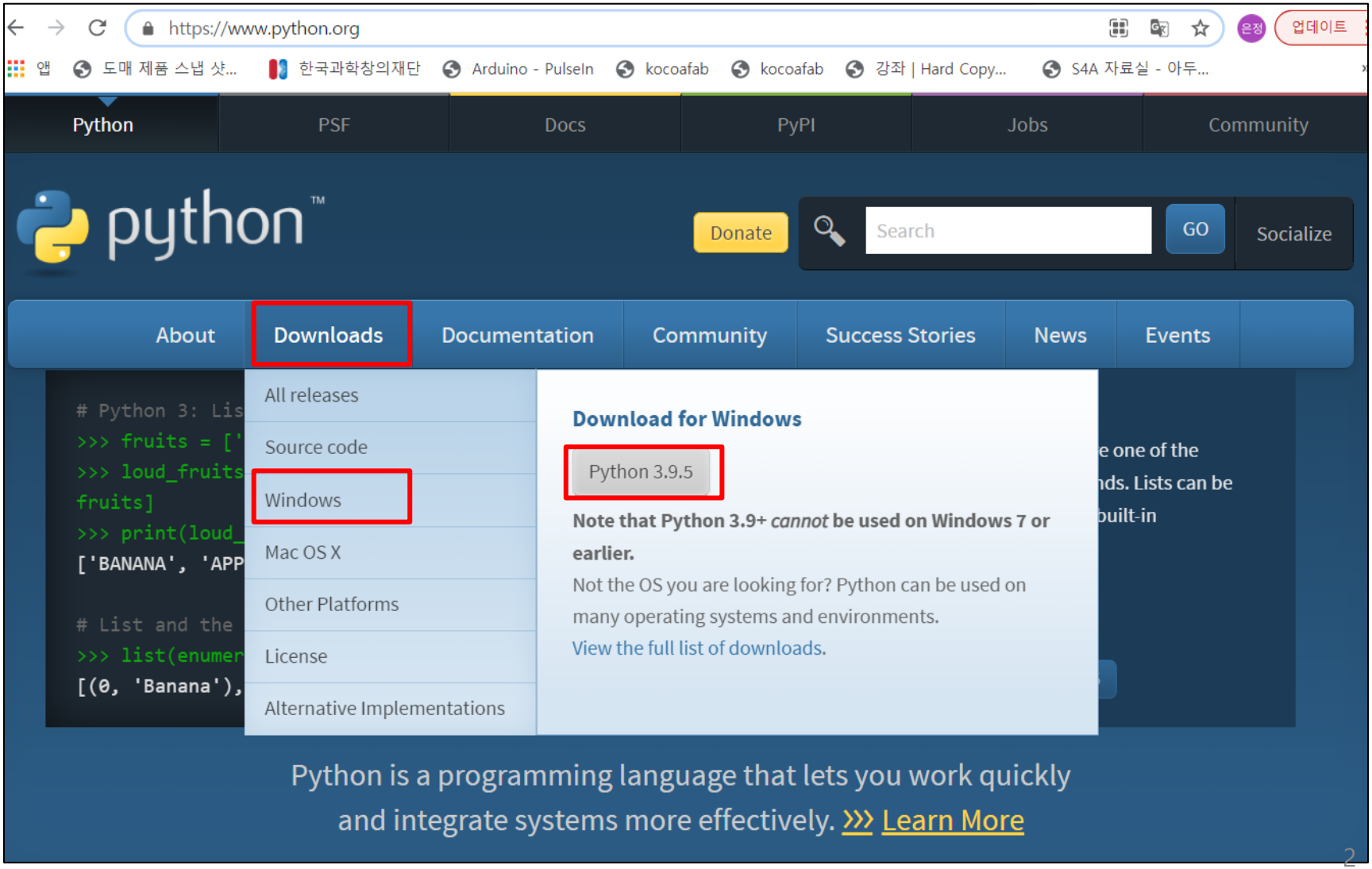
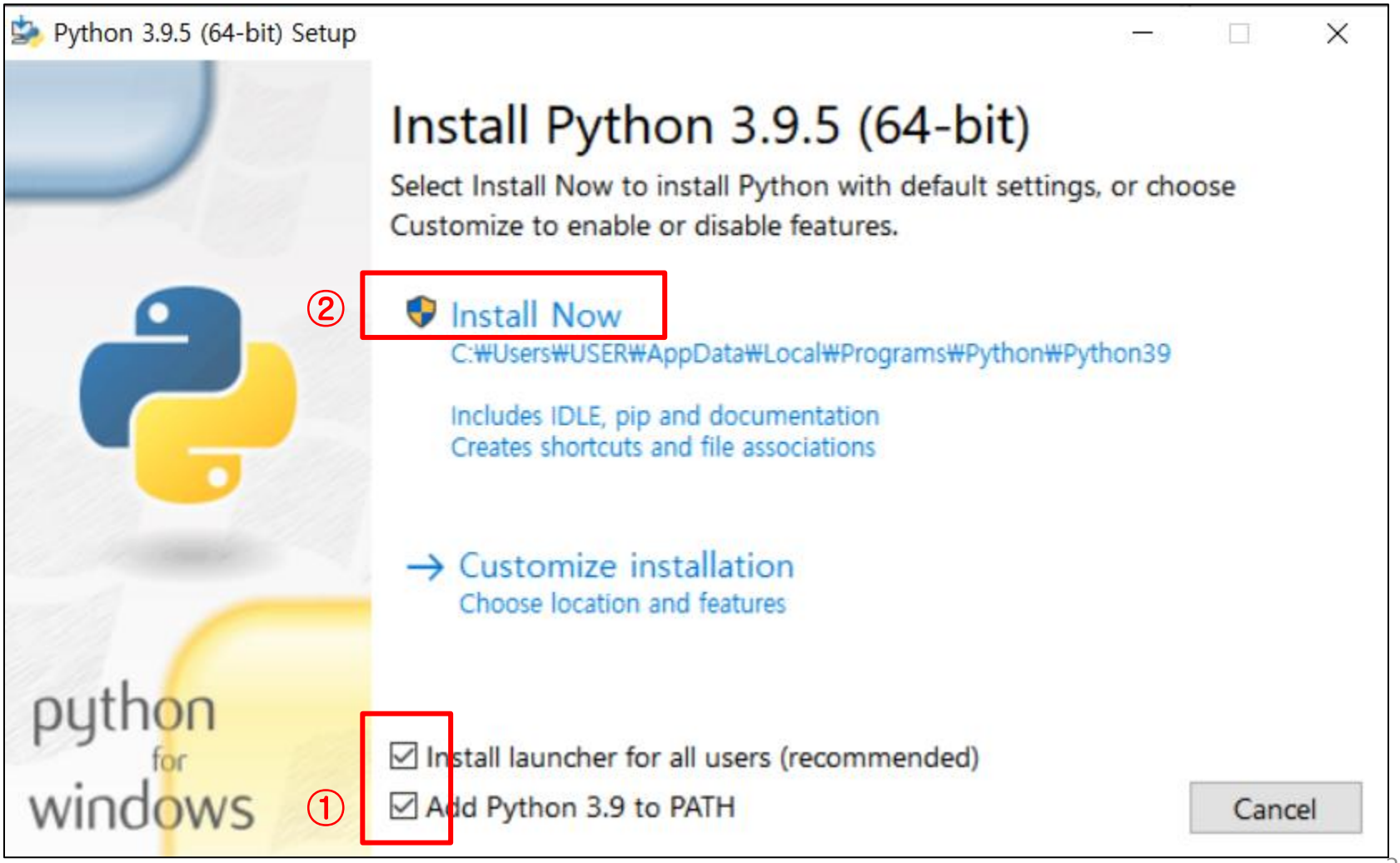
01. 파이썬 설치

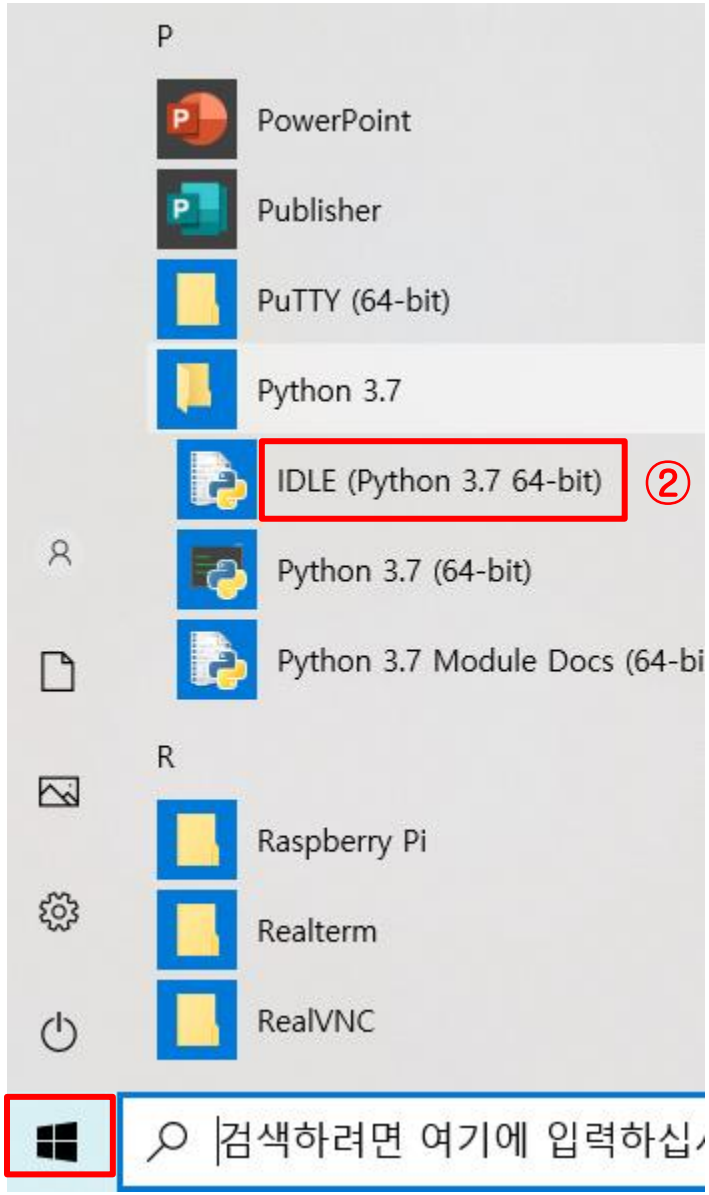
https://www.python.org/



01. 파이썬 설치


Python 3.9.5 (64-bit) Setup
Install Python 3.9.5 (64-bit)
Select Install Now to install Python with default settings, or choose
Customize to enable or disable features.
②
Install Now
C:₩Users₩USER₩AppData₩Local₩Programs₩Python₩Python39
Includes IDLE, pip and documentation
Creates shortcuts and file associations
Customize installation
Choose location and features
python
for
windows
Install launcher for all users (recommended)
①
Add Python 3.9 to PATH
Cancel

