE
CIRCLE
HEXAGON
HEPTGON
도형의 타입에 상관없이 도형을 그리려면 무조건 draw()를 호출하고 도형의 면적을 계산하려면 무조건 getArea()를 호출하면 됩니다.

내장 함수와 다형성

```
>>> list = [1, 2, 3]                # 리스트
>>> len(list)
3

>>> s = "This is a sentense"        # 문자열
>>> len(s)
18

>>> d = {'aaa': 1, 'bbb': 2}        # 딕셔너리
>>> len(d)
2
```

상속과 다형성

```
class Animal:
    def __init__(self, name):
        self.name = name
    def speak(self):
        return '알 수 없음'

class Dog(Animal):
    def speak(self):
        return '멍멍!'

class Cat(Animal):
    def speak(self):
        return '야옹!'

animalList = [Dog('dog1'),
              Dog('dog2'),
              Cat('cat1')]

for a in animalList:
    print (a.name + ': ' + a.speak())
```

Lab: Vehicle와 Car, Truck

- 일반적인 운송수단을 나타내는 Vehicle 클래스를 상속받아서 Car 클래스와 Truck 클래스를 작성해 보자.

```
truck1: 트럭을 운전합니다.
truck2: 트럭을 운전합니다.
car1: 승용자를 운전합니다.
```

Solution

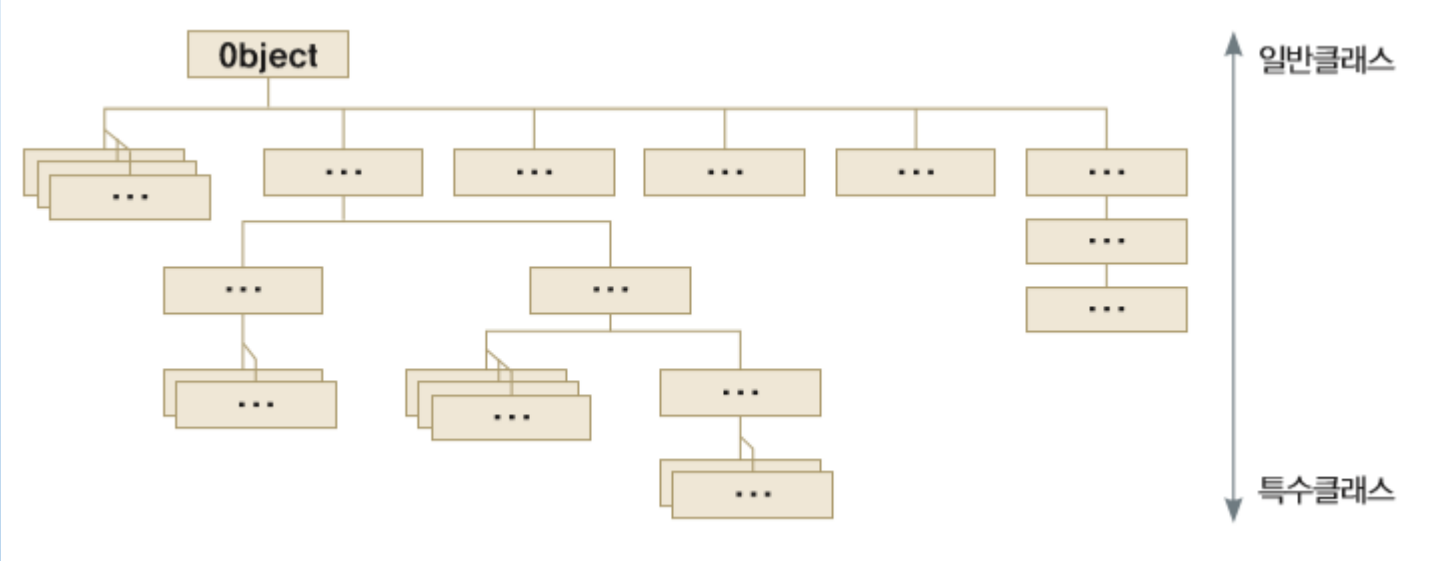
```
class Vehicle:
    def __init__(self, name):
        self.name = name

    def drive(self):
        raise NotImplementedError("이것은 추상메소드입니다. ")

    def stop(self):
        raise NotImplementedError("이것은 추상메소드입니다. ")

class Car(Vehicle):
    def drive(self):
        return '승용자를 운전합니다. '

    def stop(self):
        return '승용자를 정지합니다. '

class Truck(Vehicle):
    def drive(self):
        return '트럭을 운전합니다. '

    def stop(self):
        return '트럭을 정지합니다. '

cars = [Truck('truck1'), Truck('truck2'),  Car('car1')]

for car in cars:
    print( car.name + ': ' + car.drive())
```

