PYTHON PROGRAMMING

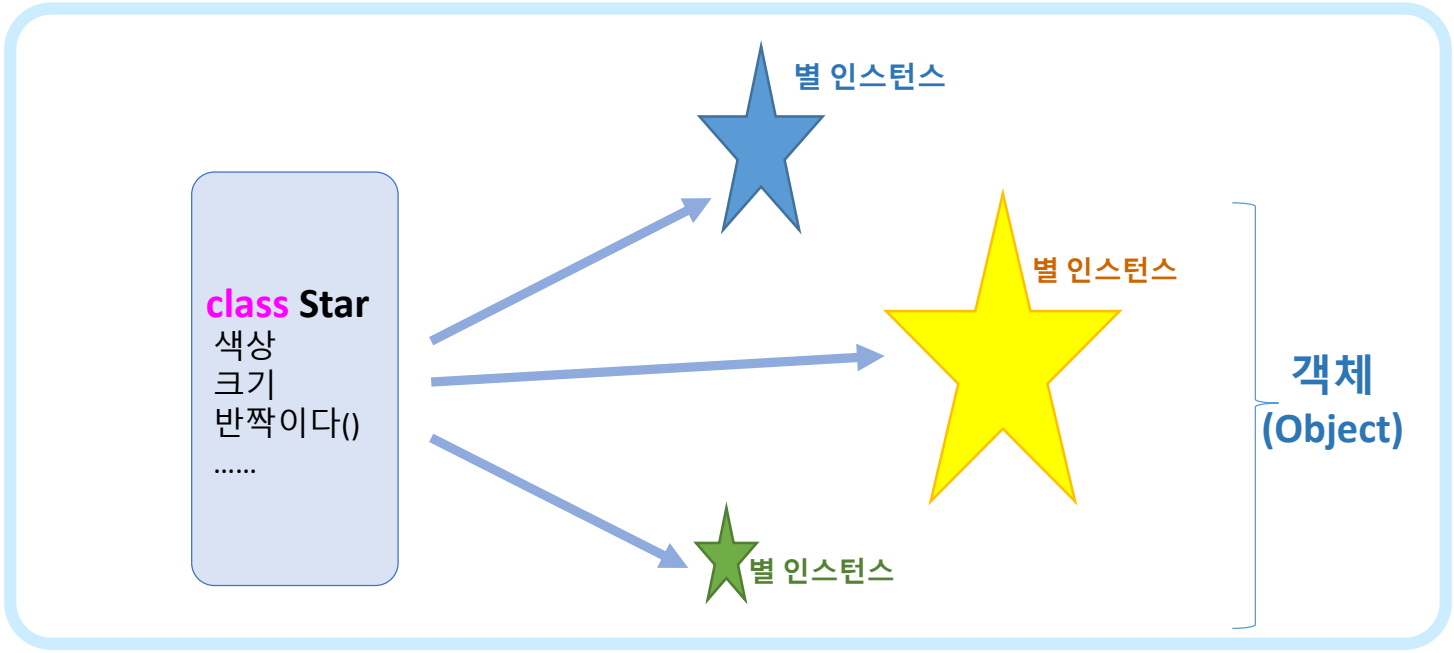
OBJECT ORIENTED PROGRAMMING

PYTHON 클래스

PYTHON 클래스

◆ 클래스(CLASS)