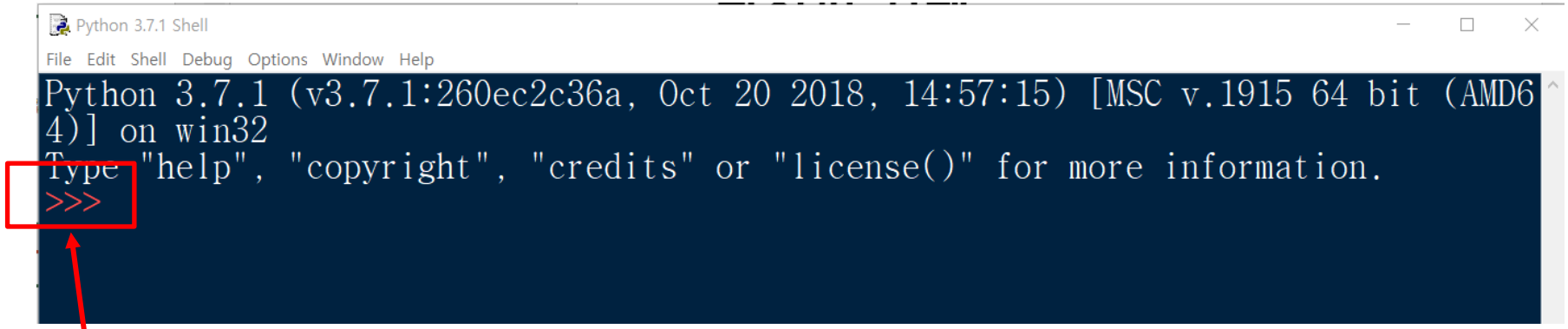
01. 파이썬 실행

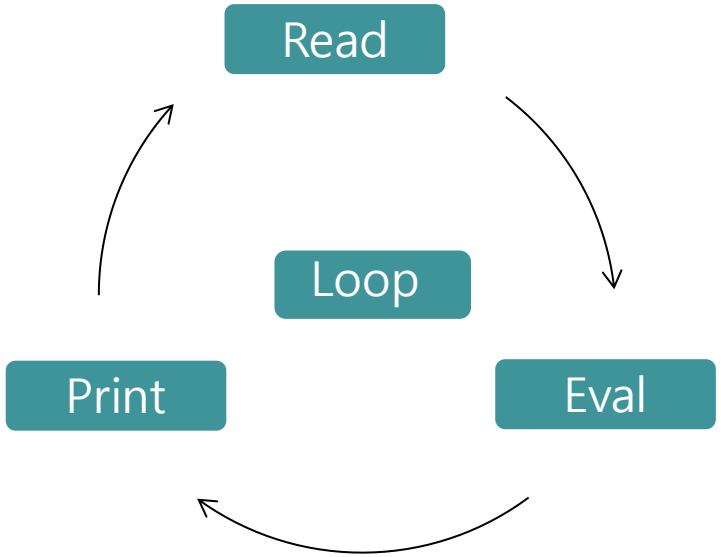


IDLE(Integrated Development Environment)
- 통합 개발 환경

01. 파이썬 실행

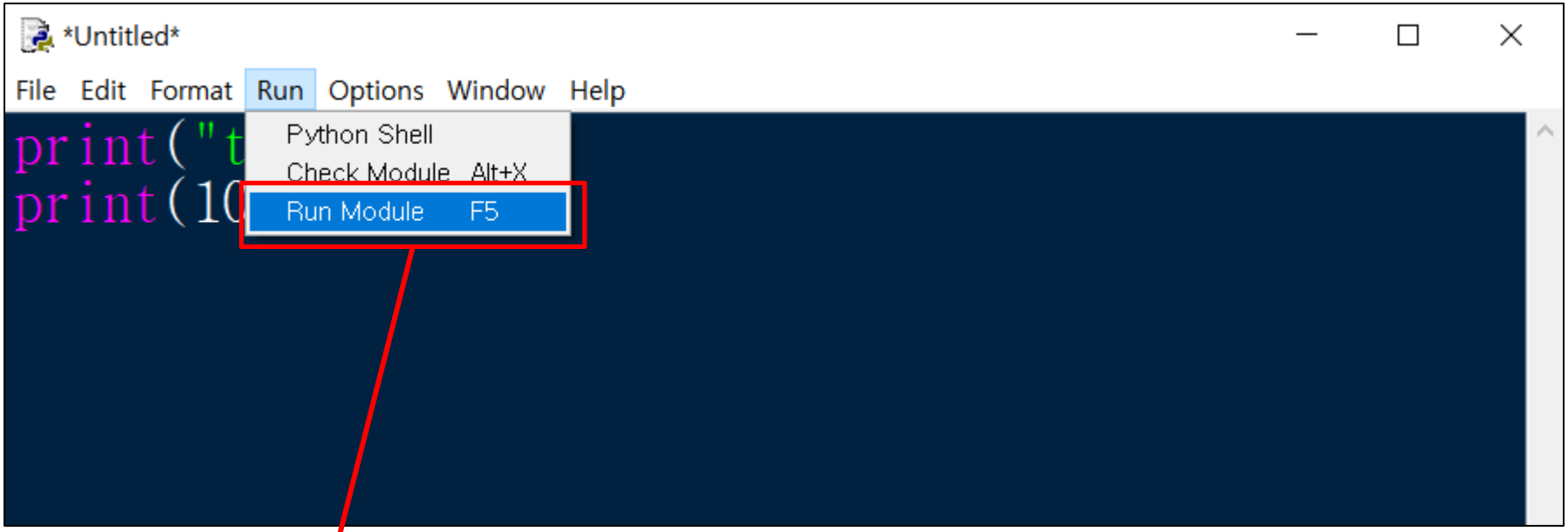
- 파이썬 쉘(python shell) – 대화형 쉘(interactive shell),
  인터랙티드 모드(interactive mode) 라고도 함





01. 파이썬 실행

➢ 스크립트 모드 – 편집, 파일로 저장 가능



실행결과

```
=== RESTART: C:/Users/USER/AppData/Local/Programs/Python/Python37/test7.py =
test
20
>>>
```

02. 식별자 기본 규칙

- 식별자 – 이름을 붙일 때 사용하는 단어

  변수 이름, 함수 이름, 클래스 이름

- 변수 이름 규칙
  - 영문자와 숫자, 밑줄 문자(_)로 이루어 진다
  - 중간에 공백이 들어가면 안된다
  - 첫 글자는 반드시 영문자 또는 밑줄 문자(_)
  - 예약어(키워드)는 사용 불가 – if, elif, while, for…
  - 대문자와 소문자는 구별된다 - sum ≠ SUM
  - 내장함수는 사용하지 않는 것이 좋다 – print, sum, abs …
  - 변수의 역할을 가장 잘 설명하는 이름을 지어야 한다

02. 식별자 기본 규칙

- 키워드 – 특별한 의미가 부여된 단어

False	None	True	and	as	assert
break	class	continue	def	del	elif
else	except	finally	for	from	global
if	import	in	is	lambda	nonlocal
not	or	pass	raise	return	try
while	with	yield			

```
>>> import keyword
>>> print(keyword.kwlist)
```

02. 식별자 기본 규칙

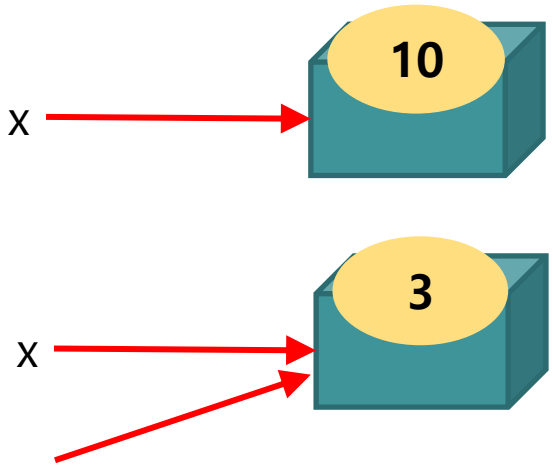
- 스네이크 케이스(snake_case) : 단어 사이에 _ 기호를 붙여 식별자 만든다
- 캐멀 케이스(CamelCase) : 단어들의 첫 글자를 대문자로 만들어 식별자를 만든다
  단어들의 첫 글자는 소문자로, 나머지 단어의 첫 글자는 대문자로 적는 방법(myNewCar)

스네이크 케이스	캐멀 케이스
item_list	ItemList
login_status	LoginStatus
character_hp	CharacterHp
rotate_angle	RotateAngle