공통 메서드와 다형성

```
class wolf(object):
   def bark(self):
      print ("hooooowll")

class dog(object):
   def bark(self):
      print ("woof")

def barkforme(dogtype):
   dogtype.bark()

my_dog = dog()
my_wolf = wolf()

barkforme(my_dog)
barkforme(my_wolf)
```

Object 클래스

- 모든 클래스의 맨 위에는 object 클래스가 있다고 생각하면 된다.



Packages, Modules

모듈이란?

- **모듈(module)**
  함수나 변수들을 모아 놓은 파일

```
## FILE: Module1.py
## 함수 선언부
def func1 () :
    print ("Module1.py의 func1 ()이 호출됨.")

def func2 () :
    print ("Module1.py의 func2 ()이 호출됨.")

def func3 () :
    print ("Module1.py의 func3 ()이 호출됨.")
```

```
## FILE: A.py

import Module1

##
Module1.func1 ()
Module1.func2 ()
Module1.func3 ()
```

출력결과

```
Module1.py의 func1 ()이 호출됨.
Module1.py의 func2 ()이 호출됨.
Module1.py의 func3 ()이 호출됨.
```

- **import문을 실행할 때 파이썬이 모듈 파일을 찾는 순서**
  1) 파이썬 인터프리터 내장(Built-**In**) 모듈
  2) sys.path에 정의되어 있는 디렉토리
- **가져오고자 하는 모듈이 앞에서 출력한 내장 모듈 목록(sys.builtin_module_names)에 없다면, 파이썬은 sys.path에 정의되어 있는 디렉토리에서 모듈 파일을 탐색**
  - 파이썬 모듈이 실행되고 있는 현재 디렉토리
  - PYTHONPATH 환경변수에 정의되어 있는 디렉토리
  - 파이썬과 함께 설치된 기본 라이브러리

모듈 - import에 대하여

- **import 의 역할**
  - "다른 모듈 내의 코드에 대한 접근"을 가능하게 하는 것
  - "다른 코드"에는 변수, 함수, 클래스 등이 모두 포함
- **import문을 사용하는 첫 번째 방법**

```
import 모듈  #모듈의 실제 파일 명은 "모듈.py"
```

- **import문을 사용하는 두 번째 방법**

```
from 모듈 import 변수 또는 함수
```

from 모듈	from 모듈 import 변수 또는 함수
import calculator print(calculator.plus(10, 5)) print(calculator.minus(10, 5)) print(calculator.multiply(10, 5)) print(calculator.divide(10, 5))	from calculator import plus from calculator import minus from calculator import multiply from calculator import divide print(plus(10, 5)) print(minus(10, 5)) print(multiply(10, 5)) print(divide(10, 5))

모듈 - import에 대하여

- **"from 모듈 import 변수 또는 함수"의 4 가지 버전**

1. 예제 : 08/calc_tester2.py

- 사용할 변수나 함수의 이름을 일일이 명기

```
from calculator import plus
from calculator import minus

print(plus(10, 5))
print(minus(10, 5))
#print(multiply(10, 5))
#print(divide(10, 5))
```

calculator 모듈의 plus라는 함수를 호출하여 "calculator." 없이 plus() 이름만으로 함수를 호출할 수 있습니다.

multiply()와 divide()는 import하지 않았습니다. 현재 모듈에서는 보이지 않는 함수입니다. 호출 불가!

2. 예제 : 08/calc_tester3.py

- 콤마(,)를 이용해서 여러 함수(또는 변수)의 이름을 한 줄에 기입

```
from calculator import plus, minus

print(plus(10, 5))
print(minus(10, 5))
#print(multiply(10, 5))
#print(divide(10, 5))
```

```
from calculator import plus
from calculator import minus
```
와 동일한 코드입니다.