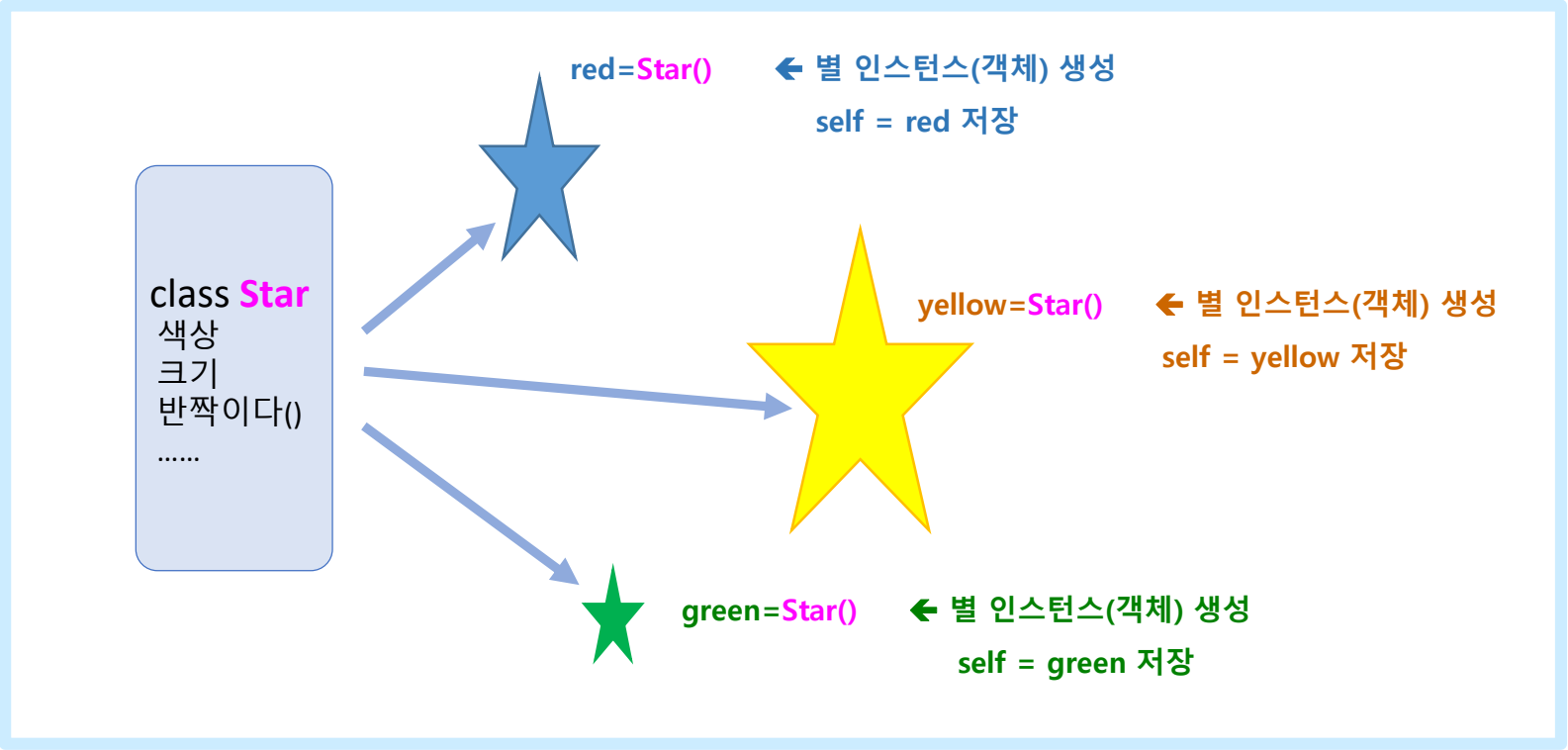
- 특정 기능을 하기 위한 **변수와 함수를 하나로 묶어 둔 Type**
- 제품의 **설계도**에 비유 / **틀**
- 설계도에 근거해서 만들어진 제품이 바로 **객체**
- 제품이 생성된 상세 클래스 정보를 **인스턴스**