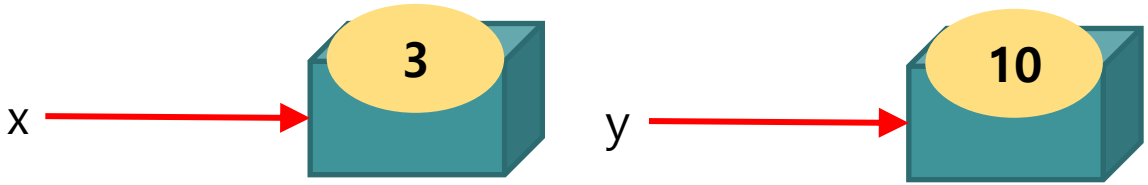
03. 자료형

- 파이썬에서는 모든 것이 객체로 되어있다
- 변수에 저장되는 것은 실제 값이 아니고 객체의 참조 값(주소)이다

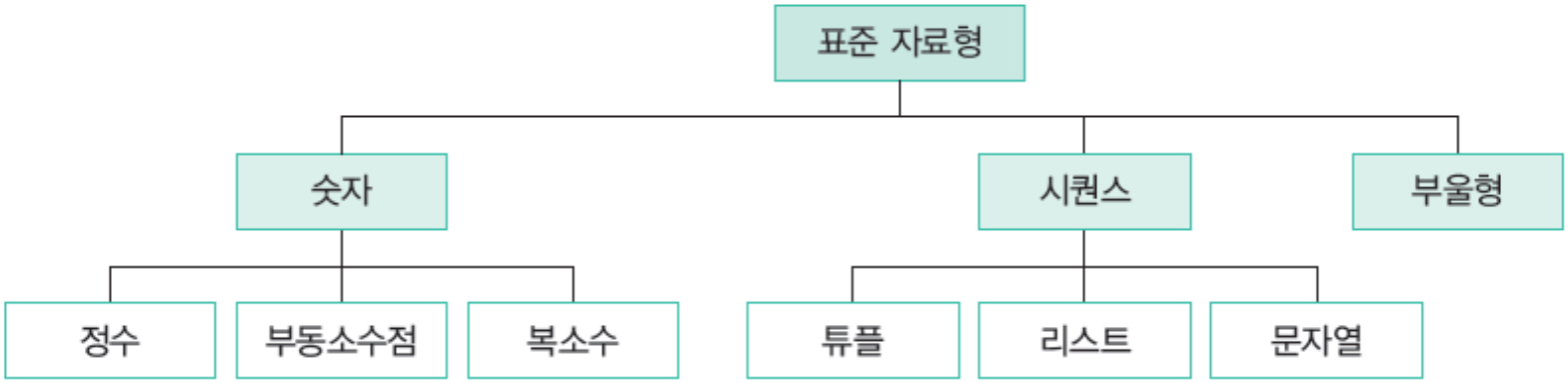
```
>>> x = 10
>>> id(x)

>>> x = 3
>>> y = x

>>> id(x)
>>> id(y)
```




```
>>> y = 10
>>> id(y)
```



03. 자료형



자료형	예
정수(int)	..., −2, −1, 0, 1, 2, ...
부동소수점수(float)	3.2, 3.14, 0.12
문자열(str)	'Hello World!', "123"

```
>>> type(10)
>>> type(True)
>>> type(12.30)
>>> type("hello")
```

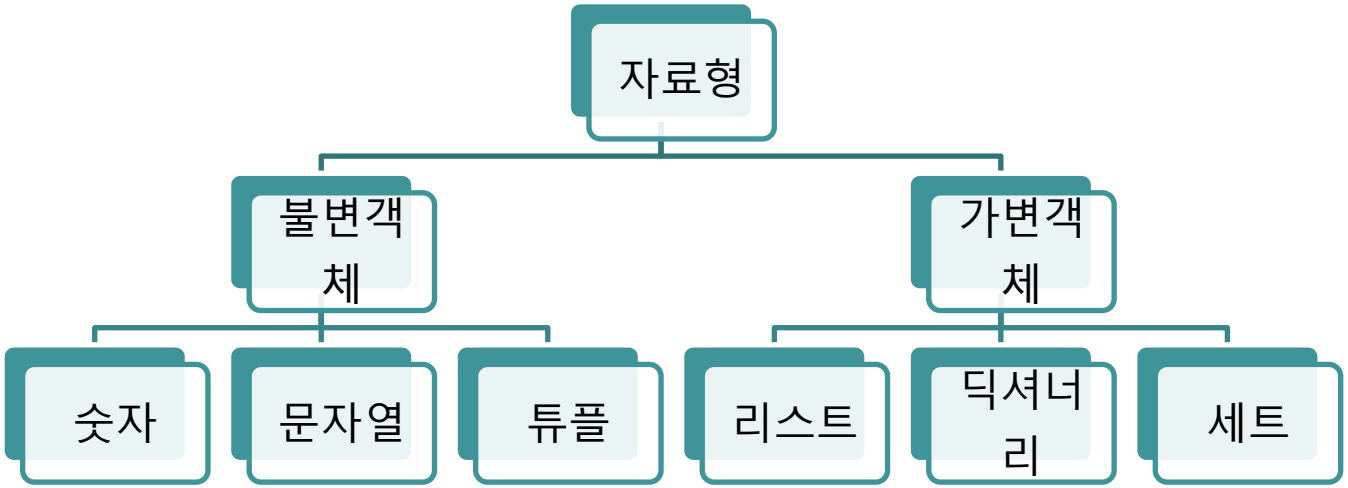
03. 자료형

- 불변 객체 : 한번 만들어지면 변경할 수 없는 객체

  정수, 실수, 문자열, 튜플

- 가변 객체 : 직접 변경할 수 있는 객체

  리스트, 딕셔너리, 세트, ...

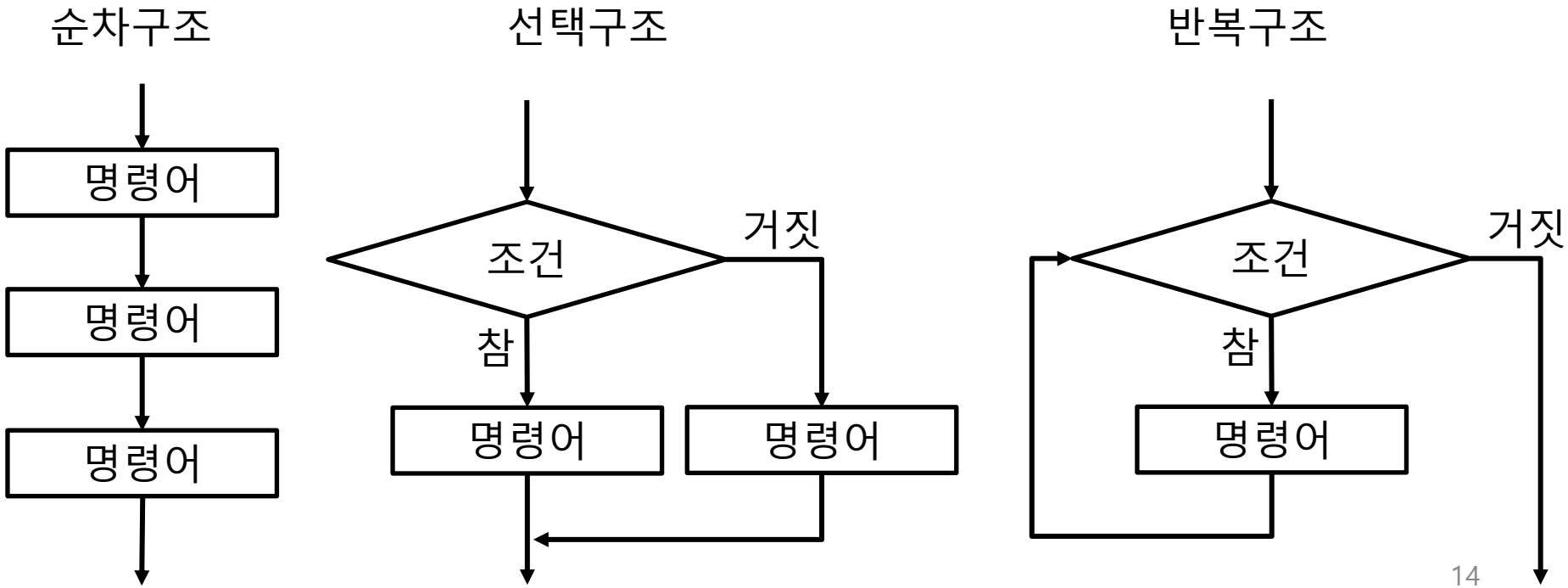


03. 자료형

	리스트	튜플	사전
작성 방법	[ ] 대괄호	( ) 소괄호	{ } 중괄호
데이터 구조	시퀀스(나열)	시퀀스(나열)	사전
접근 방법	변수[번호]	변수[번호]	변수[키]
특징	뮤터블	이뮤터블	번호는 없음

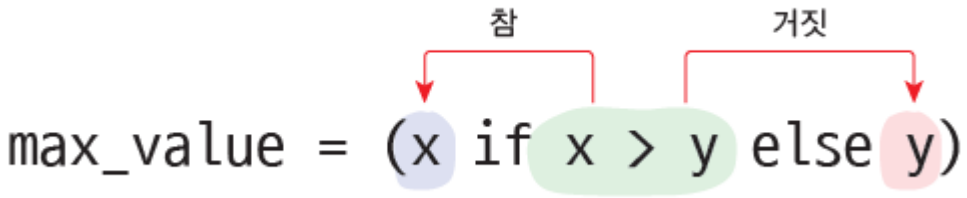
04. 제어문

- 3가지 제어 구조
  - 순차구조(sequence) – 명령들이 순차적으로 실행되는 구조
  - 선택구조(selection) – 둘 중의 하나의 명령을 선택하여 실행되는 구조
  - 반복구조(iteration) – 동일한 명령이 반복되면서 실행되는 구조



04. 제어문

- 삼항 연산자 (조건 연산자)




```
>>> a=[10, 5, 30, 9, 253, 1]
>>> max = min = a[0]
>>> for x in a:
        max = x if x > max else max
        min = x if x < min else min
>>> max
>>> min
```

04. 제어문

- 반복의 종류
  - 횟수 반복(for 문): 정해진 횟수만큼 반복한다.
  - 조건 반복(while 문): 특정한 조건이 성립되는 동안 반복한다.