모듈 - import에 대하여

3. 예제 : 08/calc_tester4.py

- 와일드카드 *를 이용

```
from calculator import *

print(plus(10, 5))
print(minus(10, 5))
print(multiply(10, 5))
print(divide(10, 5))
```

- **그러나 import *와 같은 코드는 지양할 것을 권장**
  - 코드가 복잡해지고 모듈의 수가 많아지면 어떤 모듈 또는 어떤 변수, 함수를 불러오고 있는지 파악하기 힘들어짐. 코드 가독성을 떨어뜨림

4. 예제 : 08/calc_tester5.py (import 모듈 as 새이름)

```
import calculator as c

print(c.plus(10, 5))
print(c.minus(10, 5))
print(c.multiply(10, 5))
print(c.divide(10, 5))
```

calculator 모듈을 c라는 이름으로 불러옵니다.

calculator라는 이름 대신 c를 이용하여 함수 이름에 접근합니다.

모듈 작성

```
# FILE: fibo.py
# 피보나치 수열 모듈

def fib(n):    # 피보나치 수열을 화면에 출력한다.
    a, b = 0, 1
    while b < n:
        print(b, end=' ')
        a, b = b, a+b
    print()
```

모듈 사용

- **모듈이름.함수()의 꼴로 모듈의 함수를 호출**

```
>>> import fibo

>>> fibo.fib(1000)
1 1 2 3 5 8 13 21 34 55 89 144 233 377 610 987

>>> fibo.__name__
'fibo'
```

```
>>> from fibo import *
>>> fib(500)
1 1 2 3 5 8 13 21 34 55 89 144 233 377
```

모듈을 스크립트로 실행하기

- 만약 파이썬 모듈을 다음과 같이 명령어 프롬프트를 이용하여 실행한다면

```
# FILE: fibo.py
# 피보나치 수열 모듈

def fib(n):    # 피보나치 수열을 화면에 출력한다.
    a, b = 0, 1
    while b < n:
        print(b, end=' ')
        a, b = b, a+b
    print()

if __name__ == "__main__":
    import sys
    fib(int(sys.argv[1]))
```

```
C> python fibo.py <arguments>
```

예

```
C> python fibo.py 50
1 1 2 3 5 8 13 21 34
```

```
if __name__ == "__main__":
    import sys
    fib(int(sys.argv[1]))
```

모듈-메인 모듈과 하위 모듈

- **예제 : 08/main_sub/sub.py**

```
print("beginning of sub.py...")
print('name : {0}'.format(__name__))
print("end of sub.py...")
```

- **예제 : 08/main_sub/main.py**

```
import sub

print("beginning of main.py...")
print('name : {0}'.format(__name__))
print("end of main.py...")
```

  - 실행 결과

```
>>>main.py
beginning of sub.py...
name : sub
end of sub.py...
beginning of main.py...
name : __main__
end of main.py...
```

모듈-메인 모듈과 하위 모듈

- **예제 : 08/main_sub2/sub.py**

```
if __name__ == '__main__':
    print("beginning of sub.py...")
    print('name : {0}'.format(__name__))
    print("end of sub.py...")
```

- **예제 : 08/main_sub2/main.py**

```
import sub

print("beginning of main.py...")
print('name : {0}'.format(__name__))
print("end of main.py...")
```

- 실행 결과

```
>>>main.py
beginning of main.py...
name : __main__
end of main.py...

>>>sub.py
beginning of sub.py...
name : __main__
end of sub.py...
```

sub.py의 출력문들이 실행되지 않았습니다.

최상위 수준으로 실행하면 sub 모듈의 __name__ 변수는 '__main__'이 되어 출력문들을 실행합니다.

