실전코딩

02. 파이썬 프로그래밍 리뷰
- 함수, 객체지향 프로그래밍 -

강원대학교 컴퓨터공학과 박치현

Outline

- 함수
  - 함수란
  - 함수의 구조
  - 함수 내 변수
- 객체지향프로그래밍
  - 객체지향프로그래밍 (OOP)
  - 클래스와 객체 (Class and Instance)
  - 추상화 (Abstraction)
  - 캡슐화 (Encapsulation)
  - 상속성 (Inheritance)
  - 다형성 (Polymorphism)
- 모듈

함수

- 함수

  특정 목적의 작업 수행을 위해 독립적으로 설계된 코드의 집합

f(x) = x + 8

x=1, f(1) = 9

x=2, f(2) = 10

x=3, f(3) = 11

.....

함수의 구조

- 함수

  특정 목적의 작업 수행을 위해 독립적으로 설계된 코드의 집합

  작업에 필요한 데이터를 전달받아 그 결과를 반환하는 구조

def name (parameter):
실행 코드
retrun 결과

함수의 구조
-함수의 이름**(name):** 식별자 규칙에 따라 작성된 사용자 정의 이름
-**parameter:** 함수에 입력으로 전달된 값을 받는 변수
-실행코드 **:** 함수에서 실행될 명령어 집합
-**return** 결과 **:** 함수의 반환값

c.f.) 목적에 따라 작성되므로 **parameter**와 **return** 결과가 없는 함수도 존재

실행 구조
-함수 사용을 원하는 부분에서 함수의 이름을 작성하여 호출 **(call)**
-함수 구조에 맞는, **parameter**에 전달될 **arguments**를 입력
-이 후 **return** 결과를 사용

함수의 구조

- 함수

  특정 목적의 작업 수행을 위해 독립적으로 설계된 코드의 집합

  작업에 필요한 데이터를 전달받아 그 결과를 반환하는 구조

def name (parameter):
실행 코드
retrun 결과

f(x) = x + 8

x=1, f(1) = 9

x=2, f(2) = 10

x=3, f(3) = 11

.....

parameter

```
def func1(x):
    a = x + 8
    return a

var1 = func1(1)
print(var1)          #결과 : 9
print(func1(2))      #결과 : 10
print(func1(3))      #결과 : 11
```

argument

함수의 구조

- 함수

  특정 목적의 작업 수행을 위해 독립적으로 설계된 코드의 집합

  작업에 필요한 데이터를 전달받아 그 결과를 반환하는 구조

def name (parameter):
실행 코드
retrun 결과

f(x) = x + 8

x=1, f(1) = 9

x=2, f(2) = 10

x=3, f(3) = 11

.....

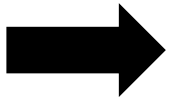
```
def func1(x):
    a = x + 8
    return a

def func2(x):
    return x + 8
```

```
var1 = func1(1)
print(var1)          #결과 : 9
print(func1(2))      #결과 : 10
print(func1(3))      #결과 : 11
```

함수의 구조

- 함수 활용의 예

```
1  print("test start.")
2
3  l1 = [1, 2, 3, 4, 5]
4
5  sum = 0.0
6  for i in l1:
7      sum += i
8
9  avg_l1 = sum / len(l1)
10 print("average of l1: ", avg_l1)
11
12 l2 = [1, 3, 5, 7, 9]
13
14 sum = 0.0
15 for i in l2:
16     sum += i
17
18 avg_l2 = sum / len(l2)
19 print("average of l2: ", avg_l2)
20
21 l3 = [2, 4, 6, 8, 10]
22
23 sum = 0.0
24 for i in l3:
25     sum += i
26
27 avg_l3 = sum / len(l3)
28 print("average of l3: ", avg_l3)
29
30 print("test end.")
```

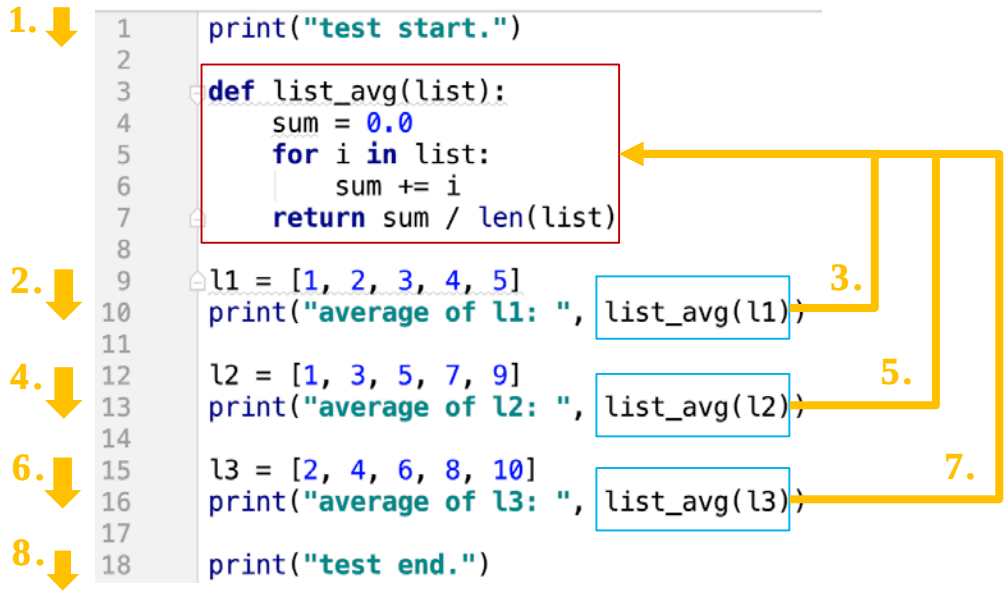


c.f.) List: 값들이 연속적으로 저장되는 자료구조
변수명 =**[v1, v2, v3, ..., vn]**

```
1  print("test start.")
2
3  def list_avg(list):
4      sum = 0.0
5      for i in list:
6          sum += i
7      return sum / len(list)
8
9  l1 = [1, 2, 3, 4, 5]
10 print("average of l1: ", list_avg(l1))
11
12 l2 = [1, 3, 5, 7, 9]
13 print("average of l2: ", list_avg(l2))
14
15 l3 = [2, 4, 6, 8, 10]
16 print("average of l3: ", list_avg(l3))
17
18 print("test end.")
```

함수의 구조

- 함수 활용의 예



실행 결과

```
test start.
average of l1:  3.0
average of l2:  5.0
average of l3:  6.0
test end.
```

함수의 구조

- 함수 활용의 예

함수 **list_avg**를 호출 **(call)**

```
list_avg(l1)
```

```
l1 = [1, 2, 3, 4, 5]
```

```
def list_avg(list):
    sum = 0.0
    for i in list:
        sum += i
    return sum / len(list)
```

c.f.) len(list): 리스트 원소의 갯수를 **return**

1) i의 값 : 1 → sum += i → sum = 1

2) i의 값 : 2 → sum += i → sum = 3

3) i의 값 : 3 → sum += i → sum = 6

4) i의 값 : 4 → sum += i → sum = 10

5) i의 값 : 5 → sum += i → sum = 15

→ return 15 / 5

함수의 구조

- 함수 활용의 예

  특정 목적의 작업 수행을 위해 독립적으로 설계된 코드의 집합

함수 **list_avg**를 호출 **(call)**

```
list_avg(l1)
```

```
l1 = [1, 2, 3, 4, 5]
```

```
def list_avg(list):
    sum = 0.0
    for i in list:
        sum += i
    return sum / len(list)
```

→ **return 15 / 5 = 3.0**

```
print("average of l1: ", list_avg(l1))
```

e.g.) var1 = list_avg(l1)
print(var1) #3.0

c.f.) len(list): 리스트 원소의 갯수를 **return**

함수의 구조

		내장 함수		
abs()	dict()	help()	min()	setattr()
all()	dir()	hex()	next()	slice()
any()	divmod()	id()	object()	sorted()
ascii()	enumerate()	input()	oct()	staticmethod()
bin()	eval()	int()	open()	str()
bool()	exec()	isinstance()	ord()	sum()
bytearray()	filter()	issubclass()	pow()	super()
bytes()	float()	iter()	print()	tuple()
callable()	format()	len()	property()	type()
chr()	frozenset()	list()	range()	vars()
classmethod()	getattr()	locals()	repr()	zip()
compile()	globals()	map()	reversed()	__import__()
complex()	hasattr()	max()	round()	
delattr()	hash()	memoryview()	set()	

참고링크 : https://docs.python.org/ko/3.6/library/functions.html

함수의 구조

- 함수

  특정 목적의 작업 수행을 위해 독립적으로 설계된 코드의 집합

1) 일반적인 구조	**def name (parameter):** 실행 코드 **retrun** 결과
2) parameter가 없는 구조	**def name ():** 실행 코드 **retrun** 결과
3) return이 없는 구조	**def name (parameter):** 실행 코드
4) parameter, return 이 없는 구조	**def name ():** 실행 코드

함수 내 변수

- 함수 내 변수의 수명

  함수 내 변수의 수명은 변수가 선언된 함수의 내부

```
1    a = 29
2
3    def func1(a):
4        a = a + 5
5        return a
6
7    print("1. call func1: ", func1(a))
8    print("2. print a:    ", a)
```