PYTHON 클래스

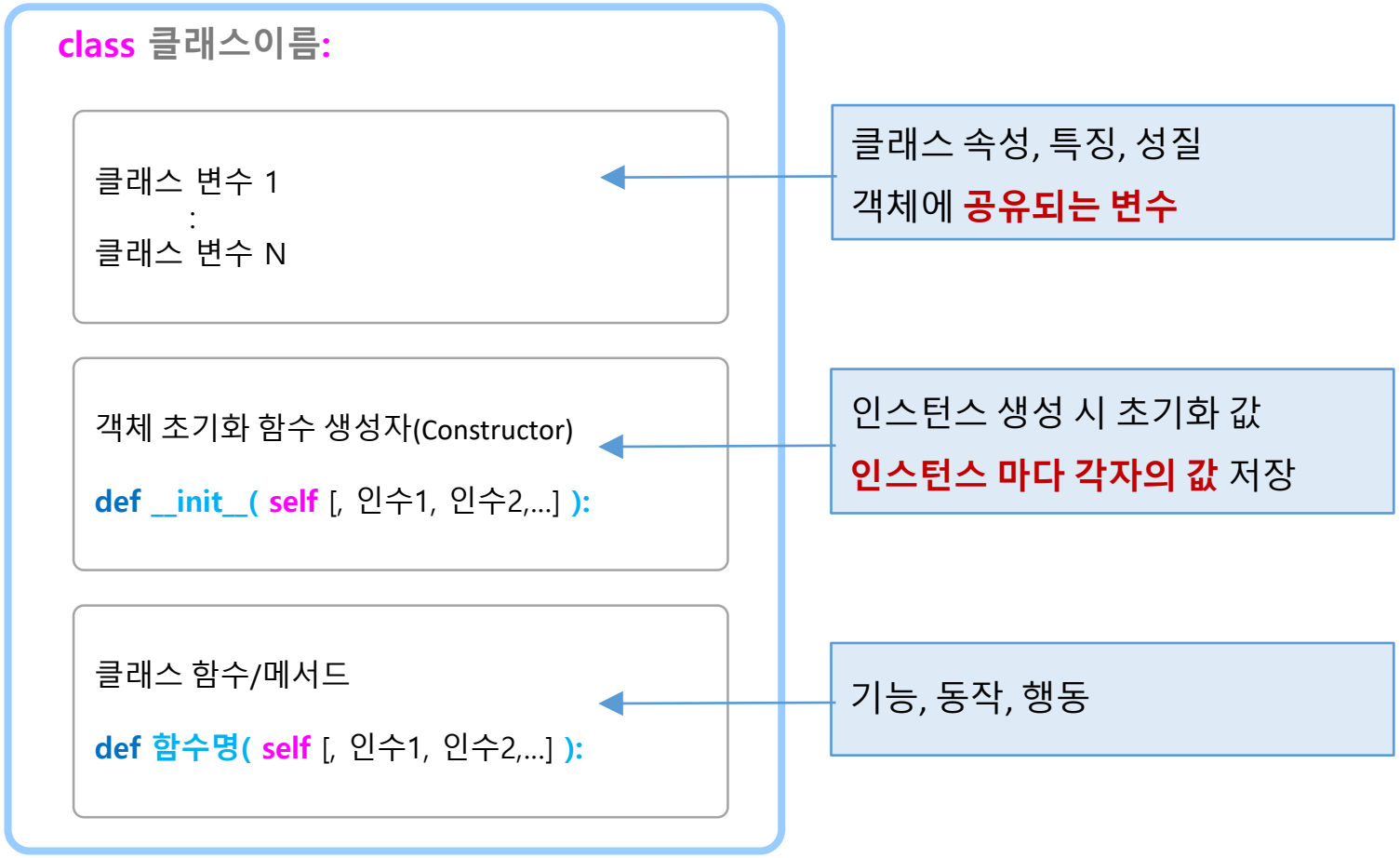
◆ 클래스(CLASS)

- **self** : 클래스로 객체를 생성한 경우 **인스턴스가 저장된 변수**



PYTHON 클래스

◆ 클래스(CLASS) 구성



PYTHON 클래스

◆ 클래스(CLASS) 구성

- 다양한 클래스 생성

```
class ProductNone:
    pass
```

속성도 메서드도 없는 **클래스**

PYTHON 클래스

◆ 클래스(CLASS) 구성

- 다양한 클래스 생성

```
class Product:


    def displayInfo(self, name, price):
        print(f"PNAME = {name}")
        print(f"PRICE = {price}₩n")
```

메서드 만 존재하는 **클래스**

PYTHON 클래스

◆ 클래스(CLASS) 구성

- 다양한 클래스 생성

```
class Product:
    def __init__(self, pname, price):
        self.pname=pname
        self.price=price

    def displayInfo(self, name, price):
        print(f"PNAME = {self.pname}")
        print(f"PRICE = {self.price}₩n")
```