패키지

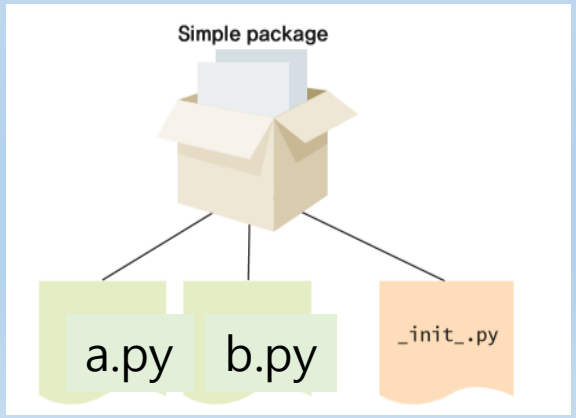
패키지

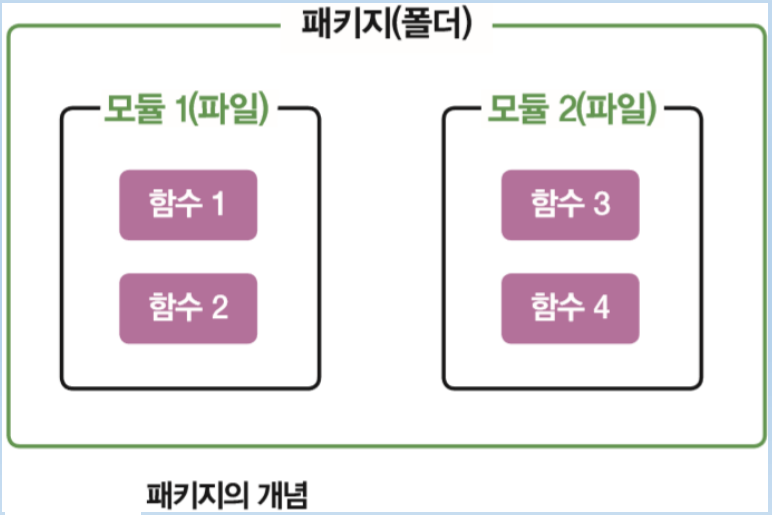
- 모듈을 모아놓는 디렉토리
- 디렉토리가 "파이썬의 패키지"로 인정받으려면 __init__.py 파일이 그 경로에 존재해야 함
  - 버전 3.3 이후 이 파일이 없어도 됨
- 패키지 함수 도입 형식

from 패키지 import *

from 패키지 import 모듈

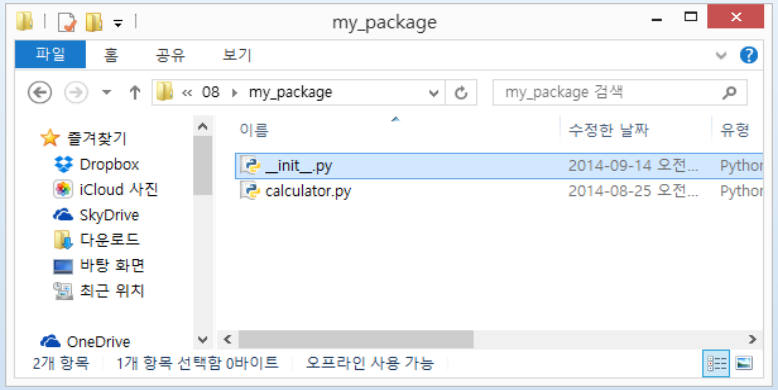
from 패키지.모듈 import 함수

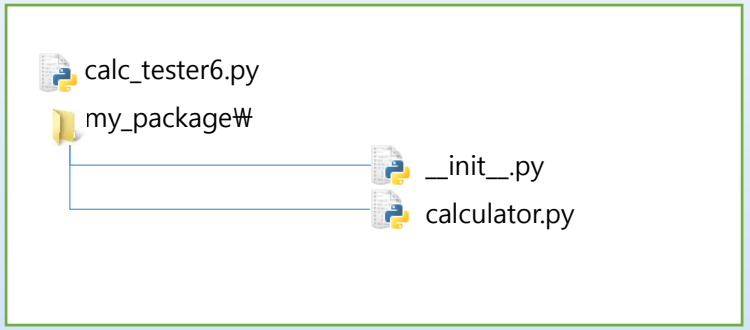




패키지의 개념

실습 (패키지에서 모듈 반입하기)





- 08/calc_tester6.py

```
from my_package import calculator

print(calculator.plus(10, 5))
print(calculator.minus(10, 5))
print(calculator.multiply(10, 5))
print(calculator.divide(10, 5))
```

"from 패키지 import 모듈"의 꼴로 모듈을 불러옵니다.