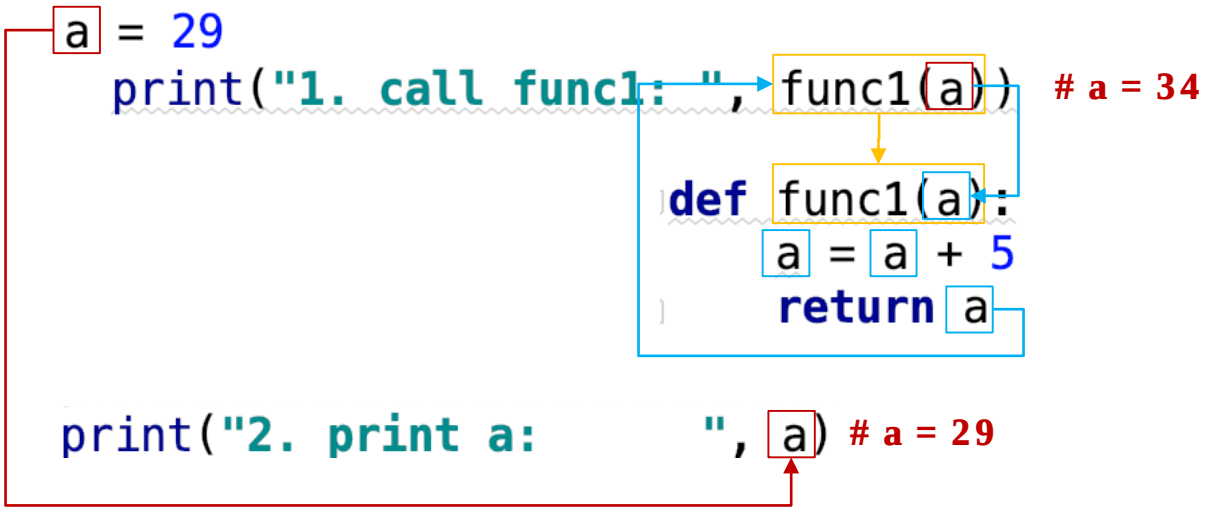
실행 결과

```
1. call func1:  34
2. print a:     29
```

함수 내 변수

- 함수 내 변수의 수명

  함수 내 변수의 수명은 변수가 선언된 함수의 내부



함수 내 변수

- 함수 내 변수의 수명

  함수 내 변수의 수명은 변수가 선언된 함수의 내부

```
1
2     def func1(a):
3         a = a + 5
4         return a
5
6     print("1. call func1: ", func1(29))
7     print("2. print a:    ", a)
```

```
    print("2. print a:    ", a)
NameError: name 'a' is not defined
```

Outline

- 함수
  - 함수란
  - 함수의 구조
  - 함수 내 변수
- **객체지향프로그래밍**
  - 객체지향프로그래밍 (OOP)
  - 클래스와 객체 (Class and Instance)
  - 추상화 (Abstraction)
  - 캡슐화 (Encapsulation)
  - 상속성 (Inheritance)
  - 다형성 (Polymorphism)
- 모듈

OOP 란

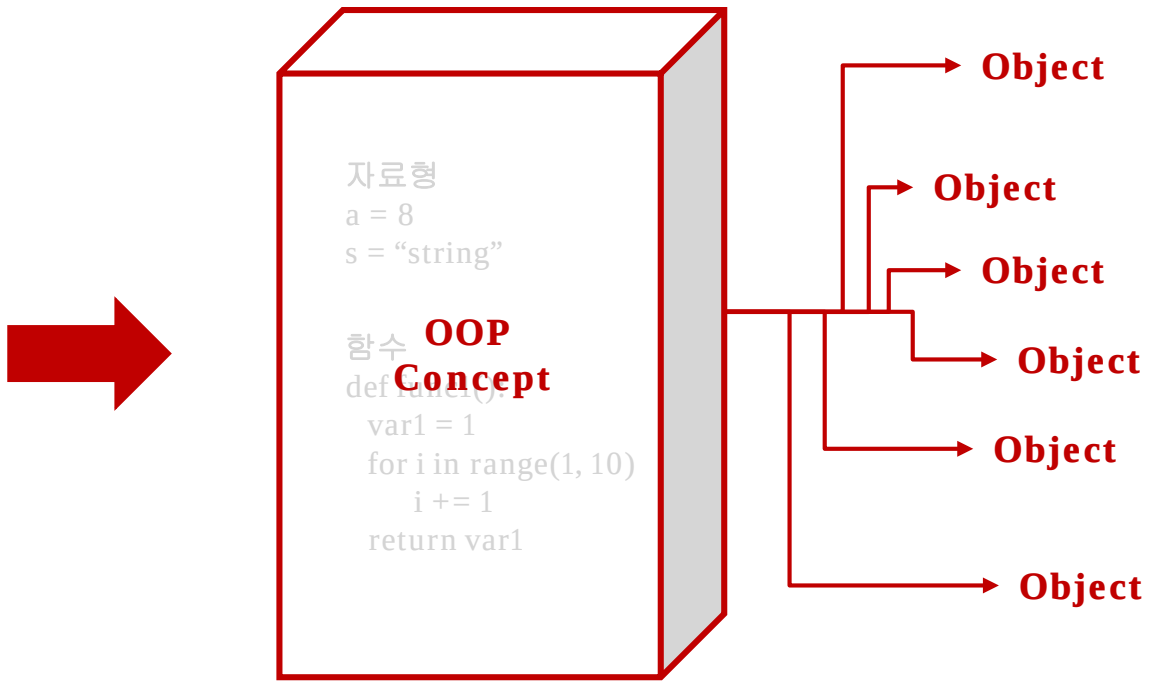
- 객체지향프로그래밍 (Object Oriented Programming)

  추상화, 캡슐화, 상속성, 그리고 다형성의 특징을 갖는 컴퓨터 프로그래밍 패러다임

자료형
```
a = 8
s = “string”
```

흐름제어
```
if(a > 8):
  var1 = 1
  for i in range(1, 10)
      i += 1
```

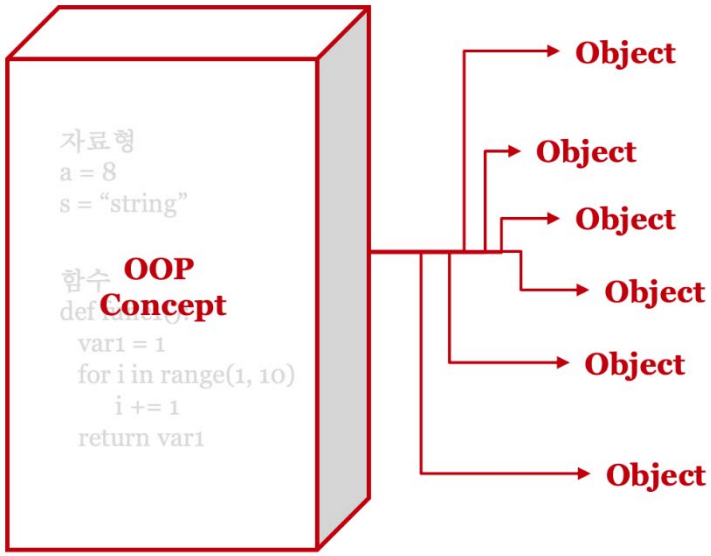
함수
```
def func1():
  var1 = 1
  for i in range(1, 10)
      i += 1
  return var1
```

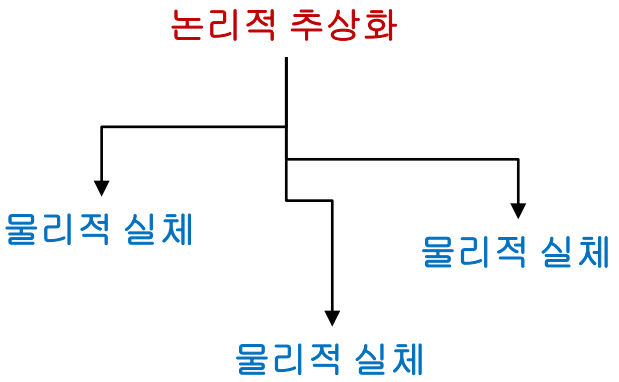


OOP 란

- 객체지향프로그래밍 (Object Oriented Programming)