필드(속성) 생성

필드, 메서드
모두 존재하는 **클래스**

특별히 생성자 메서드 라 함

PYTHON 클래스

◆ 클래스(CLASS) 구성

- 객체(인스턴스) 생성

객체변수명 = 클래스명()

```
class ProductNone:
    pass
```

생성

```
p1 = ProductNone()
p2 = ProductNone()
```

PYTHON 클래스

◆ 클래스(CLASS) 구성

- 객체(인스턴스) 속성 & 메서드 사용

속성값 변경 : 객체변수명.**속성명 = 값**

속성값 읽기 : 객체변수명.**속성명**

메서드 호출 : 객체변수명.**메서드명( )**

PYTHON 클래스

◆ 클래스(CLASS) 구성

- 객체(인스턴스) 속성 & 메서드 사용

```
# ------------------------------------------
# 객체(인스턴스) 생성
# 방법(규칙) :  변수명 = 클래스명()
# ------------------------------------------
p1=Product('Cake', 'Home', 10000)
p2=Product('Bag', 'PARK', 5000)



# ------------------------------------------
# 객체(인스턴스)의 메서드 또는 속성(필드) 사용
# 객체(인스턴스)변수명.메서드()
# ------------------------------------------
p1.displayInfo()                    # 객체( Product 인스턴스 ) 메서드 사용
```

PYTHON 클래스

◆ 클래스(CLASS) 구성

- 객체(인스턴스) 속성 & 메서드 사용

```
# ------------------------------------------------
# 객체(인스턴스)의 메서드 또는 속성(필드) 사용
# 객체(인스턴스)변수명.속성명
# ------------------------------------------------
print(f"p1.pmaker =>{p1.pmaker}")      # 객체( Product 인스턴스 ) 필드 값 사용
print(f"p1.price =>{p1.price}")        # 객체( Product 인스턴스 ) 필드 값 사용

p1.price=25000                         # 객체( Product 인스턴스 ) 필드 값 변경
p1.pmaker='Pari'                       # 객체( Product 인스턴스 ) 필드 값 변경

print(f"p1.pmaker =>{p1.pmaker}")      # 객체( Product 인스턴스 ) 필드 값 사용
print(f"p1.price =>{p1.price}")        # 객체( Product 인스턴스 ) 필드 값 사용
```