횟수 반복

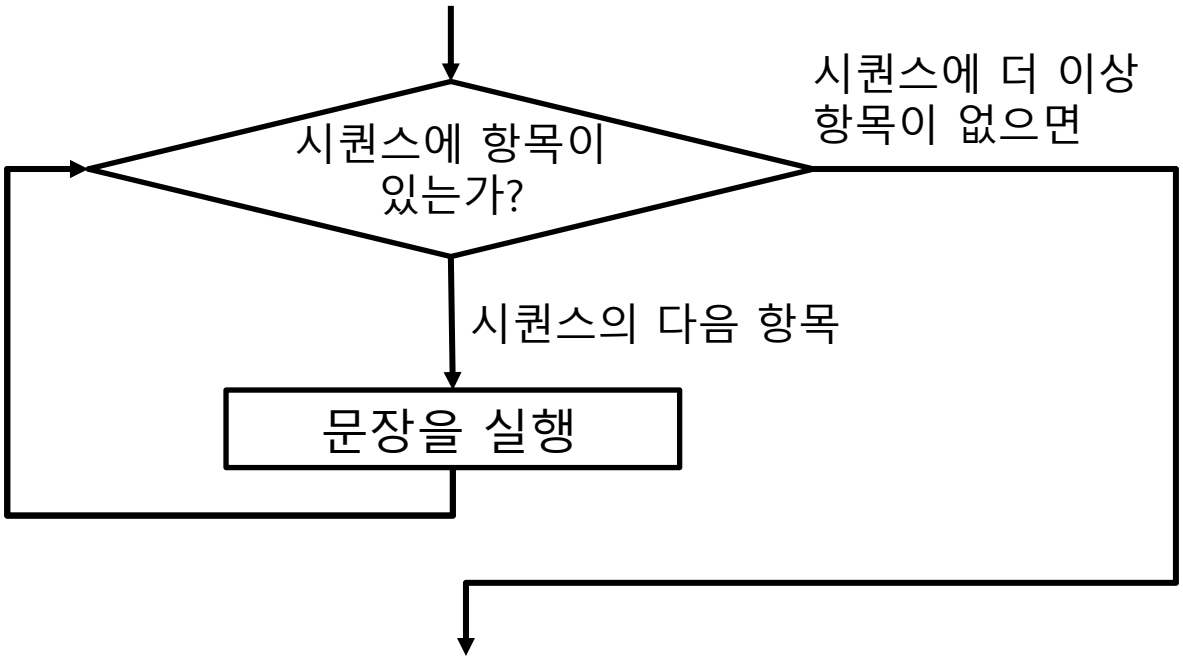
조건 반복

04. 제어문

- 횟수 반복

  반복의 횟수를 미리 아는 경우에 사용

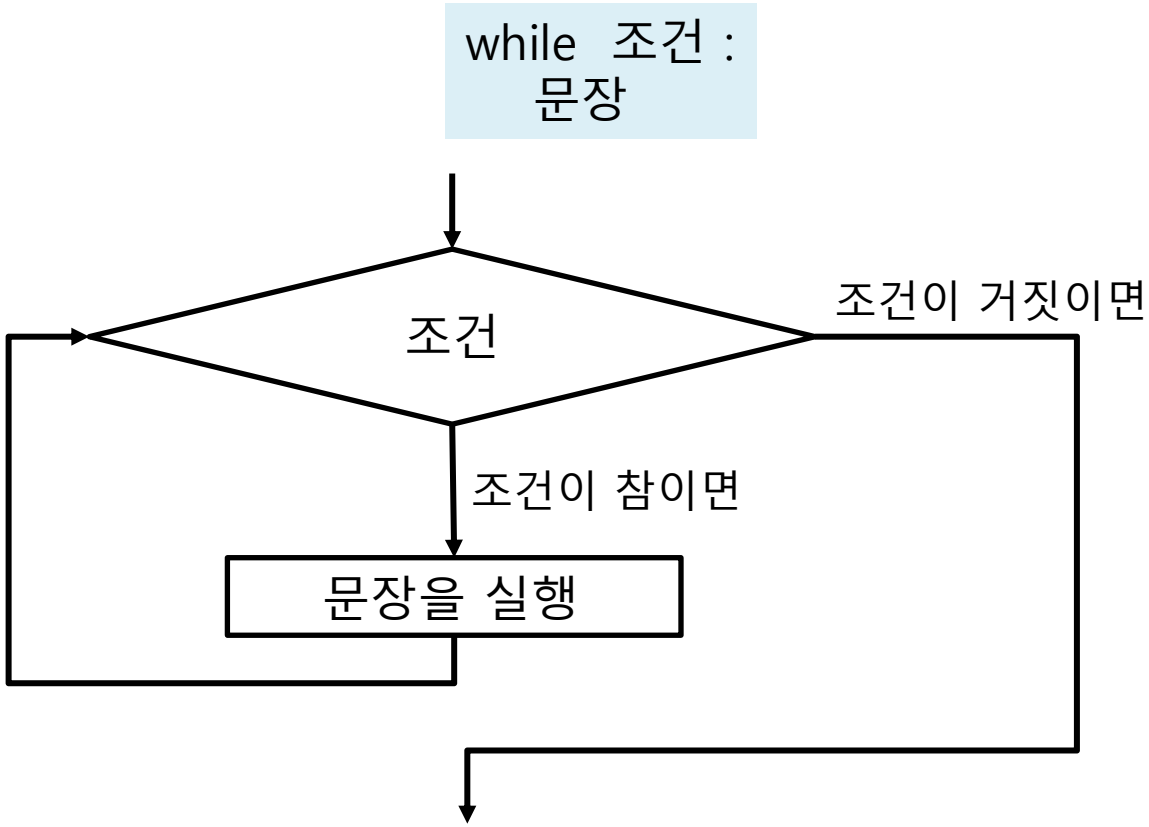
```
for 변수 in 시퀀스 :
    문장
```



04. 제어문

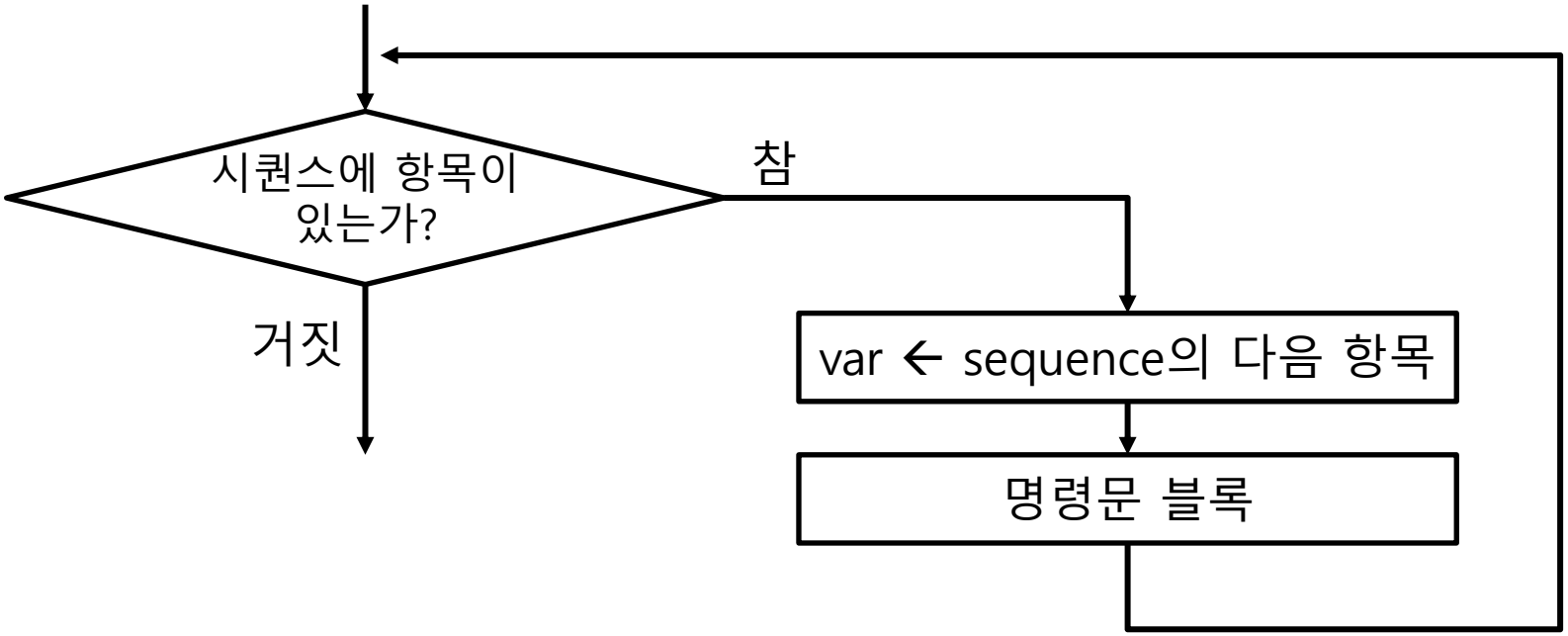
- 조건 반복

  특정한 조건이 만족되는 동안 계속 반복



04. 제어문

for 변수 in 시퀀스 :
    반복 실행할 명령어



04. 제어문

```
>>> for  i  in  [1, 2, 3, 4, 5] :
        print(i, "Good morning")
>>> for  i  in  range(10):
        print(i, end=' ')
>>> for  i  in  range(1, 10):
        print('9 x %d = %2d' %(i, 9*i))
```

04. 제어문

- 어떤 조건이 만족되는 동안(조건이 참인 동안) 반복

```
while 조건식 :
        문장1
        문장2

while money < TARGET :
        money = money + money * rate
        year = year + 1
```

참이나 거짓으로 계산되는 조건식.
관계 연산자 == != < > >= <=을 사용한다.

콜론(:)은 복합문을 의미한다.

조건이 참이면 반복되는 문장들

반복되는 문장은 동일하게
들여쓰기가 되어야 한다.

04. 제어문

➢ 투자금이 2배가 되는데 걸리는 시간 계산

```
# money.py

TARGET = 2000                    # 목표 금액
money = 1000                     # 초기 자금
year = 0                         # 연도
rate = 0.07                      # 이자율

# 현재 금액이 목표 금액보다 작으면 반복한다.
while money < TARGET :
    money = money + money * rate
    year = year + 1

print(year, "년")
```

04. 제어문

➢ 리스트 내포

- 리스트의 요소를 생성하는 문장을 리스트 안에 넣는다

[expression for x in old_list if filter(x)]

출력식 변수의 범위 조건(선택사항)

```
>>> squares = [ ]
>>> for x in range(10):
        if x % 2 == 0:
            squares.append(x*x)

>>> squares = [ x*x for x in range(10) if x % 2 == 0 ]
>>> [x*2 for x in [1,2,3,4,5]]
```