  추상화, 캡슐화, 상속성, 그리고 다형성의 특징을 갖는 컴퓨터 프로그래밍 패러다임





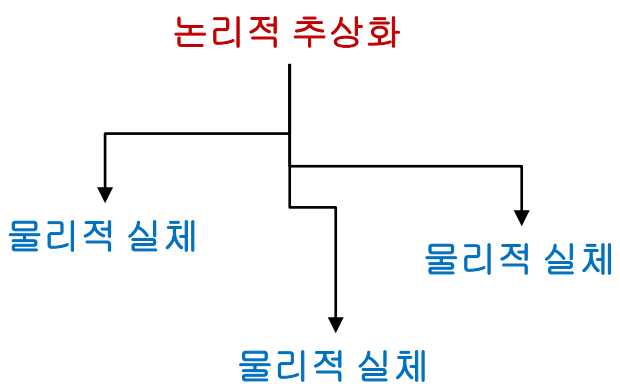
a = 8

a라는 변수는 실제 메모리 공간에 존재하는
정수형이라는 논리적 개념의 물리적 실체

OOP – 클래스와 객체

- 객체지향프로그래밍 **(Object Oriented Programming)**

  클래스와 객체 **(Class and Instance)**



Class 속성과 행위를 변수와 메소드로 정의한 논리적 개념

Instance **Class**의 정의를 기반으로 실제 메모리에 할당된 물리적 실체

추상화란? 공통적인 속성과 행위를 추출하여 구성하는 것

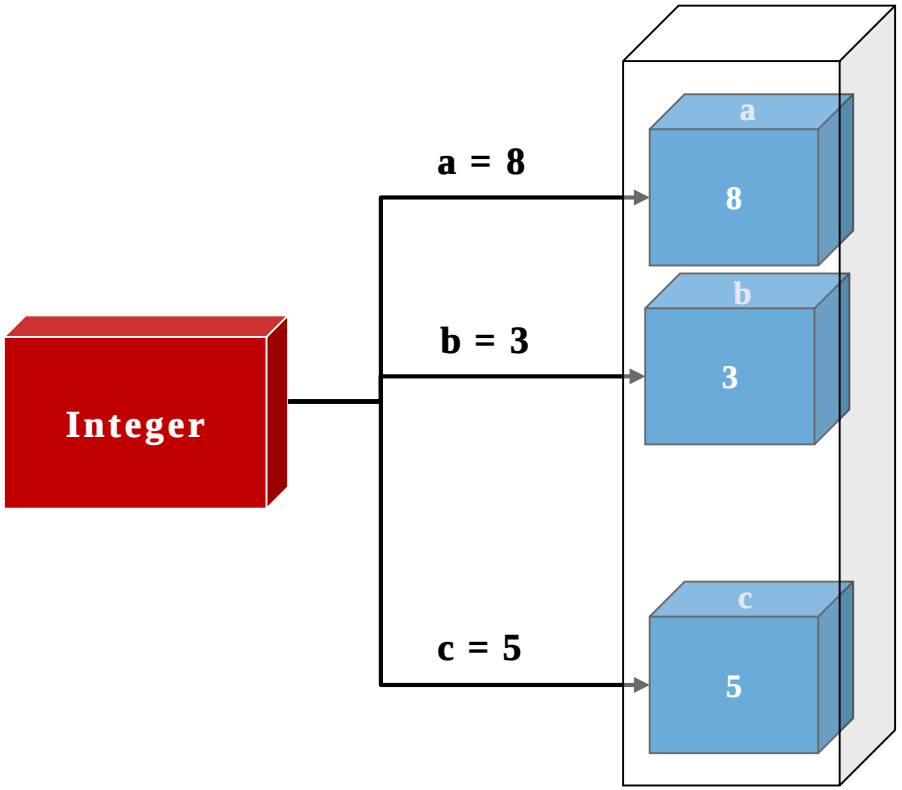
OOP – 클래스와 객체

- 객체지향프로그래밍 **(Object Oriented Programming)**

  클래스와 객체 **(Class and Instance)**

Class - 논리적 추상화

Instance - 물리적 실체



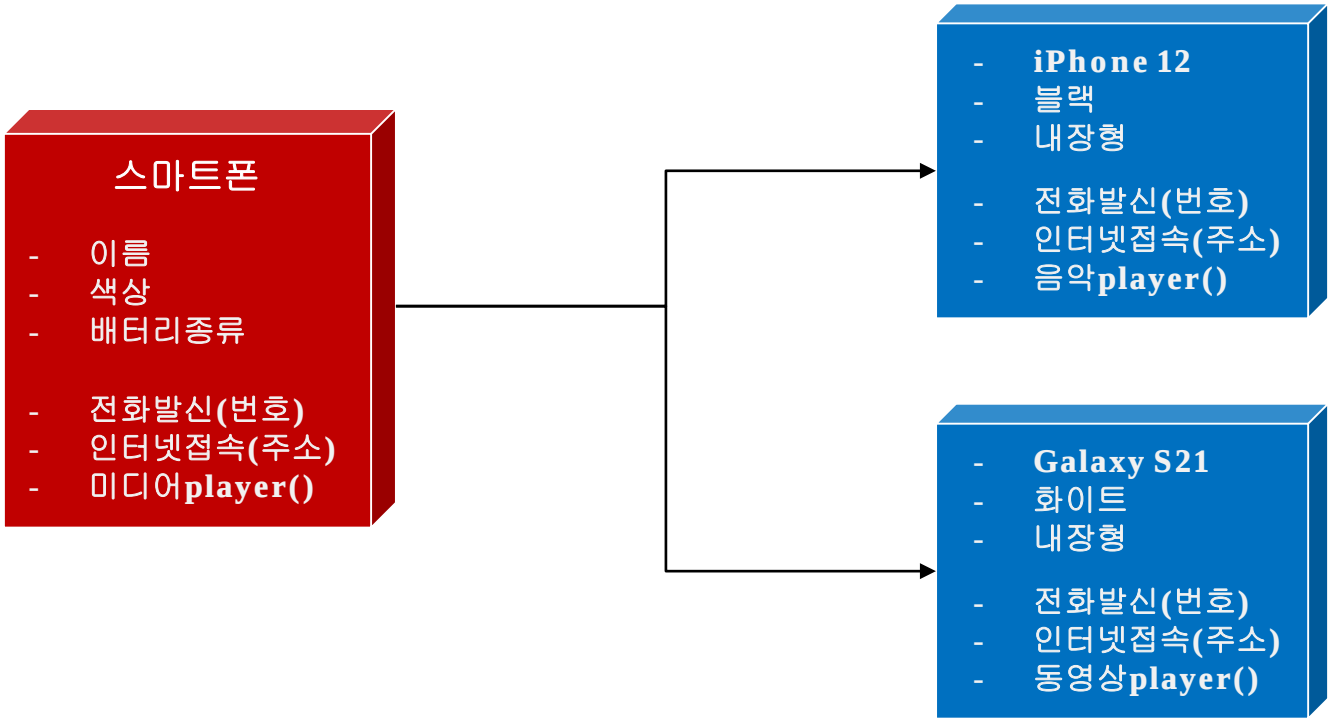
OOP – 클래스와 객체

- 객체지향프로그래밍 **(Object Oriented Programming)**

  클래스와 객체 **(Class and Instance)**

Class - 논리적 추상화

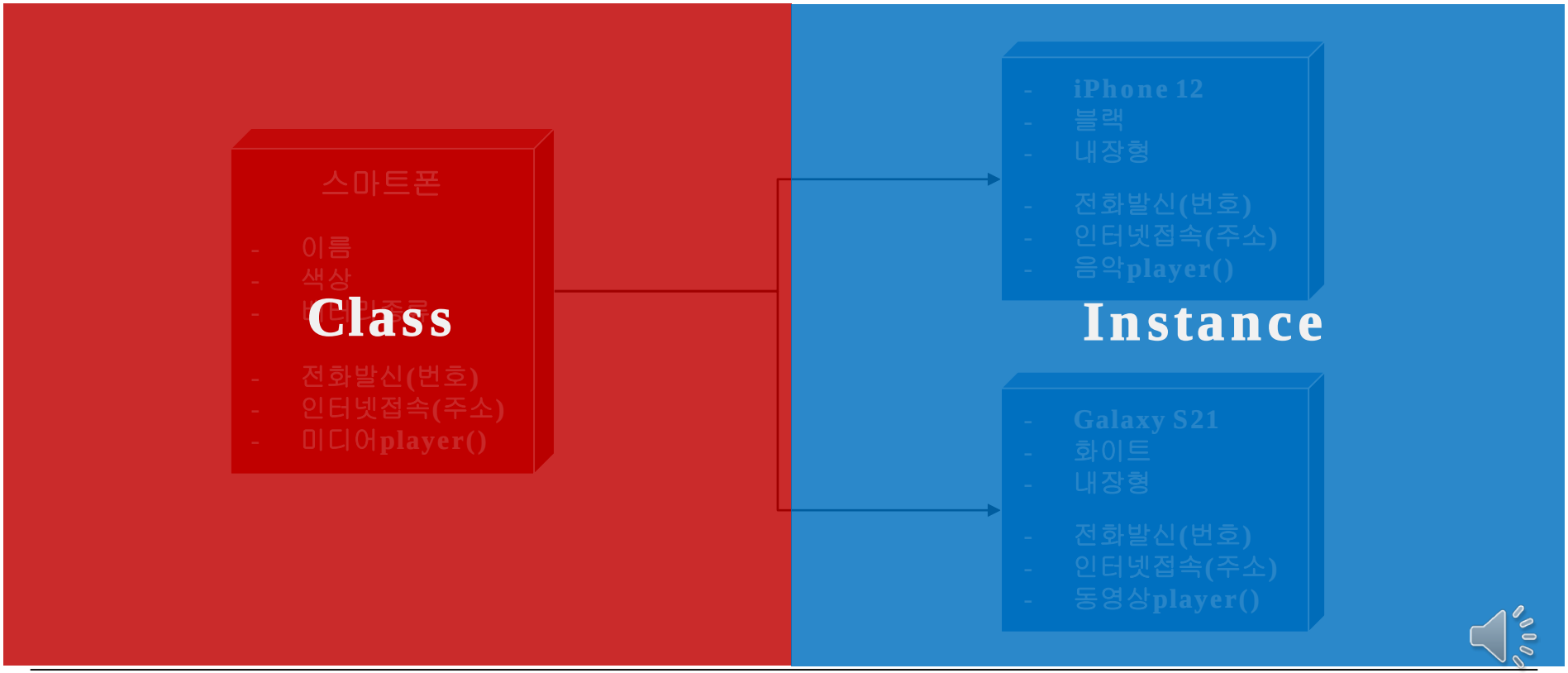
Instance - 물리적 실체



OOP – 클래스와 객체

- 객체지향프로그래밍 **(Object Oriented Programming)**

  클래스와 객체 **(Class and Instance)**



OOP – 클래스와 객체

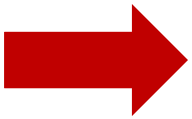
- 객체지향프로그래밍 **(Object Oriented Programming)**

클래스와 객체 **(Class and Instance)** - 클래스의 기본 구조

Class - 논리적 추상화

변수
- 클래스 변수
- 인스턴스 변수

메소드
- 생성자
- 일반 메소드
- 소멸자

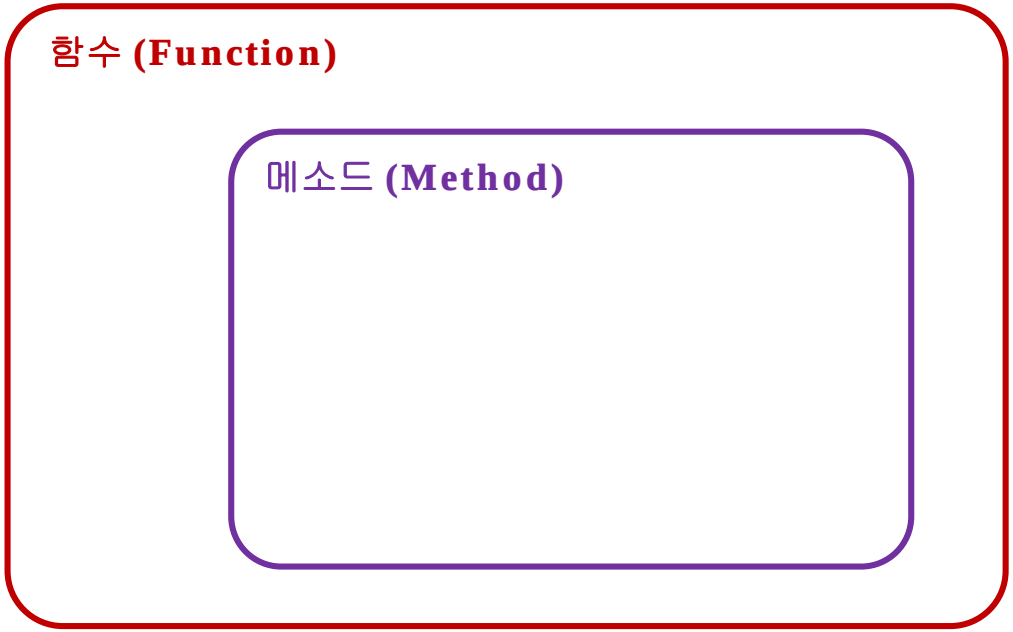


```
class 클래스이름:

  클래스변수

  def __init__(self):
    실행문
    ...

  def 메소드이름(self):
    실행문
    ...
    return data

  def __del__(self):
    실행문
    ...
```