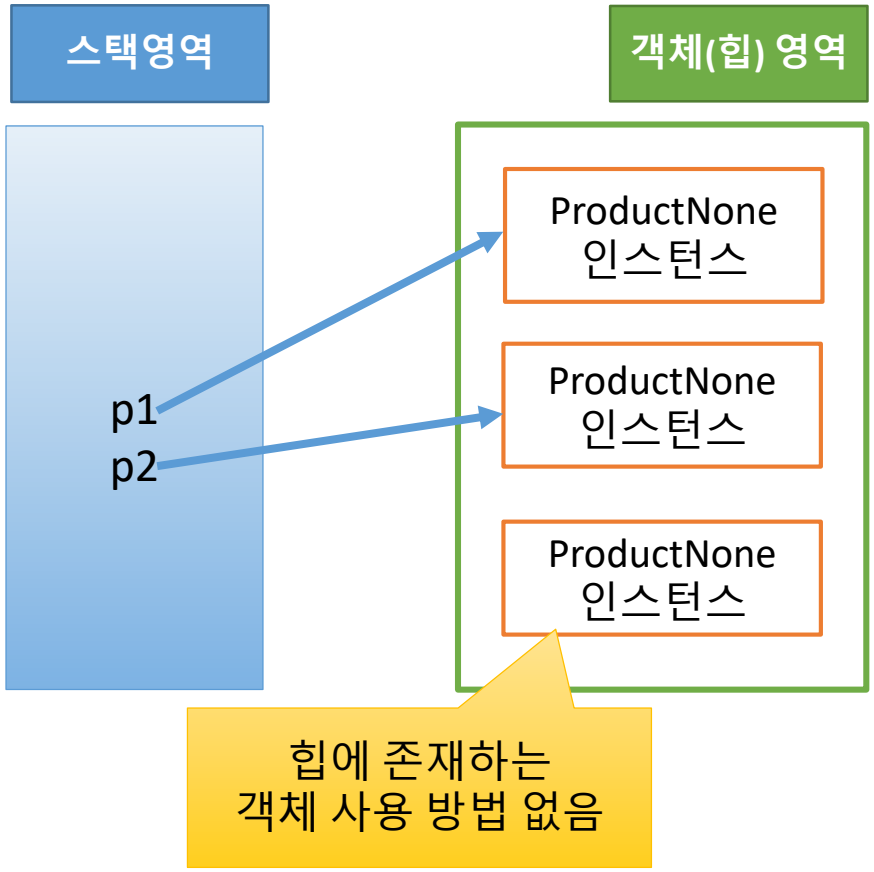
PYTHON 클래스

◆ 인스턴스 생성

```
class ProductNone:
    pass
```

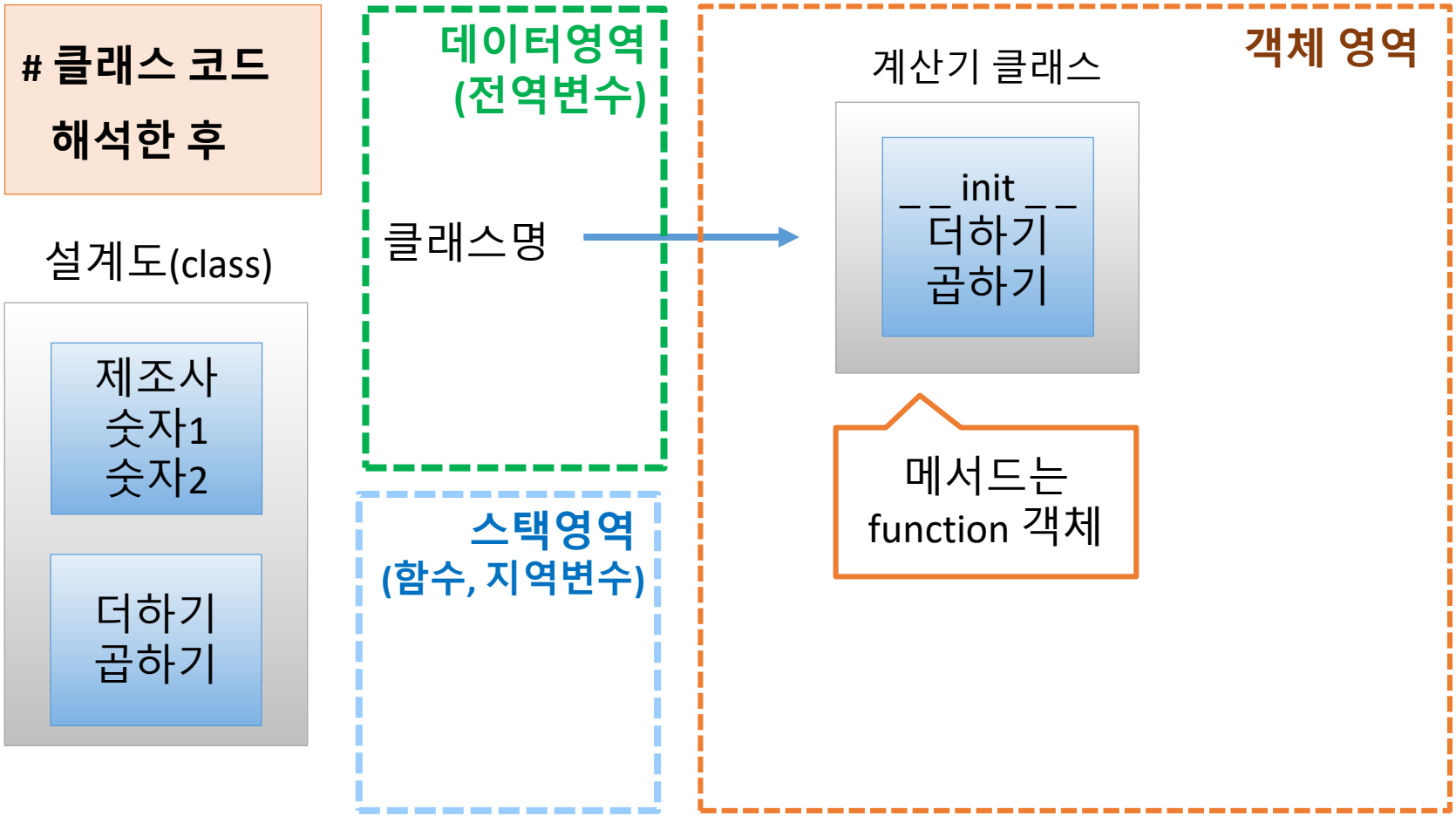
생성

```
p1 = ProductNone()
p2 = ProductNone()
ProductNone()
```