04. 제어문

➢ 리스트 내포

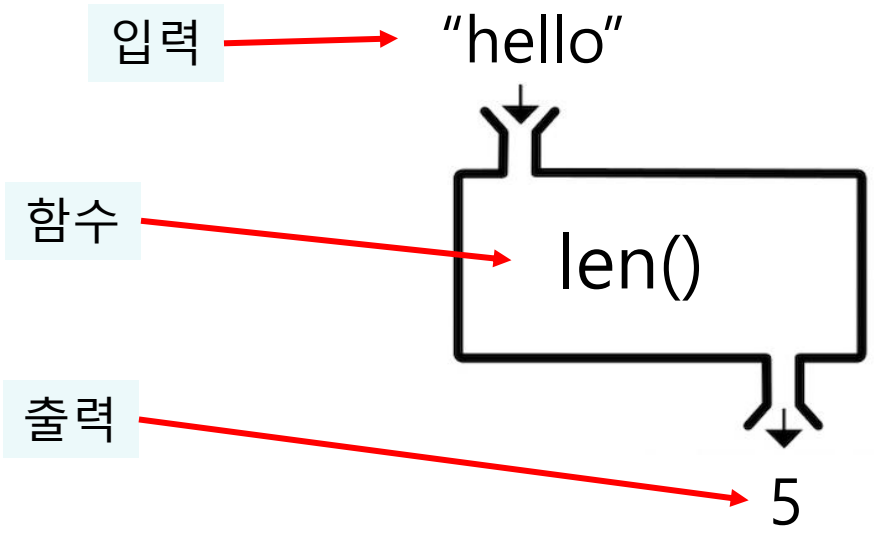
```
>>> prices = [135, -545, 922, 356, -992, 217]
>>> mprices = [ i  if  i > 0  else  0  for  i in  prices ]
>>> mprices

>>> list1 = [ 3, 4, 5]
>>> list2 = [ x*2  for  x  in list1]
>>> n1  = [ x  for  x  in  range(10)  if  x % 2 == 0 ]
>>> n2 = [ x*3  for  x  in  range(10)  if  x % 2 == 0 ]

>>> list1 = ['apple', 'banana', 'tomato', 'kiwi', 'watermelon', 'pear, 'peach']
>>> items = [ word[0]  for  word  in  list1 ]
>>> result = [ len(w)  for  w  in  list1 ]
```

05. 함수

- 함수 : 일을 수행하는 코드의 덩어리

  def 키워드로 함수 정의 후 함수 호출하여 사용

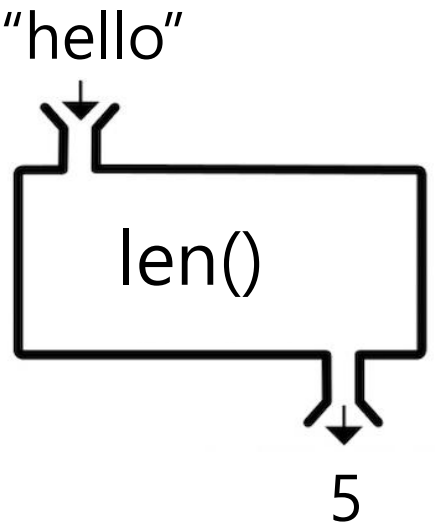
  - ✓ 프로그램 안에서 중복된 코드를 제거한다
  - ✓ 코드를 간결하게 유지할 수 있다
  - ✓ 여러 번 호출하여 사용 가능. 다른 프로그램에서도 재사용될 수 있다
  - ✓ 하나의 큰 프로그램을 나누어 작성할 수 있어 구조화된 프로그래밍이 가능 → 가독성 증대, 유지관리 쉬워진다



05. 함수

- 함수 작성하고 호출하기

```
def 함수이름 (<매개변수1>, <매개변수2>,...) :
    명령어
    명령어
    return <값>
```



05. 함수

➢ 인수와 매개변수

- 인수(argument) : 호출 프로그램에 의해 함수에 전달되는 값(정보)
- 매개변수(parameter) : 이 함수에 전달되는 값을 전달받는 변수

➔ 함수가 호출될 때마다 인수는 함수의 매개 변수로 전달된다.

- 반환 값(return value) : 함수가 호출한 곳으로 반환하는 작업의 결과값

return 키워드 사용

수식 또는 값을 뒤에 쓸 수 있다

함수로부터 반환된 값은 변수에 저장하여 사용

함수가 값을 반환하지 않을 경우 None 반환

05. 함수

- 함수에 여러 개의 입력 전달

```
>>> def  get_sum(start, end) :
        sum = 0
        for  i  in  range(start, end+1) :
            sum += i
        return  sum

>>> sum10 = get_sum(1, 10)
>>> print ('Add 1 to 10 : ', sum10)
>>> sum100 = get_sum(1, 100)
>>> print ('Add 1 to 100 : ', sum100)
```

05. 함수

➢ 디폴트 인수

- 함수의 매개변수가 가지는 기본 값

```
>>> def  greet(name, msg):
        print("안녕", name + ', ' + msg)

>>> print("철수", "좋은 아침!")
>>> print("영희")   ➔ 에러
```

```
>>> def  greet(name, msg = "별일 없죠?"):
        print("안녕", name + ', ' + msg)

>>> print("영희")
>>> print("철수", "좋은 아침!")
```

05. 함수

➢ 키워드 인수

- 인수의 이름을 명시적으로 지정해서 전달하는 방법

- 위치 인수(positional argument)와 같이 사용하는 경우는 위치 인수가 먼저 나와야 한다

```
>>> def  calc(x, y, z):
        print(x)
        print(y)
        print(z)
        return  x + y + z


>>> calc(10, 20, 30)
>>> calc(x=10, y=20, z=30)
>>> calc(y=20, x=10, z=30)
>>> calc(10, y=20, z=30)
```

05. 함수

- 가변 인수 함수
  - 인수의 개수가 정해지지 않은 가변인수에 사용
  - 매개변수 앞에 *을 붙여 사용
  - 함수 안에서는 반복문으로 처리

```
def  함수이름 ( *매개변수) :
     명령어
```

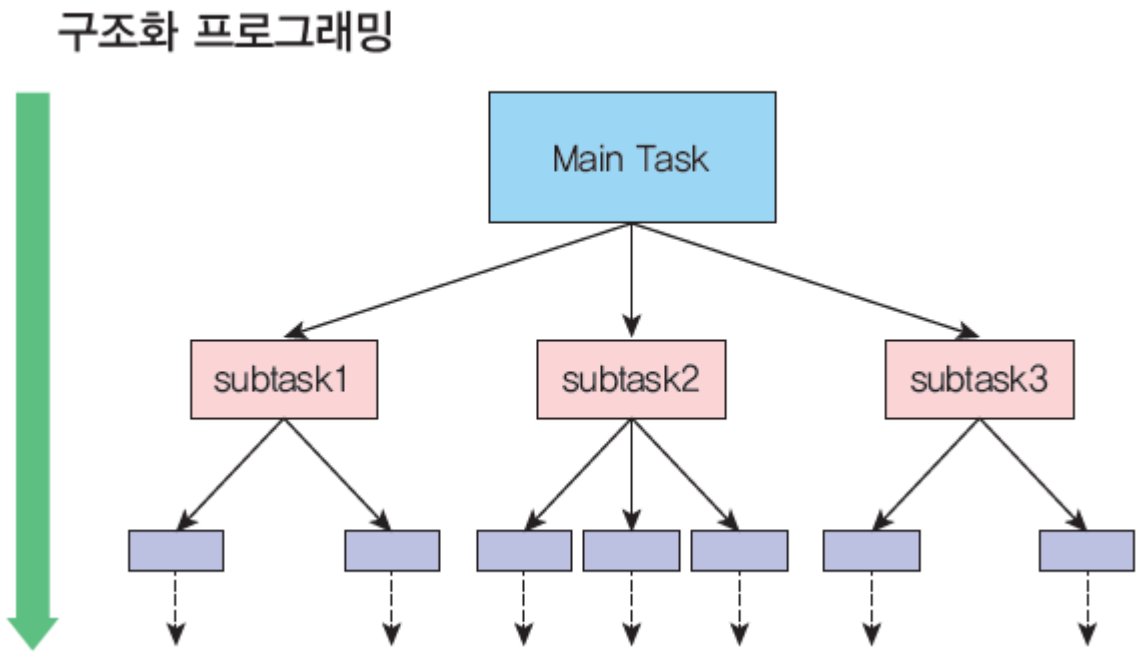
```
>>> def  add_many(*args) :
        result = 0
        for  i  in  args :
            result = result + i
        return  result

>>> add_many(20, 30)
>>> add_many(1,2,3,4,5,6,7,8,9,10)
>>> add_many(1,3,5,7,9)
>>> add_many()
```

05. 함수

➢ 함수를 사용하는 이유

- 소스 코드의 중복성을 없애준다.
- 한번 제작된 함수는 다른 프로그램을 제작할 때도 사용이 가능하다.
- 복잡한 문제를 단순한 부분으로 분해할 수 있다.

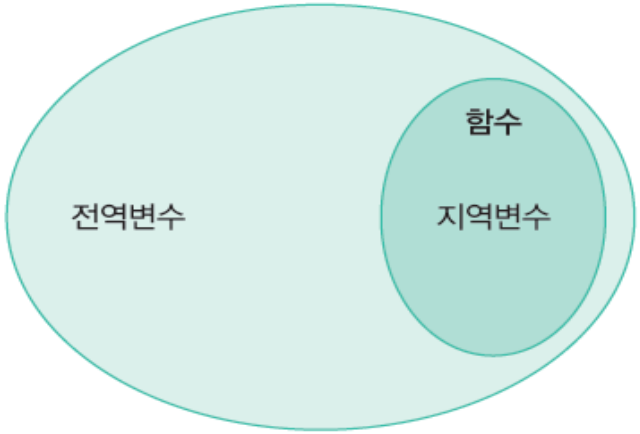
  ➔ 소스의 가독성이 좋아진다.



05. 함수

- 변수의 범위

  - 지역변수(local variable)
    - 함수 안에서 생성되는 변수
    - 함수 안에서만 사용 가능.
    - 함수가 호출될 때 생성되고 함수 호출 종료 시 사라짐
  - 전역변수(global variable) :
    - 함수의 외부에서 생성된 변수
    - 프로그램 어디서나 사용 가능



05. 함수

- 함수 안에서 전역변수 사용하기
  - 전역변수와 지역변수는 같은 이름을 가질 수 있다
  - 함수 안에서 전역변수의 값을 변경하면 지역변수로 처리한다.
  - global 키워드로 함수 안에서 전역변수를 사용할 수 있다.