OOP – 클래스와 객체

- 객체지향프로그래밍 **(Object Oriented Programming)**

  함수의 구조 – **c.f.)** 함수와 메소드



OOP – 클래스와 객체 – 예제

```
1   class test_class:
2
3       cls_var = 10
4
5       def __init__(self):
6           print("start.")
7           self.name = "test"
8           self.year = "2020"
9
10      def get_value(self):
11          return 3+5
12
13      def __del__(self):
14          print("end.")
15
16
17  a = test_class()
18
19  var2 = a.get_value()
20  name_var = a.name
21  year_var = a.year
22  a_var = a.cls_var
23  class_var = test_class.cls_var
24
25  print("var2      : ", var2)
26  print("name_var  : ", name_var)
27  print("year_var  : ", year_var)
28  print("a_var     : ", a_var)
29  print("class_var : ", class_var)
```

OOP - 예제

```
1  class test_class:
2
3      cls_var = 10
4
5      def __init__(self):
6          print("start.")
7          self.name = "test"
8          self.year = "2020"
9
10     def get_value(self):
11         return 3+5
12
13     def __del__(self):
14         print("end.")
15
16
17 a = test_class()
18
19 var2 = a.get_value()
20 name_var = a.name
21 year_var = a.year
22 a_var = a.cls_var
23 class_var = test_class.cls_var
24
25 print("var2      : ", var2)
26 print("name_var  : ", name_var)
27 print("year_var  : ", year_var)
28 print("a_var     : ", a_var)
29 print("class_var : ", class_var)
```

클래스 변수

생성자 — 객체가 생성될 때 실행

* 객체 생성시 초기화 작업

인스턴스 변수

메소드

소멸자 — 객체가 소멸될 때 실행

* 리소스 해제 등 종료 작업

test_class라는 이름의 객체(instance) a

'a.' 을 통해 객체 a의 변수 및 메소드에 접근

OOP – 클래스와 객체 – 예제

```
1   class test_class:
2
3       cls_var = 10
4
5       def __init__(self):
6           print("start.")
7           self.name = "test"
8           self.year = "2020"
9
10      def get_value(self):
11          return 3+5
12
13      def __del__(self):
14          print("end.")
15
16
17  a = test_class()
18
19  var2 = a.get_value()
20  name_var = a.name
21  year_var = a.year
22  a_var = a.cls_var
23  class_var = test_class.cls_var
24
25  print("var2      : ", var2)
26  print("name_var  : ", name_var)
27  print("year_var  : ", year_var)
28  print("a_var     : ", a_var)
29  print("class_var : ", class_var)
```

클래스 변수

생성자

인스턴스 변수

메소드

소멸자

test_class라는 이름의 객체(instance) a

'a.' 을 통해 객체 a의 변수 및 메소드에 접근

클래스변수: 여러 **instance** 사이에 공유된 값을 가지는 변수

인스턴스변수: 각각의 **instance**가 독립적으로 갖는 변수

OOP – 클래스와 객체 – 예제

```
1   class test_class:
2
3       cls_var = 10
4
5       def __init__(self):
6           print("start.")
7           self.name = "test"
8           self.year = "2020"
9
10      def get_value(self):
11          return 3+5
12
13      def __del__(self):
14          print("end.")
15
16
17  a = test_class()
18
19  var2 = a.get_value()
20  name_var = a.name
21  year_var = a.year
22  a_var = a.cls_var
23  class_var = test_class.cls_var
24
25  print("var2      : ", var2)
26  print("name_var  : ", name_var)
27  print("year_var  : ", year_var)
28  print("a_var     : ", a_var)
29  print("class_var : ", class_var)
```

실행결과

```
start.
var2      :  8
name_var  :  test
year_var  :  2020
a_var     :  10
class_var :  10
end.
```

OOP – 클래스와 객체

- 객체지향프로그래밍 **(Object Oriented Programming)**

  클래스와 객체 **(Class and Instance)** - 네임스페이스

  이름과 객체의 매핑관계를 포함하고 있는 공간

```
1   class test_class:
2
3       cls_var = 10
4
5       def __init__(self):
6           print("start.")
7           self.name = "test"
8           self.year = "2020"
9
10      def get_value(self):
11          return 3+5
12
13      def __del__(self):
14          print("end.")
```

test_class의 네임스페이스

{'cls_var': '10'}

클래스의 네임스페이스에
Dictionary 형태로 클래스 변수를 포함

c.f.) dictionary: {'key': 'value'}형태로
데이터를 저장, 관리하는 자료구조

c.f.) 파이썬에서 클래스는 실행 시 자체로
객체가 생성

OOP – 클래스와 객체

- 객체지향프로그래밍 **(Object Oriented Programming)**

  클래스와 객체 **(Class and Instance)** - 네임스페이스

  이름과 객체의 매핑관계를 포함하고 있는 공간

```
1   class test_class:
2
3       cls_var = 10
4
5       def __init__(self):
6           print("start.")
7           self.name = "test"
8           self.year = "2020"
9
10      def get_value(self):
11          return 3+5
12
13      def __del__(self):
14          print("end.")
15
16
17  a = test_class()
18  a.cls_var = 25
19
20  b = test_class()
```

test_class의 네임스페이스

{'cls_var': '10'}

a의 네임스페이스

{'cls_var': '25',
'name': 'test',
'year': '2020'}

b의 네임스페이스

{ 'name': 'test',
'year': '2020'}

OOP – 클래스와 객체

- 객체지향프로그래밍 **(Object Oriented Programming)**

  클래스와 객체 **(Class and Instance)** - 네임스페이스

  이름과 객체의 매핑관계를 포함하고 있는 공간

```
1   class test_class:
2
3       cls_var = 10
4
5       def __init__(self):
6           print("start.")
7           self.name = "test"
8           self.year = "2020"
9
10      def get_value(self):
11          return 3+5
12
13      def __del__(self):
14          print("end.")
15
16
17  a = test_class()
18  a.cls_var = 25
19
20  b = test_class()
21
22  print("test_class.cls_var  : ", test_class.cls_var)
23  print("a.cls_var           :", a.cls_var)
24  print("b.cls_var           :", b.cls_var)
```

test_class의 네임스페이스

{'cls_var': '10'}

a의 네임스페이스

{'cls_var': '25',
'name': 'test',
'year': '2020'}

b의 네임스페이스

{ 'name': 'test',
'year': '2020'}

OOP – 클래스와 객체

- 객체지향프로그래밍 **(Object Oriented Programming)**

  클래스와 객체 **(Class and Instance)** - 네임스페이스

  이름과 객체의 매핑관계를 포함하고 있는 공간

```
1   class test_class:
2
3       cls_var = 10
4
5       def __init__(self):
6           print("start.")
7           self.name = "test"
8           self.year = "2020"
9
10      def get_value(self):
11          return 3+5
12
13      def __del__(self):
14          print("end.")
15
16
17  a = test_class()
18  a.cls_var = 25
19
20  b = test_class()
21
22  print("test_class.cls_var  : ", test_class.cls_var)
23  print("a.cls_var           :", a.cls_var)
24  print("b.cls_var           :", b.cls_var)
```

실행결과

```
start.
start.
test_class.cls_var  :  10
a.cls_var           : 25
b.cls_var           : 10
end.
end.
```

test_class의 네임스페이스

{'cls_var': '10'}

a의 네임스페이스

{'cls_var': '25',
'name': 'test',
'year': '2020'}

b의 네임스페이스

{ 'name': 'test',
'year': '2020'}

OOP – 클래스와 객체

- 객체지향프로그래밍 **(Object Oriented Programming)**

클래스와 객체 **(Class and Instance)** - 네임스페이스

이름과 객체의 매핑관계를 포함하고 있는 공간

```
1   class test_class:
2
3       cls_var = 10
4
5       def __init__(self):
6           print("start.")
7           self.name = "test"
8           self.year = "2020"
9
10      def get_value(self):
11          return 3+5
12
13      def __del__(self):
14          print("end.")
15
16
17  a = test_class()
18  a.cls_var = 25
19
20  b = test_class()
21
22  print("test_class.cls_var  : ", test_class.cls_var)
23  print("a.cls_var           :", a.cls_var)
24  print("b.cls_var           :", b.cls_var)
```

실행결과

```
start.
start.
test_class.cls_var  :  10
a.cls_var           : 25
b.cls_var           : 10
end.
end.
```

a.cls_var는 인스탄스.변수이름 이렇게 되므로, a라는 인스탄스에 새로운 인스턴스변수 cls_var가 생긴것으로 간주됨

따라서 아래와 같은 코드를 실행하면, test_class의 클래스변수 cls_var는 1로 변하지만, 인스탄스 a에서 cls_var를 출력해보면 여전히 25로 나옴

```
test_class.cls_var = 1
print("test_class.cls_var : ", test_class.cls_var)
print("a.cls_var : ", a.cls_var)
print("b.cls_var : ", b.cls_var)
```

실행결과

```
test_class.cls_var :  1
a.cls_var :  25
b.cls_var :  1
```

가능하면 '인스탄스.클래스변수' 처럼 접근하는 것은 피해야 함

OOP – 클래스와 객체

- 객체지향프로그래밍 **(Object Oriented Programming)**

함수의 구조 – **self**

- 객체 자신을 참조하는 **parameter**
- 파이썬 메소드의 첫 **argument**로 항상 **instance**가 전달

```
1   class test_class:
2
3       cls_var = 10
4
5       def __init__(self):
6           self.name = "test"
7           self.year = "2020"
8
9       def get_value(self):
10          return 3+5
11
12      def __del__(self):
13          print()
14
15
16  a = test_class()
17  print("id(a): ", id(a))
```

실행결과

```
id(a):  4345277712
```

id(*object*)

객체의 《아이덴티티》를 돌려준다. 이것은 객체의 수명 동안 유일하고 바뀌지 않음이 보장되는 정수입니다. 수명이 겹치지 않는 두 개의 객체는 같은 `id()` 값을 가질 수 있습니다.