- 실행 결과

```
>>>calc_tester6.py
15
5
50
2.0
```

패키지 - __init__.py에 대하여

- **보통의 경우, init__.py 파일은 대개 비워둠**
- **이 파일을 손대는 경우는 __all__이라는 변수를 조정할 때 정도**
  - __all__은 다음과 같은 코드를 실행할 때 패키지로부터 반입할 모듈의 목록을 정의하기 위해 사용

```
from 패키지 import *
```

❖ **import *은 사용을 자제하는 것이 좋음.**

패키지 - __init__.py에 대하여

- 실습 (__all__ 변수 조정하기)
- 08/luv_song/eeny.py

```
def test():
    print('module name : {0}'.format(__name__))
```

- 08/luv_song/meeny.py

```
def test():
    print('module name : {0}'.format(__name
```

- 08/luv_song/miny.py

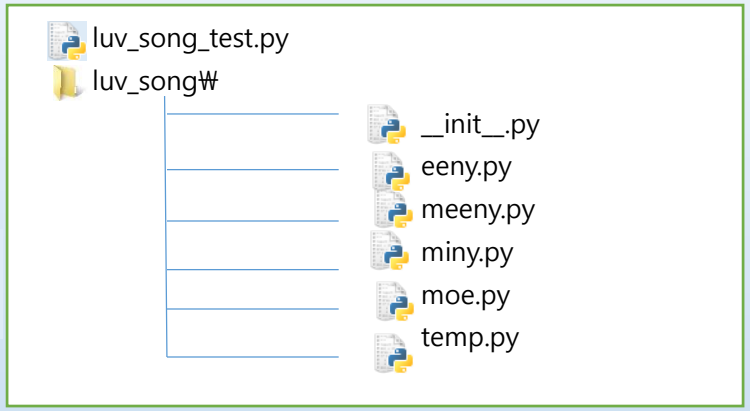
```
def test():
    print('module name : {0}'.format(__name__))
```

- 08/luv_song/moe.py

```
def test():
    print('module name : {0}'.format(__name__))
```

- 08/luv_song/__init__.py

```
__all__ = ['eeny', 'meeny', 'miny', 'moe']
```



❖ 08/luv_song_test.py

```
from luv_song import *

eeny.test()
meeny.test()
miny.test()
moe.test()
```

- 실행 결과

```
>luv_song_test.py
module name : luv_song.eeny
module name : luv_song.meeny
module name : luv_song.miny
module name : luv_song.moe
```

패키지 - site-packages에 대하여

- **site-packages**
  - 파이썬의 기본 라이브러리 패키지 외에 추가적인 패키지를 설치하는 디렉토리
  - 각종 서드 파티 모듈을 바로 이 곳에 설치함

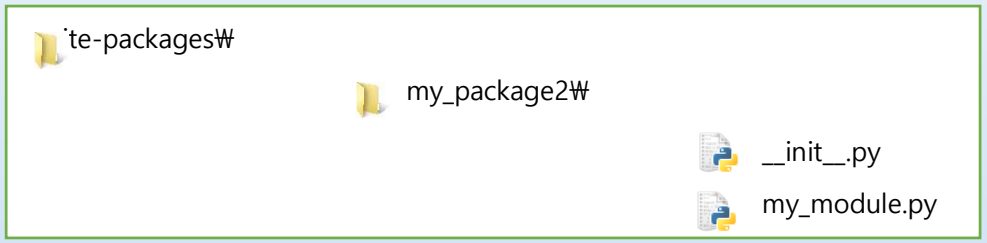
- **실습 1 (site-packages 확인)**

```
>>> import sys
>>> sys.path
['', 'C:₩₩Python34₩₩Lib₩₩idlelib',
'C:₩₩WINDOWS₩₩SYSTEM32₩₩python34.zip', 'C:₩₩Python34₩₩DLLs',
'C:₩₩Python34₩₩lib', 'C:₩₩Python34',
'C:₩₩Python34₩₩lib₩₩site-packages']
```

site-package는 파이썬이 기본적으로 모듈을 탐색하는 경로에 포함되어 있습니다.

패키지 - site-packages에 대하여

- **실습 2**
  - sys.path에 있던 site-package 디렉토리에 다음과 같이 my_package2 디렉토리를 만들고, 그 안에 __init__.py와 my_module.py 를 생성



  - …₩site-packages₩my_package2₩my_module.py

```
def info():
    print(__name__)
    print(__file__)
```

  - 파이썬 쉘을 열고 다음 코드를 입력

```
>>> from my_package2 import my_module
>>> my_module.info()
my_package2.my_module
C:₩Python34₩lib₩site-packages₩my_package2₩my_module.py
```