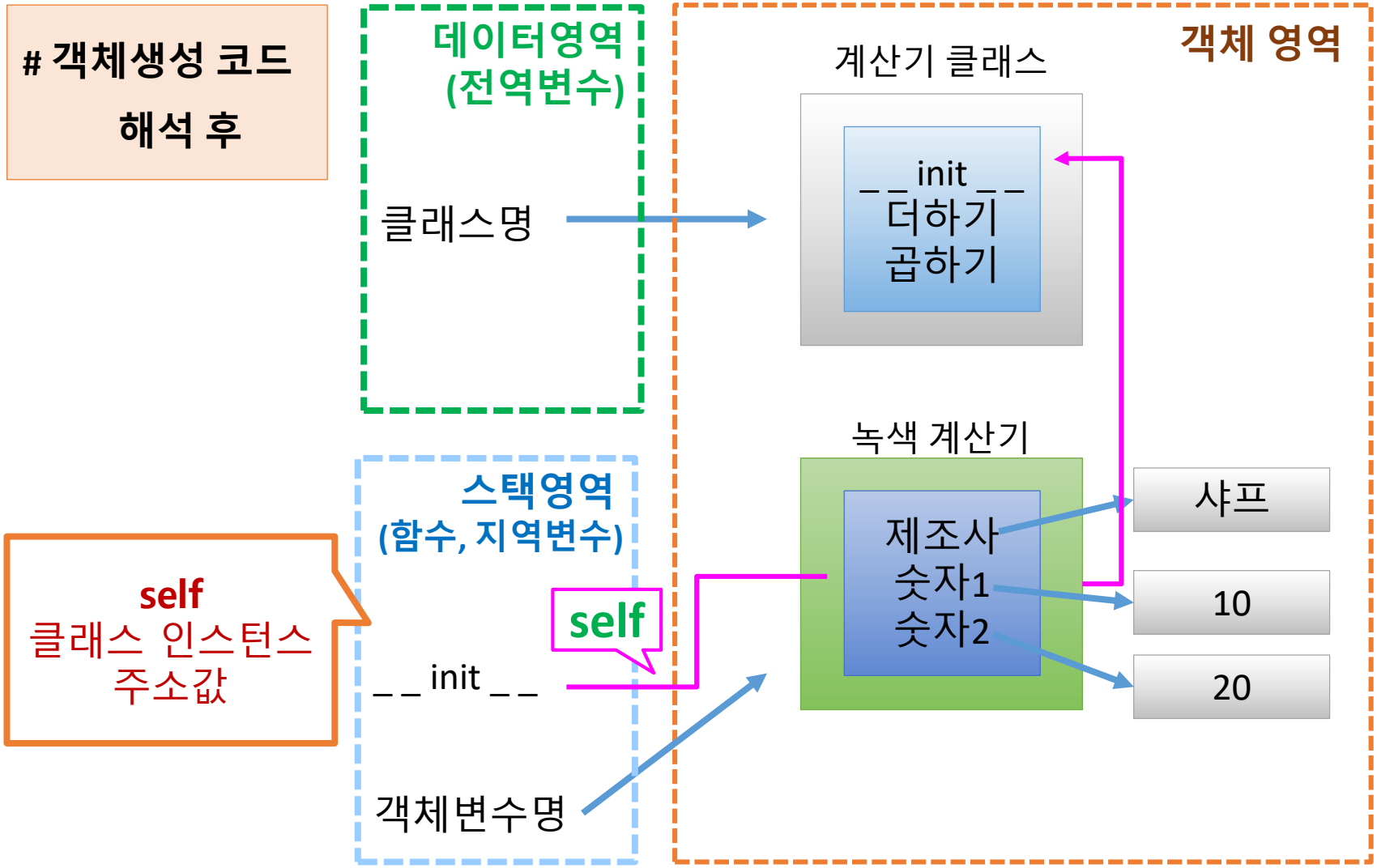
PYTHON 클래스

◆ 인스턴스 생성