05. 함수

- 람다 함수(무명 함수)
  - 이름은 없고 몸체만 있는 함수
  - 한 번 사용되는 함수를 만드는데 사용
  - lambda 키워드로 만든다.
  - print()를 호출할 수 없고, 계산만 가능하다.
  - 여러 개의 매개변수를 가질 수 있으나 반환 값은 하나만 있어야 함
  - return 키워드가 필요 없다.

```
lambda   매개변수1, 매개변수2, ... : 수식
```

```
lambda   x, y : x+y
```

05. 함수

- 람다 함수

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


>>> sum = lambda  x, y : x + y
>>> sum(4, 3)
>>> result = sum(4, 3)
>>> print(result)


>>> min = (lambda  x, y : x  if  x < y  else  y)
>>> min(100, 200)
```

05. 함수

- map 함수

  - 리스트나 튜플의 요소 전체에 대해서 어떤 처리를 할 때 사용

```
map(  처리를 하는 함수, 리스트나 튜플)
```

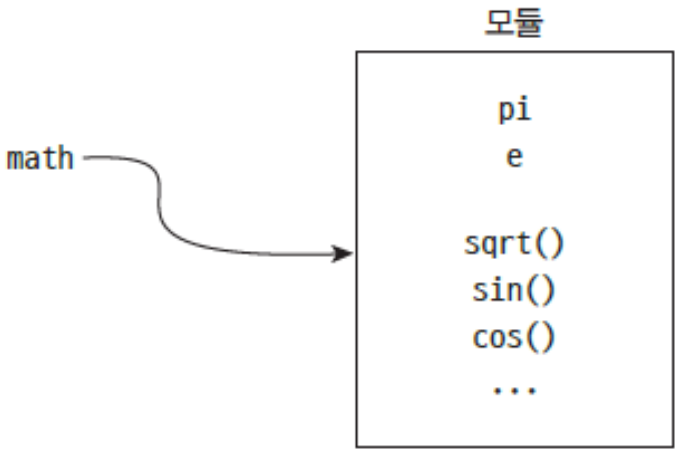
```
>>> def make_double(x):
        return x * 2
>>> list(map(make_double, [1,2,3]))
```

```
>>> list(map(lambda x: x*2, [1,2,3]))
```

06. 모듈

- 모듈 : 각종 변수, 함수, 클래스를 담고 있는 파일

  .py. 파일 단위로 작성
- 패키지 : 특정 기능과 관련된 여러 모듈을 묶은 것
- 파이썬 표준 라이브러리 : 파이썬에 기본으로 설치된 모듈과 패키지, 내장 함수를 묶은 것



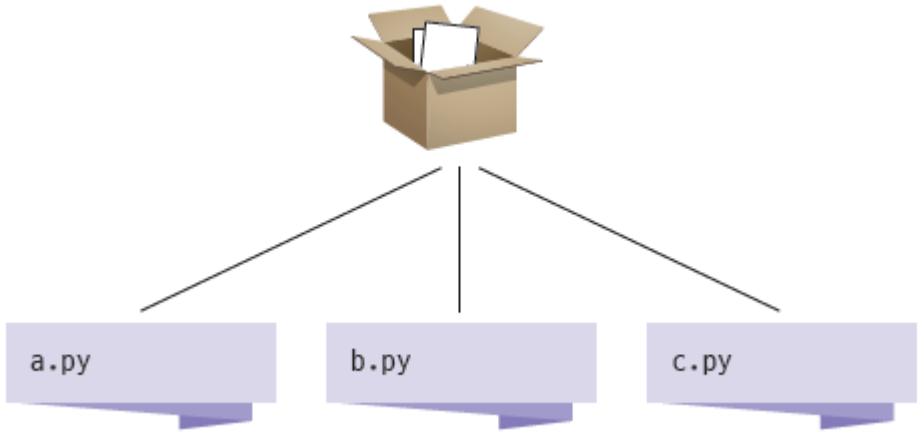
06. 모듈

- 모듈 : 함수나 변수 또는 클래스를 모아 놓은 파일

  길어지는 파일을 여러 개의 파일로 분할 – 유지 보수가 쉽다

  다른 파이썬 프로그램에서 불러와 사용할 수 있게 만든 파이썬 파일

  모듈 안에 있는 함수들은 import 문장으로 다른 모듈로 포함될 수 있다



06. 모듈

- import : 이미 만들어진 파이썬 모듈을 사용할 수 있게 해주는 명령어
  모듈 이름은 .py를 제거한 파이썬 파일 이름이다

➢ import 모듈이름

```
>>> import mod1
>>> mod1.add(11, 33)
>>> mod1.sub(23, 10)
```

➢ from 모듈이름 import 모듈함수

```
>>> from mod1 import add
>>> add(23, 100)
>>> from mod1 import add, sub
>>> from mod1 import *
```

➢ import 모듈이름 as 별명

```
>>> import mod1 as m
>>> m.add(22, 44)
>>> m.sub(82, 11)
```

06. 모듈

- import 모듈
- import 모듈1, 모듈2
- 모듈.변수
- 모듈.함수()
- 모듈.클래스()

```
>>> import math
>>> math.pi
3.141592653589793

>>> import math
>>> math.sqrt(4.0)
2.0
>>> math.sqrt(2.0)
1.4142135623730951
```

06. 모듈

- import 모듈 as 이름 ➔ 모듈이름(별명)지정

```
>>> import math as m
>>> m.pi
3.141592653589793
>>> m.sqrt(3.0)
1.732050807568772
```

- from 모듈 import 변수 ➔ 모듈의 일부만 가져오기
- from 모듈 import 함수
- from 모듈 import 클래스

```
>>> from math import pi
>>> pi
3.141592653589793
>>> from math import sqrt
>>> sqrt(4.0)
2.0
```

06. 모듈

- from 모듈 import 변수, 함수, 클래스 ➔ 모듈 안의 여러 개 변수, 함수, 클래스 가져오기

```
>>> from math import pi, sqrt
>>> pi
3.141592653589793
>>> sqrt(2.0)
1.4142135623730951
```

- from 모듈 import * ➔ 모듈 안의 모든 변수, 함수, 클래스 가져오기

```
>>> from math import *
>>> pi
3.141592653589793
>>> sqrt(2.0)
1.4142135623730951
```

06. 모듈

- from 모듈 import 변수 as 이름 ➔ 모듈의 일부를 가져온 뒤 이름 지정하기
- from 모듈 import 변수 as 이름1, 함수 as 이름2, 클래스 as 이름3

```
>>> from math import sqrt as s
>>> s(3.0)
1.732050807568772

>>> from math import pi as p, sqrt as s
>>> p
3.141592653589793
>>> s(2.0)
1.4142135623730951
```

06. 모듈

```
# square2.py        거듭제곱 구하기

base = 2

def  square(n):
      return  base ** n
```

```
>>> import square2
>>> print(square2.base)
2
>>> print(square2.square(5))
32
>>>
```

```
# main.py

import  square2

print(square2.base)
print(square2.square(10))
```

06. 모듈

➢ 모듈에 클래스 작성하기

```
#person.py

class   Person:
    def __init__(self, name, age, address):
        self.name = name
        self.age = age
        self.address = address

    def   greeting(self):
        print('안녕하세요. 저는 {0}입니다.'.format(self.name))
```

```
# main1.py

import   person

maria = person.Person('마리아',20,'서울')
maria.greeting()
```

06. 모듈

➢ 스크립트 파일로 실행하거나 모듈로 사용하는 코드 만들기

```
# calc.py

def  add(a, b):
    return a + b

def  mul(a, b):
    return a * b

if __name__ == '__main__':
    print(add(10, 20))
    print(mul(10, 20))
```

파일을 독립적으로 실행시키면 실행되지만 다른 파일에서 import 할 때에는 실행되지 않는다