Synchronization Between Threads

멀티쓰레딩

```
### 메소드에 의한 구현
import time
import threading

class RacingCar:
    carName=""

    def __init__ (self, name):
        self.carName = name

    def runCar (self) :
        while True : ### for i in range (0, 3):
            carStr = self.carName + ' ~~ run₩n'
            print (carStr, end='')
            time.sleep (0.1)

if __name__ == "__main__" :
    car1 = RacingCar ("Mustang")
    car2 = RacingCar ("Taurus")
    car3 = RacingCar ("Tesla")

    thd1 = threading.Thread (target=car1.runCar)
    thd2 = threading.Thread (target=car2.runCar)
    thd3 = threading.Thread (target=car3.runCar)

    thd1.start ()
    thd2.start ()
    thd3.start ()
```

```
### 객체에 의한 구현
import time
import threading

class RacingCar (threading.Thread):
    carName=""

    def __init__ (self, name):
        threading.Thread.__init__(self)
        self.carName = name

    def run (self) :
        while True : ### for i in range (0, 3):
            carStr = self.carName + ' ~~ run₩n'
            print (carStr, end='')
            time.sleep (0.1)

if __name__ == "__main__" :
    condition = threading.Condition()

    car1 = RacingCar ("Mustang")
    car2 = RacingCar ("Taurus")
    car3 = RacingCar ("Tesla")

    car1.start ()
    car2.start ()
    car3.start ()
    car1.join ()
    car2.join ()
    car3.join ()
```

실습:

- **다음 코드를 사용하여 아래의 결과를 무한히 반복하도록 수정 하시오.**

  thread1 은 "한국산업기술대학교" 출력

  thread2 는 "컴퓨터공학부" 출력

  thread3 은 "홍길동" 출력

Producer – Consumer Problem

```
self.condition.acquire()

 ## Critical section

# self.condition.notify() # notify to a consumer
 self.condition.notifyAll () # notify to all consumers

 self.condition.release()
```

```
self.condition.acquire()

while True:
    self.condition.wait()

     # Critical section

self.condition.release()
```

Producer – Consumer Problem

```
import threading
import random
import time

class Producer(threading.Thread):
    def __init__(self, integers, condition):
        threading.Thread.__init__(self)
        self.integers = integers
        self.condition = condition

    def run(self):
        while True:
            integer = random.randint(0, 256)

            self.condition.acquire()

            print ('condition acquired by %s' % self.name)
            self.integers.append(integer)
            print ('%d appended to list by %s' % (integer, self.name))
            print ('condition notified by %s' % self.name)

            # self.condition.notify() # notify to a consumer
            self.condition.notifyAll () # notify to all consumers

            print ('condition released by %s' % self.name)

            self.condition.release()
            time.sleep(1)
```

```
import threading

class Consumer(threading.Thread):
    def __init__(self, integers, condition):
        threading.Thread.__init__(self)
        self.integers = integers
        self.condition = condition

    def run(self):
        while True:
            self.condition.acquire()

            print ('condition acquired by %s' % self.name)
            while True:
                if self.integers:
                    integer = self.integers.pop()
                    print ('%d popped from list by %s' % (integer, self.name))
                    break
                print ('condition wait by %s' % self.name)

                self.condition.wait()

            print ('condition released by %s' % self.name)

            self.condition.release()
```

Producer – Consumer Problem

```
import threading
from prod import *
from cons import *

def main():
    integers = []
    condition = threading.Condition()
    t1 = Producer(integers, condition)
    t2 = Consumer(integers, condition)
    t1.start()
    t2.start()
    t1.join()
    t2.join()

if __name__ == '__main__':
    main()
```

Synchronization by Condition variables

```
import threading
import time
import random

def main():
    cond = threading.Condition()
    turn = CondVar()

    t1 = School(cond, turn)
    t2 = Dept(cond, turn)
    t3 = Name(cond, turn)

    t1.start()
    t2.start()
    t3.start()
    t1.join()
    t2.join()
    t3.join()

class CondVar :
    myTurn = 0
    def __init__ (self):
        pass
```

Synchronization by Condition variables

```
class School (threading.Thread):
    def __init__(self, condition, turn):
        threading.Thread.__init__(self)
        self.condition = condition
        self.turn = turn

    def run(self):
        while True:
            self.condition.acquire() ### mutex_lock
            while self.turn.myTurn != 0: self.condition.wait()

            #print ('condition acquired by %s' % self.name)
            print ("Sanki University")

            self.turn.myTurn = 1

            # self.condition.notify() # notify to a consumer
            self.condition.notifyAll () # notify to all consumers
            #print ('condition notified by %s' % self.name)

            self.condition.release() ### mutex_unlock
            #print ('condition released by %s' % self.name)
            time.sleep(1)
```