CPython implementation detail: This is the address of the object in memory.

참고링크 : https://docs.python.org/ko/3/library/functions.html#id

OOP – 클래스와 객체

- 객체지향프로그래밍 **(Object Oriented Programming)**

함수의 구조 – **self**

- 객체 자신을 참조하는 **parameter**
- 파이썬 메소드의 첫 **argument**로 항상 **instance**가 전달

```
1   class test_class:
2
3       cls_var = 10
4
5       def __init__(self):
6           self.name = "test"
7           self.year = "2020"
8
9       def get_value(self):
10          print("id(self) in get_value: ", id(self))
11          return 3+5
12
13      def __del__(self):
14          print()
15
16
17  a = test_class()
18  print("id(a):                    ", id(a))
19
20  result_value = a.get_value()
21  print("result_value: ", result_value)
```

실행결과

```
id(a):                    4433292560
id(self) in get_value:  4433292560
result_value:  8
```

OOP – 클래스와 객체 – 예제2

```python
class Computer:

    def __init__(self, ram, ssd, hdd,
                 cpu, name, color, category):
        self.__ram = ram
        self.__ssd = ssd
        self.hdd = hdd
        self.cpu = cpu
        self.name = name
        self.color = color
        self.category = category

    def connect_web(self, url):
        print(url, "주소로 연결되었습니다.")

    def sw_exe(self, sw):
        print(sw, "프로그램이 실행되었습니다.")


    def set_ram(self, ram):
        print(self.__ram, "의 (RAM)크기가 ", ram, "으로 변경.")
        self.__ram = ram

    def get_ram(self):
        return self.__ram

    def set_ssd(self, ssd):
        print(self.ssd, "의 (SSD)크기가 ", ssd, "으로 변경.")
        self.ssd = ssd

    def get_ssd(self):
        return self.ssd

    def __del__(self):
        print(self.name, "종료합니다.")
```

```python
iMac = Computer('64G', '2T', '3T',
                'i7', 'iMac20', 'silver', 'desktop')
iMac.connect_web('www.youtube.com')
iMac.sw_exe('chrome')
iMac.set_ram('128G')
ram_iMac = iMac.get_ram()
print(ram_iMac)
print(iMac.get_ram())
```

OOP – 클래스와 객체 – 예제2

```
1  class Computer:
2
3      def __init__(self, ram, ssd, hdd,
4                   cpu, name, color, category):
5          self.__ram = ram
6          self.__ssd = ssd
7          self.hdd = hdd
8          self.cpu = cpu
9          self.name = name
10         self.color = color
11         self.category = category
12
13     def connect_web(self, url):
14         print(url, "주소로 연결되었습니다.")
15
16     def sw_exe(self, sw):
17         print(sw, "프로그램이 실행되었습니다.")
18
19
20     def set_ram(self, ram):
21         print(self.__ram, "의 (RAM)크기가 ", ram, "으로 변경.")
22         self.__ram = ram
23
24     def get_ram(self):
25         return self.__ram
26
27     def set_ssd(self, ssd):
28         print(self.ssd, "의 (SSD)크기가 ", ssd, "으로 변경.")
29         self.ssd = ssd
30
31     def get_ssd(self):
32         return self.ssd
33
34     def __del__(self):
35         print(self.name, "종료합니다.")
```

생성자

소멸자

OOP – 클래스와 객체 – 예제2

```
1  class Computer:
2
3      def __init__(self, ram, ssd, hdd,
4                   cpu, name, color, category):
5          self.__ram = ram
6          self.__ssd = ssd
7          self.hdd = hdd
8          self.cpu = cpu
9          self.name = name
10         self.color = color
11         self.category = category
12
13     def connect_web(self, url):
14         print(url, "주소로 연결되었습니다.")
15
16     def sw_exe(self, sw):
17         print(sw, "프로그램이 실행되었습니다.")
18
19
20     def set_ram(self, ram):
21         print(self.__ram, "의 (RAM)크기가 ", ram, "으로 변경.")
22         self.__ram = ram
23
24     def get_ram(self):
25         return self.__ram
26
27     def set_ssd(self, ssd):
28         print(self.ssd, "의 (SSD)크기가 ", ssd, "으로 변경.")
29         self.ssd = ssd
30
31     def get_ssd(self):
32         return self.ssd
33
34     def __del__(self):
35         print(self.name, "종료합니다.")
```

생성자

```
38 iMac = Computer('64G', '2T', '3T',
39                 'i7', 'iMac20', 'silver', 'desktop')
40 iMac.connect_web('www.youtube.com')
41 iMac.sw_exe('chrome')
42 iMac.set_ram('128G')
43 ram_iMac = iMac.get_ram()
44 print(ram_iMac)
45 print(iMac.get_ram())
```

```
www.youtube.com 주소로 연결되었습니다.
chrome 프로그램이 실행되었습니다.
64G 의 (RAM)크기가  128G 으로 변경.
128G
128G
iMac20 종료합니다.
```

실행결과

소멸자

OOP – 추상화

- 객체지향프로그래밍 **(Object Oriented Programming)**

추상화 **(Abstraction)** - 객체들의 공통 속성과 행위를 추출하는 것

```
1   class Computer:
2
3       def __init__(self, ram, ssd, hdd,
4                    cpu, name, color, category):
5           self.__ram = ram
6           self.__ssd = ssd
7           self.hdd = hdd
8           self.cpu = cpu
9           self.name = name
10          self.color = color
11          self.category = category
12
13      def connect_web(self, url):
14          print(url, "주소로 연결되었습니다.")
15
16      def sw_exe(self, sw):
17          print(sw, "프로그램이 실행되었습니다.")
18
19
20      def set_ram(self, ram):
21          print(self.__ram, "의 (RAM)크기가 ", ram, "으로 변경.")
22          self.__ram = ram
23
24      def get_ram(self):
25          return self.__ram
26
27      def set_ssd(self, ssd):
28          print(self.ssd, "의 (SSD)크기가 ", ssd, "으로 변경.")
29          self.ssd = ssd
30
31      def get_ssd(self):
32          return self.ssd
33
34      def __del__(self):
35          print(self.name, "종료합니다.")
```

컴퓨터가 같는 공통 속성

컴퓨터가 같는 공통 행위

OOP – 캡슐화

- 객체지향프로그래밍 **(Object Oriented Programming)**

캡슐화 **(Encapsulation)**
- 속성과 행위를 목적에 적합하게 묶어 독립된 단위로 구성
- 정보은닉 **(information hiding):** 속성과 행위에 대한 접근 제한

- 결합도 **(Coupling)** – 어떤 기능 (메소드)이 다른 기능 (메소드)에 의존하는 정도
- 응집도 **(Cohesion)** – 기능 (메소드) 내부 요소들의 관련 정도

  ⟹ 따라서, 낮은 결합도와 높은 응집도를 고려하여 구성하는 것이 중요

- 정보은닉 – 접근지정자를 이용하여 외부 접근의 접근성 설정

OOP – 캡슐화

- 객체지향프로그래밍 **(Object Oriented Programming)**

캡슐화 (Encapsulation)
- 속성과 행위를 목적에 적합하게 묶어 독립된 단위로 구성
- 정보은닉 **(information hiding):** 속성과 행위에 대한 접근 제한

정보은닉 - 접근지정자

사용예	타입	의미
name	public	정의된 클래스 내부 뿐만 아니라 외부에서도 사용 가능
_name	protected	상속 클래스 및 클래스 내부에서 사용 가능
__name	private	클래스 내부에서만 사용 가능

OOP – 캡슐화

- 객체지향프로그래밍 **(Object Oriented Programming)**

캡슐화 **(Encapsulation)** - 속성과 행위를 목적에 적합하게 묶어 독립된 단위로 구성
- 정보은닉 **(information hiding):** 속성과 행위에 대한 접근 제한

```
1  class Computer:
2
3      def __init__(self, ram, ssd, hdd,
4                   cpu, name, color, category):
5          self.__ram = ram
6          self.__ssd = ssd
7          self.hdd = hdd
8          self.cpu = cpu
9          self.name = name
10         self.color = color
11         self.category = category
12
13     def connect_web(self, url):
14         print(url, "주소로 연결되었습니다.")
15
16     def sw_exe(self, sw):
17         print(sw, "프로그램이 실행되었습니다.")
18
19
20     def set_ram(self, ram):
21         print(self.__ram, "의 (RAM)크기가 ", ram, "으로 변경.")
22         self.__ram = ram
23
24     def get_ram(self):
25         return self.__ram
26
27     def set_ssd(self, ssd):
28         print(self.ssd, "의 (SSD)크기가 ", ssd, "으로 변경.")
29         self.ssd = ssd
30
31     def get_ssd(self):
32         return self.ssd
33
34     def __del__(self):
35         print(self.name, "종료합니다.")
```

private

사용예	타입	의미
name	public	정의된 클래스 내부 뿐만 아니라 외부에서도 사용 가능
_name	protected	상속 클래스 및 클래스 내부에서 사용 가능
__name	private	클래스 내부에서만 사용 가능

```
iMac.
  get_ram(self)                Computer
  name                         Computer
  set_ram(self, ram)           Computer
  sw_exe(self, sw)             Computer
  connect_web(self, url)       Computer
  ssd                          Computer
  category                     Computer
  color                        Computer
  cpu                          Computer
  get_ssd(self)                Computer
  hdd                          Computer
Press ^. to choose the selected (or first) suggestion and insert a dot afterwards >>
```