```
30
200
```
➔ calc.py 실행하면 출력

```
>>> import  calc
```
➔ calc 를 모듈로 사용시 if 문 이하 실행되지 않음

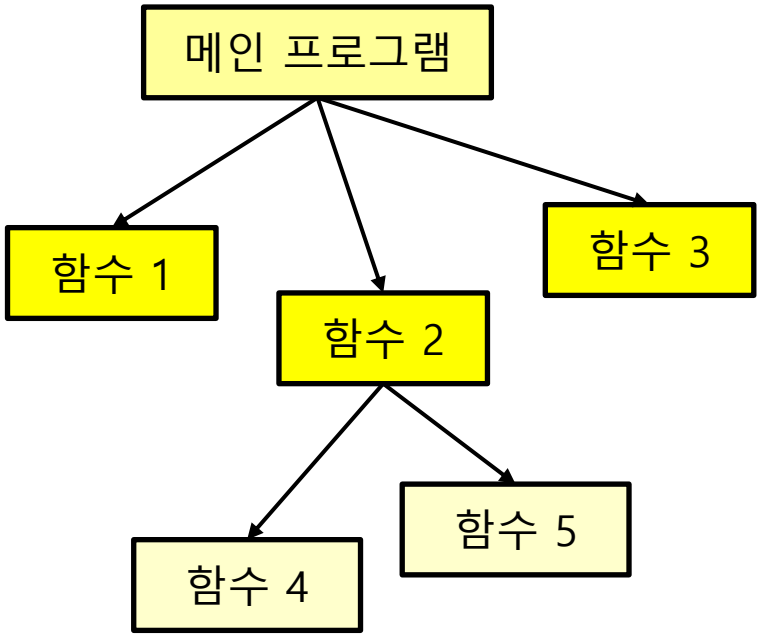
09. 객체지향

- 객체 지향 프로그래밍에서는 서로 관련 있는 데이터와 함수를 묶어서 객체(object)로 만들고 이들 객체들이 모여서 하나의 프로그램이 된다.

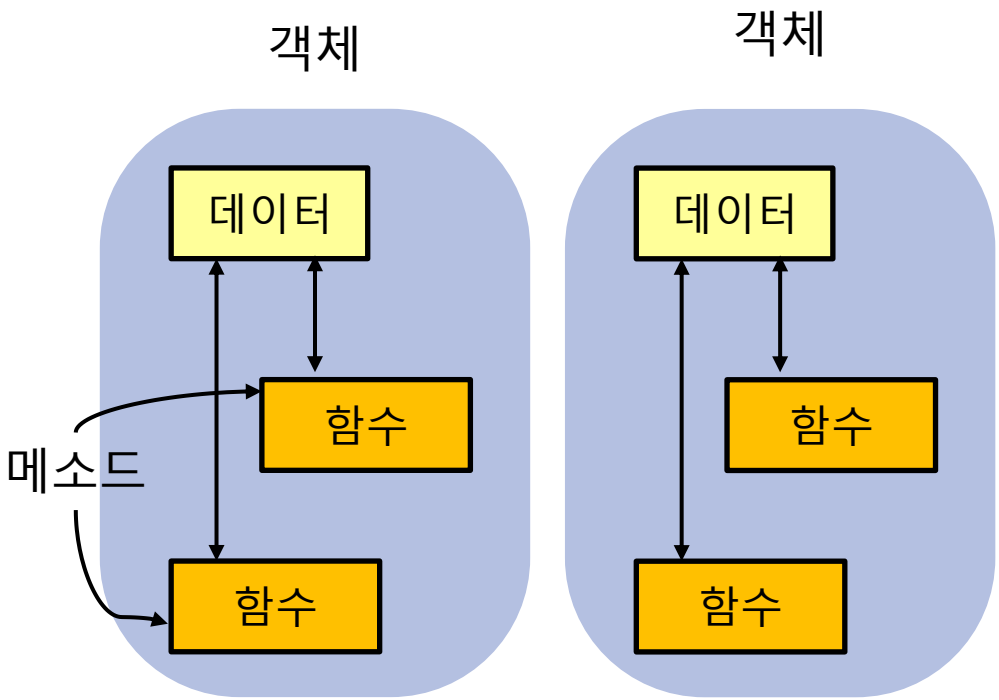
09. 객체지향

- 절차 지향 프로그래밍(procedural programming) : 프로시저(procedure)를 기반으로 하는 프로그래밍 방법
- 객체 지향 프로그래밍(OOP : Object-Oriented Programming) : 소프트웨어를 객체로 구성하는 방법

절차 지향 프로그래밍



객체 지향 프로그래밍



09. 객체지향

- 객체 지향 프로그래밍 : 데이터와 함수를 하나의 덩어리로 묶어서 생각하는 방법
- 물건 ➔ 객체(object) , 인스턴스
  - 물건의 특징(속성) : 메이커, 모델, 색상, 연식, 가격 ➔ 프로퍼티
  - 물건의 조작(동작) : 주행하기, 방향 전환, 정지 ➔ 메서드

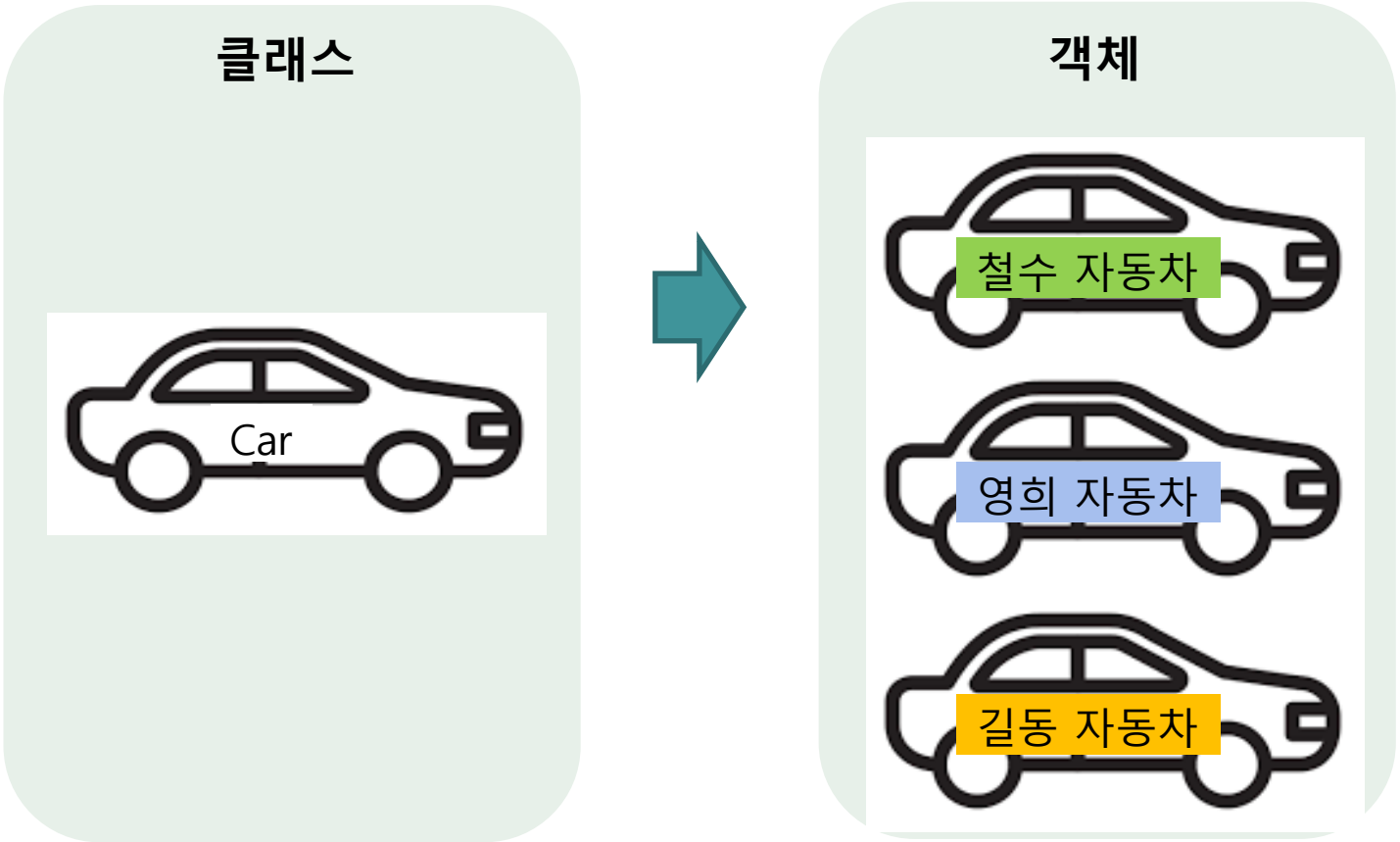
객체 = 변수 + 함수

속성
메이커
모델
색상
연식
가격

동작
주행하기
방향 바꾸기
주차하기

09. 객체지향

- 클래스(class) : 똑같은 무언가를 계속해서 만들어낼 수 있는 설계도면
- 객체(object) : 클래스로 만든 피조물
- 클래스로부터 만들어지는 각각의 객체를 그 클래스의 인스턴스(instance) 라 한다



09. 객체지향

- 생성자 : 객체를 만들기 위한 전용 함수

형식

```
class  클래스이름:
    def  __init__(self, ...):
        ...
    def  메소드1(self, ...):
        ...
    def  메소드2(self, ...):
        ...
```

생성자(Constructor)

예

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

생성자 정의

메소드 정의

09. 객체지향

- 클래스 작성하기

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


a = Counter()
print("count 의 값 : ", a.count)
a.increment()
print("count 의 값 : ", a.count)
```

09. 객체지향

- 파이썬에서는 클래스 당 하나의 생성자만 허용한다.

```
class Counter:
    def __init__(self, initvalue = 0):
        self.count = initvalue
    def increment(self):
        self.count += 1

a = Counter(100)
print("count 의 값 : ", a.count)
a.increment()
print("count 의 값 : ", a.count)

b = Counter()
print("count 의 값 : ", b.count)
b.increment()
print("count 의 값 : ", b.count)
```