Synchronization by Condition variables

```
class Dept (threading.Thread):
    def __init__(self, condition, turn):
        threading.Thread.__init__(self)
        self.condition = condition
        self.turn = turn

    def run(self):
        while True:
            self.condition.acquire() ### mutex_lock
            while self.turn.myTurn != 1: self.condition.wait()

            #print ('condition acquired by %s' % self.name)
            print ("Dept of Computer Engineering")

            self.turn.myTurn = 2

            # self.condition.notify() # notify to a consumer
            self.condition.notifyAll () # notify to all consumers
            #print ('condition notified by %s' % self.name)

            self.condition.release() ### mutex_unlock
            #print ('condition released by %s' % self.name)
            time.sleep(1)
```

Synchronization by Condition variables

```
class Name (threading.Thread):
    def __init__(self, condition, turn):
        threading.Thread.__init__(self)
        self.condition = condition
        self.turn = turn

    def run(self):
        while True:
            self.condition.acquire() ### mutex_lock
            while self.turn.myTurn != 2: self.condition.wait()

            print ("Noiri Lab")
            print ()

            self.turn.myTurn = 0

            # self.condition.notify() # notify to a consumer
            self.condition.notifyAll () # notify to all consumers
            #print ('condition notified by %s' % self.name)


            self.condition.release() ### mutex_unlock
            #print ('condition released by %s' % self.name)
            time.sleep(1)

if __name__ == '__main__':
    main()
```

Socket Programming

TCP timestamp server

```
from socket import *
from time import ctime

HOST = 'localhost'
PORT = 28812
BUFSIZE = 1024
ADDR = (HOST, PORT)

tcpTimeSrvrSock = socket (AF_INET,SOCK_STREAM)
tcpTimeSrvrSock.bind (ADDR)
tcpTimeSrvrSock.listen (50)

while True:
  print 'waiting for connection...'
  tcpTimeClientSock, addr = tcpTimeSrvrSock.accept ()
  print '...connected from:', addr

  while True:
    data = tcpTimeClientSock.recv (BUFSIZE)
    if not data:
      break
    tcpTimeClientSock.send ('[%s] %s' % (ctime(), data))

  tcpTimeClientSock.close ()

tcpTimeSrvrSock.close ()
```

TCP timestamp client

```
from socket import *

HOST = 'localhost'
PORT = 28812
BUFSIZE = 1024
ADDR = (HOST, PORT)

tcpTimeClientSock = socket (AF_INET, SOCK_STREAM)
tcpTimeClientSock.connect (ADDR)

while True:
  data = raw_input ('> ')
  if not data:
      break
  tcpTimeClientSock.send (data)
  data = tcpTimeClientSock.recv (BUFSIZE)
  if not data:
      break
print data

tcpTimeClientSock.close ()
```

실습:

- **TCP/IP 프로토콜을 사용하여 실시간 채팅 프로그램을 작성하시오.**
  - ✓Client-server 기반

Client – Server Model: UCP/IP

- **멀티미디어 데이터 전송**
- **데이터의 손실보다 시간적 가치가 중요한 데이터**
  - 실시간 방송용 동영상
  - 실시간 방송용 음성
- **다중 전송 가능**
  - TCP 방식은 기본적으로 불가

UDP - server

```
import random
import socket

server_socket = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
server_socket.bind(('', 12000))

while True:
    rand = random.randint(0, 10)
    message, address = server_socket.recvfrom(1024)
    message = message.upper()
    if rand >= 4:
        server_socket.sendto(message, address)
```

UDP - client

```
import time
import socket

for pings in range(10):
    client_socket = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
    client_socket.settimeout(1.0)
    message = b'test'
    addr = ("127.0.0.1", 12000)

    start = time.time()
    client_socket.sendto(message, addr)
    try:
        data, server = client_socket.recvfrom(1024)
        end = time.time()
        elapsed = end - start
        print ("{} {} {}".format(data, pings, elapsed))
    except socket.timeout:
        print('REQUEST TIMED OUT')
```

실습:

- **UDP/IP 프로토콜을 사용하여 실시간 채팅 프로그램을 작성하시오.**
  - ✓Client-server 기반
  - ✓다중 참여자 기반