* setter와 getter를 통해 접근

OOP – 캡슐화

- 객체지향프로그래밍 **(Object Oriented Programming)**

캡슐화 **(Encapsulation)** - 속성과 행위를 목적에 적합하게 묶어 독립된 단위로 구성
- 정보은닉 **(information hiding):** 속성과 행위에 대한 접근 제한

```
1  class Computer:
2
3      def __init__(self, ram, ssd, hdd,
4                   cpu, name, color, category):
5          self.__ram = ram
6          self.__ssd = ssd
7          self.hdd = hdd
8          self.cpu = cpu
9          self.name = name
10         self.color = color
11         self.category = category
12
13     def connect_web(self, url):
14         print(url, "주소로 연결되었습니다.")
15
16     def sw_exe(self, sw):
17         print(sw, "프로그램이 실행되었습니다.")
18
19
20     def set_ram(self, ram):
21         print(self.__ram, "의 (RAM)크기가 ", ram, "으로 변경.")
22         self.__ram = ram
23
24     def get_ram(self):
25         return self.__ram
26
27     def set_ssd(self, ssd):
28         print(self.ssd, "의 (SSD)크기가 ", ssd, "으로 변경.")
29         self.ssd = ssd
30
31     def get_ssd(self):
32         return self.ssd
33
34     def __del__(self):
35         print(self.name, "종료합니다.")
```

private

*** Python의 setter(@.setter)와 getter(@property)**

```
1  class Computer:
2
3      def __init__(self, ram, ssd, hdd,
4                   cpu, name, color, category):
5          self.__ram = ram
6          self.__ssd = ssd
7          self.hdd = hdd
8          self.cpu = cpu
9          self.name = name
10         self.color = color
11         self.category = category
12
13     def connect_web(self, url):
14         print(url, "주소로 연결되었습니다.")
15
16
17     def sw_exe(self, sw):
18         print(sw, "프로그램이 실행되었습니다.")
19
20     @property
21     def ram(self):
22         return self.__ram
23
24     @ram.setter
25     def ram(self, ram):
26         print(self.__ram, "의 (RAM)크기가 ", ram, "으로 변경.")
27         self.__ram = ram
28
29     @property
30     def ssd(self):
31         return self.__ssd
32
33     @ssd.setter
34     def ssd(self, ssd):
35         print(self.__ssd, "의 (SSD)크기가 ", ssd, "으로 변경.")
36         self.__ssd = ssd
37
38     def __del__(self):
39         print(self.name, "종료합니다.")
```

OOP – 캡슐화

- 객체지향프로그래밍 **(Object Oriented Programming)**

캡슐화 **(Encapsulation)**
- 속성과 행위를 목적에 적합하게 묶어 독립된 단위로 구성
- 정보은닉 **(information hiding):** 속성과 행위에 대한 접근 제한

*** Python의 setter(@.setter)와 getter(@property)**

```
1  class Computer:
2
3      def __init__(self, ram, ssd, hdd,
4                   cpu, name, color, category):
5          self.__ram = ram
6          self.__ssd = ssd
7          self.hdd = hdd
8          self.cpu = cpu
9          self.name = name
10         self.color = color
11         self.category = category
12
13     def connect_web(self, url):
14         print(url, "주소로 연결되었습니다.")
15
16
17     def sw_exe(self, sw):
18         print(sw, "프로그램이 실행되었습니다.")
19
20     @property
21     def ram(self):
22         return self.__ram
23
24     @ram.setter
25     def ram(self, ram):
26         print(self.__ram, "의 (RAM)크기가 ", ram, "으로 변경.")
27         self.__ram = ram
28
29     @property
30     def ssd(self):
31         return self.__ssd
32
33     @ssd.setter
34     def ssd(self, ssd):
35         print(self.__ssd, "의 (SSD)크기가 ", ssd, "으로 변경.")
36         self.__ssd = ssd
37
38     def __del__(self):
39         print(self.name, "종료합니다.")
```

@property 데코레이터와 @메소드이름.setter를 사용하면 메소드를 속성처럼 바로 사용할 수 있음

```
42  iMac = Computer('64G', '2T', '3T',
43                  'i7', 'iMac20', 'silver', 'desktop')
44
45  iMac.ram = '128G'
46  print("iMac.ram getter: ", iMac.ram)
47
48  iMac.ssd = '10T'
49  print("iMac.ssd getter: ", iMac.ssd)
```

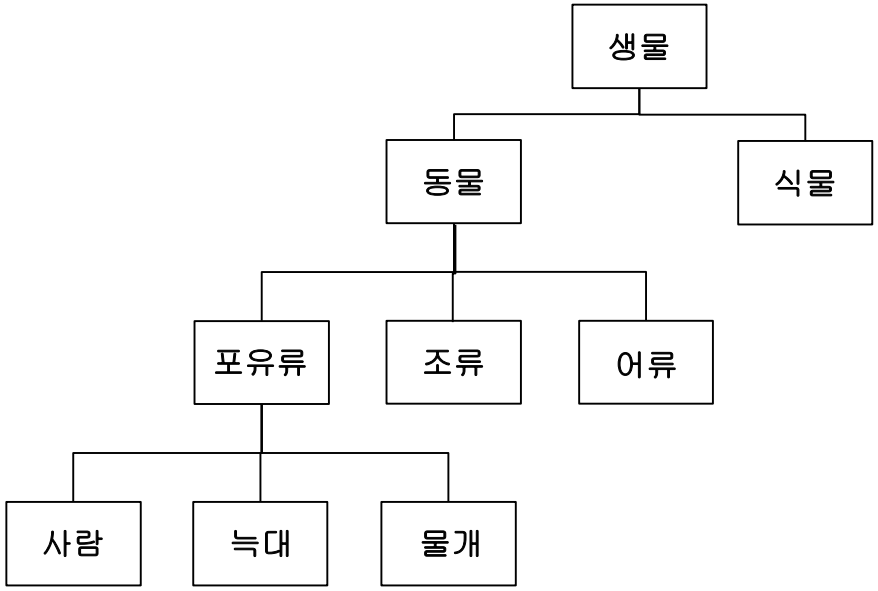
```
64G 의 (RAM)크기가  128G 으로 변경.
iMac.ram getter:  128G
2T 의 (SSD)크기가  10T 으로 변경.
iMac.ssd getter:  10T
iMac20 종료합니다.
```

실행결과

OOP – 상속성

- 객체지향프로그래밍 **(Object Oriented Programming)**

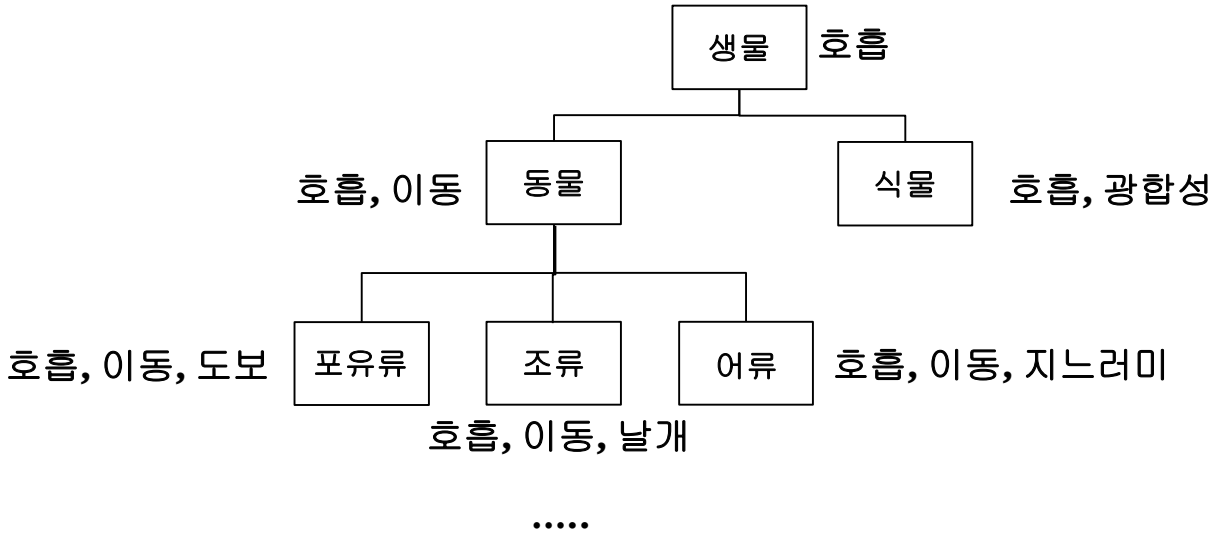
  상속성 **(Inheritance)** - 자식 클래스가 부모클래스의 특성과 기능을 물려받는 것



OOP – 상속성

- 객체지향프로그래밍 **(Object Oriented Programming)**

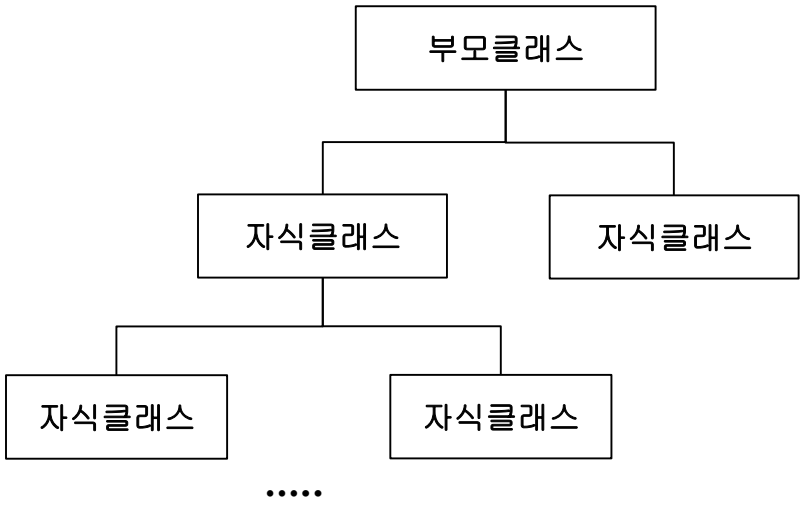
  상속성 **(Inheritance)** - 자식 클래스가 부모클래스의 특성과 기능을 물려받는 것