09. 객체지향

- 변수

  - self 매개변수 : 객체 자신을 참조하는 변수

    객체.메소드() 로 사용 ➔ 객체가 self 로 전달됨

  - 인스턴스 변수 : 생성자 안에서 생성된 변수, 앞에 self가 붙는다

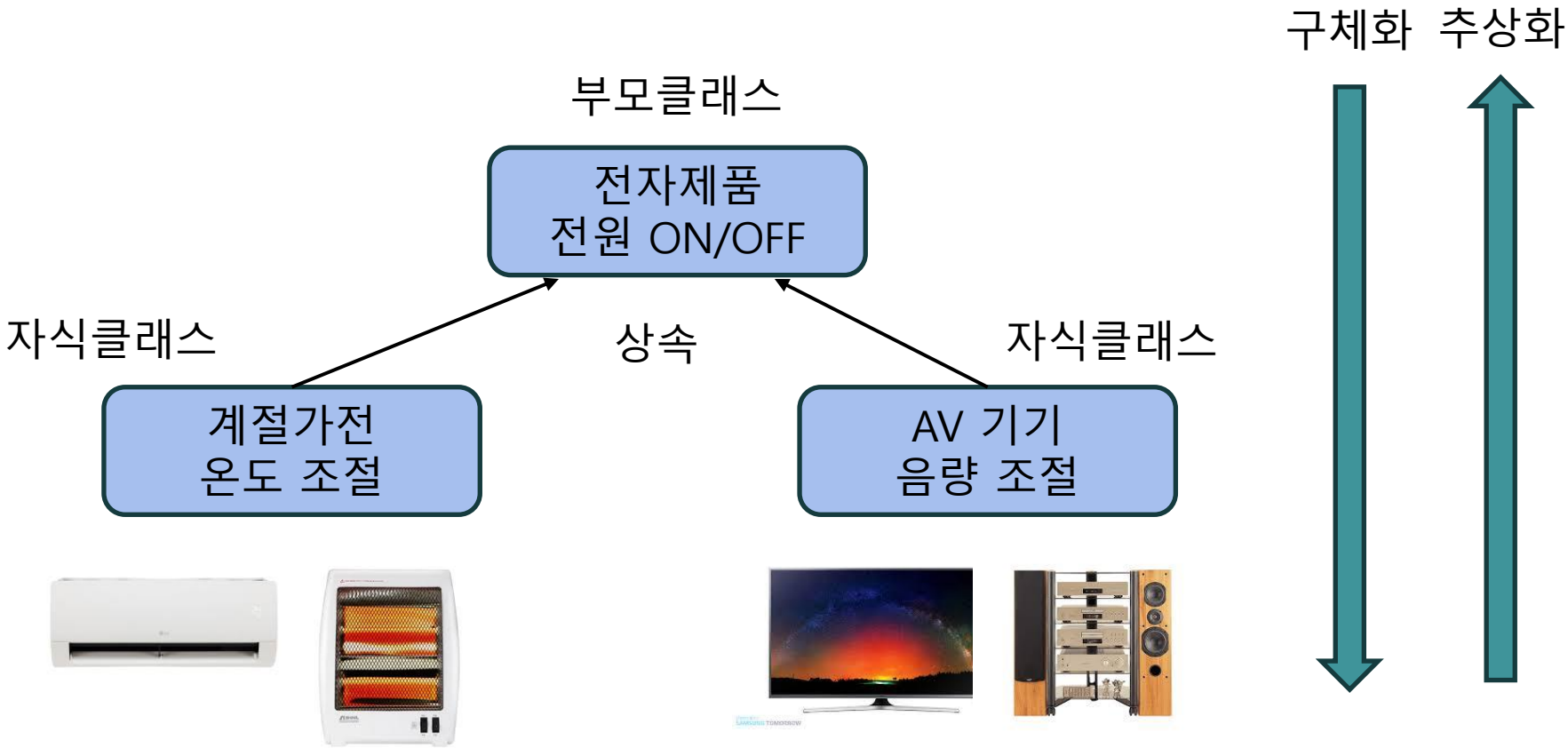
    클래스 전체에서 사용 가능

  - 지역 변수 : 함수 안에서 선언되는 변수

  - 전역 변수 : 함수 외부에서 선언되는 변수

09. 객체지향

- 상속(inheritance) : 기존에 존재하는 클래스로부터 코드와 데이터를 이어받고 자신이 필요한 기능을 추가하는 기법

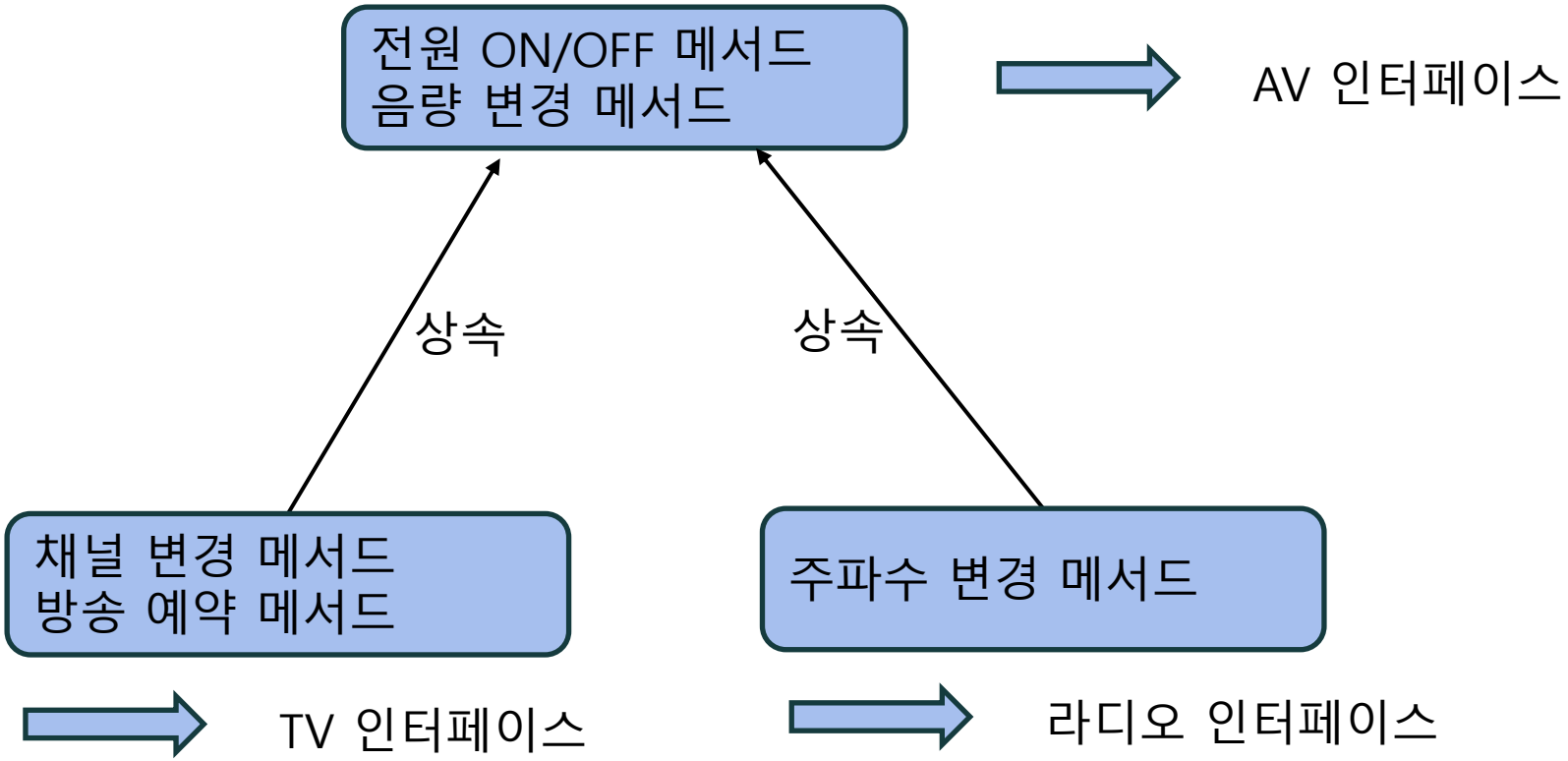


09. 객체지향

- 부모 클래스(super class, base class) : 추상적
- 자식 클래스(sub class, derived class) : 구체적

부모 클래스	자식 클래스
Animal(동물)	Lion(사자), Dog(개), Cat(고양이)
Bike(자전거)	MountainBike(산악자전거), RoadBike, TandemBike
Vehicle(탈것)	Car(자동차), Bus(버스), Truck(트럭), Boat(보트), Motorcycle(오토바이), Bicycle(자전거)
Student(학생)	GraduateStudent(대학원생), UnderGraduate(학부생)
Employee(직원)	Manager(관리자)
Shape(도형)	Rectangle(사각형), TTriangle(삼각형), Circle(원)

09. 객체지향


전원 ON/OFF 메서드
음량 변경 메서드
AV 인터페이스
상속
상속
채널 변경 메서드
방송 예약 메서드
주파수 변경 메서드
TV 인터페이스
라디오 인터페이스

09. 객체지향

- 상속(inheritance)

```
>>> import  pygame
>>> class MyRect(pygame.Rect):
        def flip(self):
            self.width, self.height = (self.height, self.width)


>>> r = MyRect(10, 20, 30, 40)
>>> r.size
(30, 40)
>>> r.flip()
>>> r.size
(40, 30)
```

09. 객체지향

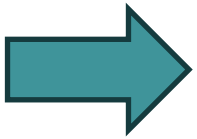
- 상속(inheritance)

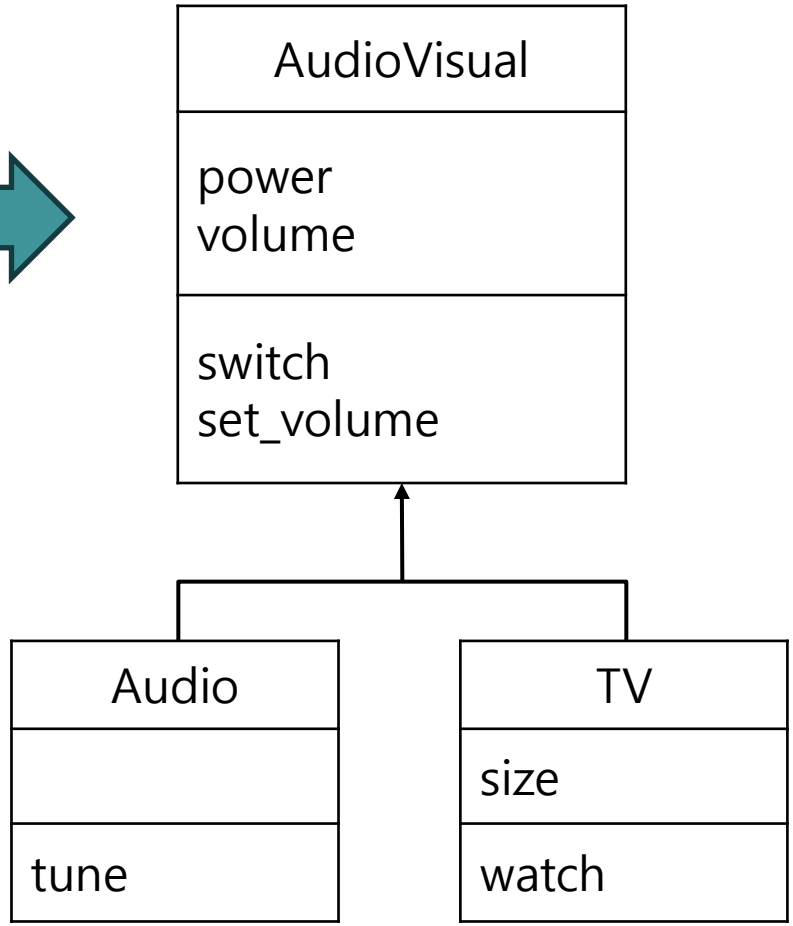
```
class Audio:
    def __init__(self, power, volume):
        slef.power = power
        self.volume = volume
    def switch(self, on_off):
        self.on_off = on_off
    def set_volume(self, vol):
        self.vol = vol
    def tune(self):
        str = "La la la ..." if
self.power else "turn it on"
        print(str)

mp3 = Audio(False, 8)
mp3.set_volume(12)
mp3.tune()
```

```
class TV:
    def __init__(self, power, volume, size):
        slef.power = power
        self.volume = volume
        self.size = size
    def switch(self, on_off):
        self.on_off = on_off
    def set_volume(self, vol):
        self.vol = vol
    def watch(self):
        str = "have fun!!" if  self.power
else "switch on"
        print(str)

obj = TV(True, 14, 40)
obj.switch(True)
obj.watch()
obj.set_volume(10)
```

09. 객체지향

Audio	TV
power volume	power volume size
switch set_volume tune	switch set_volume watch





09. 객체지향

```
# classInheritance.py

class AudioVisual:
    def __init__(self, power, volume):
        self.power = power
        self.volume = volume
    def switch(self, on_off):
        self.on_off = on_off
    def set_volume(self, vol):
        self.vol = vol

class Audio(AudioVisual):
    def __init__(self, power, volume):
        super().__init__(power, volume)
    def tune(self):
        str = "La la la ..." if  self.power else "turn it on"
        print(str)
```

09. 객체지향

```
class TV(AudioVisual):
    def __init__(self, power, volume, size):
        super().__init__(power, volume)
        self.size = size
    def watch(self):
        str = "have fun!!" if  self.power else "switch on"
        print(str)


Obj1 = TV(False, 12, 40)
Obj1.switch(True)
Obj1.watch()
#print("TV volume : ", Obj1.volume)

Obj2 = Audio(True, 15)
Obj2.set_volume(6)
Obj2.tune()
#print("Audio volume : ", Obj2.volume)
```