OOP – 상속성

- 객체지향프로그래밍 **(Object Oriented Programming)**

  상속성 **(Inheritance)** - 자식 클래스가 부모클래스의 특성과 기능을 물려받는 것

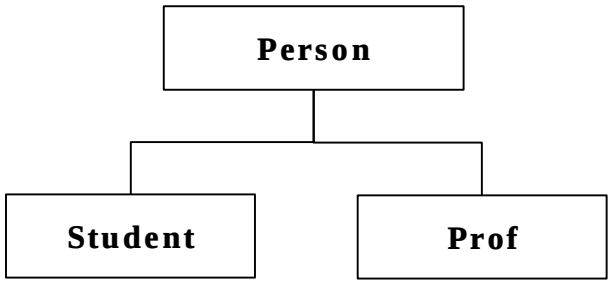



```python
class Person:
    def thinking(self):
        print("생각 중입니다.")

class Student(Person):
    def study(self):
        print("공부 중입니다.")

class Prof(Person):
    def lecture(self):
        print("강의 중입니다.")

person_p = Person()
student_s = Student()
prof_p = Prof()

print("1. call person_p.thinking()")
person_p.thinking()
print("2. call student_s.thinking()")
student_s.thinking()
print("3. call student_s.study()")
student_s.study()
print("4. call prof_p.lecture()")
prof_p.lecture()
```

OOP – 상속성

- 객체지향프로그래밍 **(Object Oriented Programming)**

  상속성 **(Inheritance)** - 자식 클래스가 부모클래스의 특성과 기능을 물려받는 것

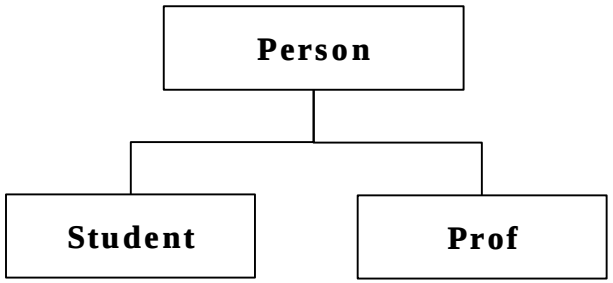



```python
class Person:
    def thinking(self):
        print("생각 중입니다.")

class Student(Person):
    def study(self):
        print("공부 중입니다.")

class Prof(Person):
    def lecture(self):
        print("강의 중입니다.")

person_p = Person()
student_s = Student()
prof_p = Prof()

print("1. call person_p.thinking()")
person_p.thinking()
print("2. call student_s.thinking()")
student_s.thinking()
print("3. call student_s.study()")
student_s.study()
print("4. call prof_p.lecture()")
prof_p.lecture()
```

OOP – 상속성

- 객체지향프로그래밍 **(Object Oriented Programming)**

상속성 **(Inheritance)** - 자식 클래스가 부모클래스의 특성과 기능을 물려받는 것



실행결과

```
1. call person_p.thinking()
생각 중입니다.
2. call student_s.thinking()
생각 중입니다.
3. call student_s.study()
공부 중입니다.
4. call prof_p.lecture()
강의 중입니다.
```

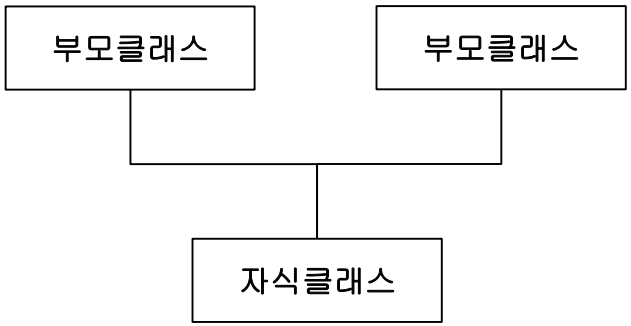
```python
class Person:
    def thinking(self):
        print("생각 중입니다.")

class Student(Person):
    def study(self):
        print("공부 중입니다.")

class Prof(Person):
    def lecture(self):
        print("강의 중입니다.")

person_p = Person()
student_s = Student()
prof_p = Prof()

print("1. call person_p.thinking()")
person_p.thinking()
print("2. call student_s.thinking()")
student_s.thinking()
print("3. call student_s.study()")
student_s.study()
print("4. call prof_p.lecture()")
prof_p.lecture()
```

OOP – 상속성

- 객체지향프로그래밍 **(Object Oriented Programming)**

  상속성 **(Inheritance)** - 자식 클래스가 부모클래스의 특성과 기능을 물려받는 것

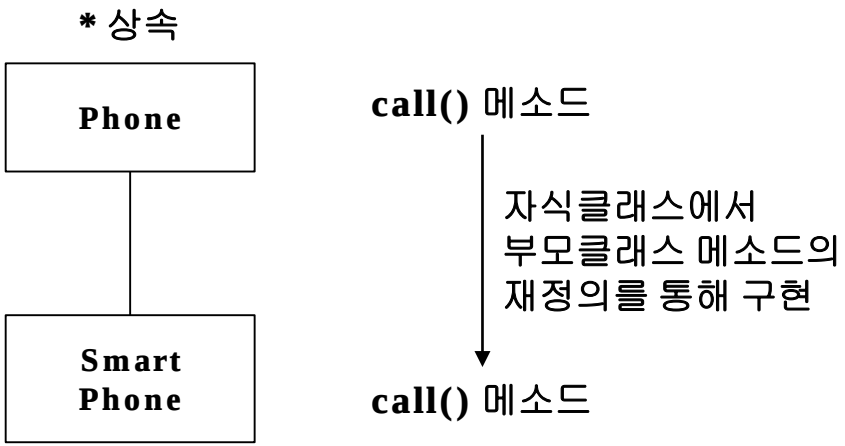
* 다중상속



OOP – 다형성

- 객체지향프로그래밍 **(Object Oriented Programming)**

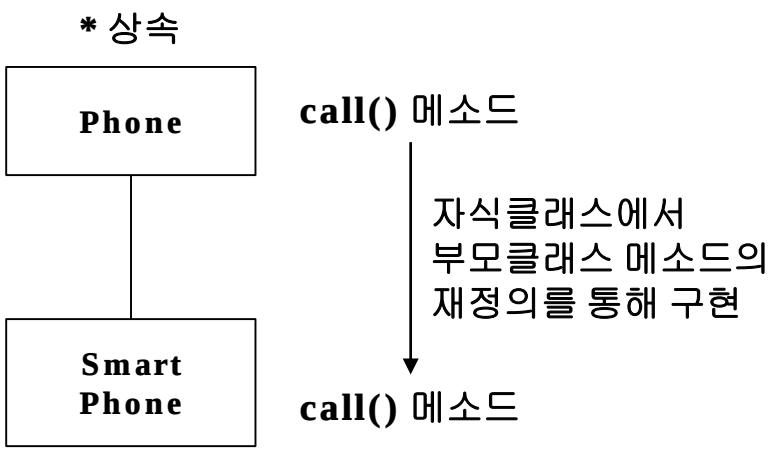
  다형성 **(Polymorphism)** - 같은 메소드 이름으로 다른 동작을 하는 것



OOP – 다형성

- 객체지향프로그래밍 **(Object Oriented Programming)**

  다형성 **(Polymorphism)** - 같은 메소드 이름으로 다른 동작을 하는 것



Method Override

```
1  class Phone:
2      def call(self):
3          print("통화 중입니다.")
4
5  class InternetCmt:
6      def connect(self):
7          print("인터넷 사용 중입니다.")
8
9  class SmartPhone(Phone, InternetCmt):
10     def call(self):
11         print("smart 통화 중입니다.")
12     def slide(self):
13         print("밀어서 잠금해제 중입니다.")
14     def app_exe(self):
15         print("Application이 실행 중입니다.")
```

OOP – 다형성

- 객체지향프로그래밍 **(Object Oriented Programming)**

  다형성 **(Polymorphism)** - 같은 메소드 이름으로 다른 동작을 하는 것

Method Override

```
1  class Phone:
2      def call(self):
3          print("통화 중입니다.")
4
5  class InternetCmt:
6      def connect(self):
7          print("인터넷 사용 중입니다.")
8
9  class SmartPhone(Phone, InternetCmt):
10     def call(self):
11         print("smart 통화 중입니다.")
12     def slide(self):
13         print("밀어서 잠금해제 중입니다.")
14     def app_exe(self):
15         print("Application이 실행 중입니다.")
```

```
17 city_phone = Phone()
18 print("1. call call(): class Phone")
19 city_phone.call()
20
21 yPhone = SmartPhone()
22 print("2. call call(): class SmartPhone")
23 yPhone.call()
```

실행결과

```
1. call call(): class Phone
통화 중입니다.
2. call call(): class SmartPhone
smart 통화 중입니다.
```

OOP – 클래스와 객체

- 객체지향프로그래밍 **(Object Oriented Programming)**

  - ✓ 추상화 **(Abstraction)** - 객체들의 공통 속성과 행위를 추출하는 것
  - ✓ 캡슐화 **(Encapsulation)** - 속성과 행위를 목적에 적합하게 묶어 독립된 단위로 구성
    - 정보은닉 **(information hiding):** 속성과 행위에 대한 접근 제한
  - ✓ 상속성 **(Inheritance)** - 자식 클래스가 부모클래스의 특성과 기능을 물려받는 것
  - ✓ 다형성 **(Polymorphism)** - 같은 메소드 이름으로 다른 동작을 하는 것

Outline

- 함수
  - 함수란
  - 함수의 구조
  - 함수 내 변수
- 객체지향프로그래밍
  - 객체지향프로그래밍 (OOP)
  - 클래스와 객체 (Class and Instance)
  - 추상화 (Abstraction)
  - 캡슐화 (Encapsulation)
  - 상속성 (Inheritance)
  - 다형성 (Polymorphism)
- 모듈

모듈이란

- **모듈 (Module)**

  함수, 변수, 클래스를 모아놓은 파이썬 파일 – 유사한 기능의 코드들을 작성한 파일

모듈이란

- **모듈 (Module) – 예제1**

모듈 사용 명령어 – **import** 모듈이름 → 모듈의 모든 기능을 사용할 수 있음
from 모듈이름 **import** 기능 → 해당 함수(기능)만 **import** 하여 사용하겠다는 의미
as → **alias,** 즉 내가 이름을 지정해서 사용하겠다.

```
import math

print("1. math.pow(2, 3): ", math.pow(2, 3))
print("2. math.pow(5, 2): ", math.pow(5, 2))
print("3. math.pow(10, 2): ", math.pow(10, 2))

print("4. type(math.pow(2, 3)): ", type(math.pow(2, 3)))

print("5. math.factorial(5): ", math.factorial(5))
print("6. math.factorial(8): ", math.factorial(8))

print("7. type(math.factorial(5)): ", type(math.factorial(5)))

print("8. math.sqrt(4)", math.sqrt(4))

print("9. math.log(8, 2): ", math.log(8, 2))
print("10. math.log(32, 2): ", math.log(32, 2))

print("11. math.pi: ", math.pi)
print("12. math.e: ", math.e)
```

```
import math
print(math.pow(2,3))

import math as mh
print(mh.pow(2, 3))

from math import pow as pw
print(pw(2, 3))
```

실행결과

```
1. math.pow(2, 3):  8.0
2. math.pow(5, 2):  25.0
3. math.pow(10, 2):  100.0
4. type(math.pow(2, 3)):  <class 'float'>
5. math.factorial(5):  120
6. math.factorial(8):  40320
7. type(math.factorial(5)):  <class 'int'>
8. math.sqrt(4) 2.0
9. math.log(8, 2):  3.0
10. math.log(32, 2):  5.0
11. math.pi:  3.141592653589793
12. math.e:  2.718281828459045
```

모듈이란

- **모듈 (Module) – 예제2**

  모듈 사용 명령어 – **import** 모듈이름, **from** 모듈이름 **import** 기능, **as**

```
1   import numpy
2
3   m1 = numpy.matrix(([1, 2, 3], [4, 5, 6], [7, 8, 9]))
4   m2 = numpy.matrix(([7, 8, 9], [4, 5, 6], [1, 2, 3]))
5
6   print("1. m1: ")
7   print(m1)
8   print("2. m2: ")
9   print(m1)
10  print("3. m1 + m2: ")
11  print(m1 + m2)
12  print("4. m1 * m2: ")
13  print(m1 * m2)
14  print("5. type(m1): ", type(m1))
```

실행결과

```
1. m1:
[[1 2 3]
 [4 5 6]
 [7 8 9]]
2. m2:
[[1 2 3]
 [4 5 6]
 [7 8 9]]
3. m1 + m2:
[[ 8 10 12]
 [ 8 10 12]
 [ 8 10 12]]
4. m1 * m2:
[[ 18  24  30]
 [ 54  69  84]
 [ 90 114 138]]
5. type(m1):  <class 'numpy.matrix'>
```

모듈이란

- **모듈 (Module) – 예제3**

  모듈 사용 명령어 – **import** 모듈이름, **from** 모듈이름 **import** 기능, **as**

```
1   import datetime
2   import time
3   import random
4
5   now = datetime.datetime.now()
6   print("1. now: ", now)
7
8   print("2. time.time(): ", time.time())
9   print("3. time.gmtime().tm_year: ", time.gmtime().tm_year)
10  print()
11  start_time = time.time()
12
13  cnt = 0
14  while(True):
15      cnt += 1
16      rand_num = random.randrange(1, 100)
17      if(rand_num == 83):
18          break
19
20  end_time = time.time()
21  print("4. cnt: ", cnt)
22  print("5. exe: ", end_time - start_time, "sec")
```

실행결과

```
1. now:  2020-04-12 04:29:52.401269
2. time.time():  1586633392.401323
3. time.gmtime().tm_year:  2020

4. cnt:  85
5. exe:  0.0001392364501953125 sec
```

모듈이란

- **모듈 (Module) – 예제4**

  모듈 사용 명령어 – **import** 모듈이름, **from** 모듈이름 **import** 기능, **as**

  from 모듈이름 **import** 기능 (함수) – 모듈이름 생략가능

```
1   import numpy
2
3   m1 = numpy.matrix(([1, 2, 3], [4, 5, 6], [7, 8, 9]))
4   m2 = numpy.matrix(([7, 8, 9], [4, 5, 6], [1, 2, 3]))
5
6   print("1. m1: ")
7   print(m1)
8   print("2. m2: ")
9   print(m1)
10  print("3. m1 + m2: ")
11  print(m1 + m2)
12  print("4. m1 * m2: ")
13  print(m1 * m2)
14  print("5. type(m1): ", type(m1))
```

```
1   from numpy import matrix
2
3   m1 = matrix(([1, 2, 3], [4, 5, 6], [7, 8, 9]))
4   m2 = matrix(([7, 8, 9], [4, 5, 6], [1, 2, 3]))
5
6   print("1. m1: ")
7   print(m1)
8   print("2. m2: ")
9   print(m1)
10  print("3. m1 + m2: ")
11  print(m1 + m2)
12  print("4. m1 * m2: ")
13  print(m1 * m2)
14  print("5. type(m1): ", type(m1))
```

모듈이란

- **모듈 (Module) – 예제5**

  모듈 사용 명령어 – **import** 모듈이름, **from** 모듈이름 **import** 기능, **as**

  from 모듈이름 **import** * – 모듈이름 생략가능

```
1   import numpy
2
3   m1 = numpy.matrix(([1, 2, 3], [4, 5, 6], [7, 8, 9]))
4   m2 = numpy.matrix(([7, 8, 9], [4, 5, 6], [1, 2, 3]))
5
6   print("1. m1: ")
7   print(m1)
8   print("2. m2: ")
9   print(m1)
10  print("3. m1 + m2: ")
11  print(m1 + m2)
12  print("4. m1 * m2: ")
13  print(m1 * m2)
14  print("5. type(m1): ", type(m1))
```

```
1   from numpy import *
2
3   m1 = matrix(([1, 2, 3], [4, 5, 6], [7, 8, 9]))
4   m2 = matrix(([7, 8, 9], [4, 5, 6], [1, 2, 3]))
5
6   print("1. m1: ")
7   print(m1)
8   print("2. m2: ")
9   print(m1)
10  print("3. m1 + m2: ")
11  print(m1 + m2)
12  print("4. m1 * m2: ")
13  print(m1 * m2)
14  print("5. type(m1): ", type(m1))
```

모듈이란

- **모듈 (Module) – 예제6**

  모듈 사용 명령어 – **import** 모듈이름, **from** 모듈이름 **import** 기능, **as**

```python
import datetime
import time
import random

now = datetime.datetime.now()
print("1. now: ", now)

print("2. time.time(): ", time.time())
print("3. time.gmtime().tm_year: ", time.gmtime().tm_year)
print()
start_time = time.time()

cnt = 0
while(True):
    cnt += 1
    rand_num = random.randrange(1, 100)
    if(rand_num == 83):
        break

end_time = time.time()
print("4. cnt: ", cnt)
print("5. exe: ", end_time - start_time, "sec")
```

```python
from datetime import datetime
from time import *
import random

now = datetime.now()
print("1. now: ", now)

print("2. time.time(): ", time())
print("3. time.gmtime().tm_year: ", gmtime().tm_year)
print()
start_time = time()

cnt = 0
while(True):
    cnt += 1
    rand_num = random.randrange(1, 100)
    if(rand_num == 83):
        break

end_time = time()
print("4. cnt: ", cnt)
print("5. exe: ", end_time - start_time, "sec")
```

모듈이란

- **모듈 (Module) – 예제7**

  모듈 사용 명령어 – **import** 모듈이름, **from** 모듈이름 **import** 기능, **as**

```
1   import numpy
2
3   m1 = numpy.matrix(([1, 2, 3], [4, 5, 6], [7, 8, 9]))
4   m2 = numpy.matrix(([7, 8, 9], [4, 5, 6], [1, 2, 3]))
5
6   print("1. m1: ")
7   print(m1)
8   print("2. m2: ")
9   print(m1)
10  print("3. m1 + m2: ")
11  print(m1 + m2)
12  print("4. m1 * m2: ")
13  print(m1 * m2)
14  print("5. type(m1): ", type(m1))
```

```
1   import numpy as np
2
3   m1 = np.matrix(([1, 2, 3], [4, 5, 6], [7, 8, 9]))
4   m2 = np.matrix(([7, 8, 9], [4, 5, 6], [1, 2, 3]))
5
6   print("1. m1: ")
7   print(m1)
8   print("2. m2: ")
9   print(m1)
10  print("3. m1 + m2: ")
11  print(m1 + m2)
12  print("4. m1 * m2: ")
13  print(m1 * m2)
14  print("5. type(m1): ", type(m1))
```

모듈이란

- **모듈 (Module) – 예제8**

  모듈 사용 명령어 – **import** 모듈이름, **from** 모듈이름 **import** 기능, **as**

```
1   from numpy import matrix
2
3   m1 = matrix(([1, 2, 3], [4, 5, 6], [7, 8, 9]))
4   m2 = matrix(([7, 8, 9], [4, 5, 6], [1, 2, 3]))
5
6   print("1. m1: ")
7   print(m1)
8   print("2. m2: ")
9   print(m1)
10  print("3. m1 + m2: ")
11  print(m1 + m2)
12  print("4. m1 * m2: ")
13  print(m1 * m2)
14  print("5. type(m1): ", type(m1))
```

```
1   from numpy import matrix as mt
2
3   m1 = mt(([1, 2, 3], [4, 5, 6], [7, 8, 9]))
4   m2 = mt(([7, 8, 9], [4, 5, 6], [1, 2, 3]))
5
6   print("1. m1: ")
7   print(m1)
8   print("2. m2: ")
9   print(m1)
10  print("3. m1 + m2: ")
11  print(m1 + m2)
12  print("4. m1 * m2: ")
13  print(m1 * m2)
14  print("5. type(m1): ", type(m1))
```

퀴즈 (O|X)

1. 파이썬 함수는 파라미터와 리턴이 항상 있어야 한다.

2. 클래스는 속성과 행위를 변수와 메소드로 정의한 논리적 개념이고, 인스탄스는 클래스 정의를 기반으로 실제 메모리에 할당된 물리적 실체이다.