PYTHON 클래스

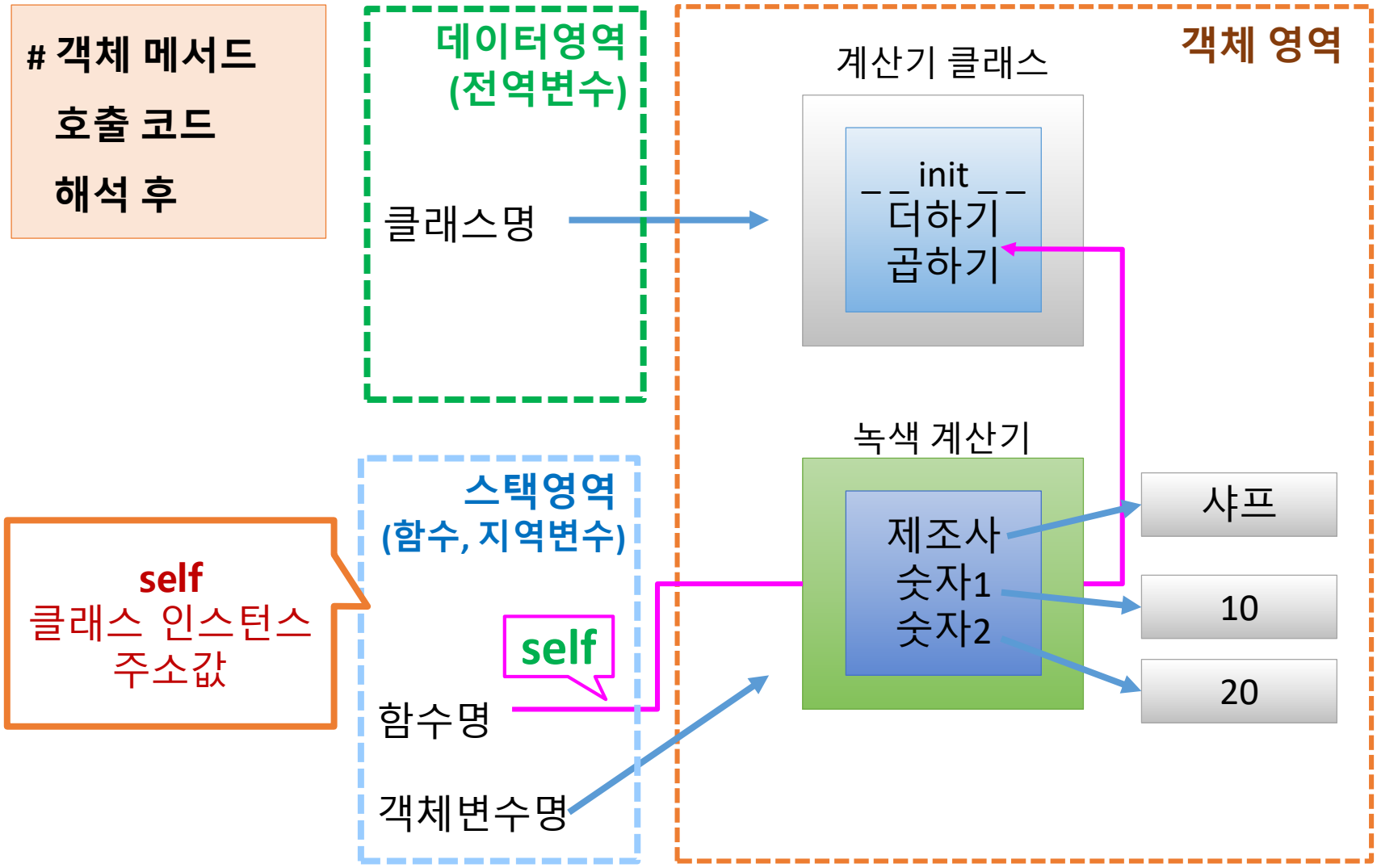
◆ 인스턴스 생성



PYTHON 클래스

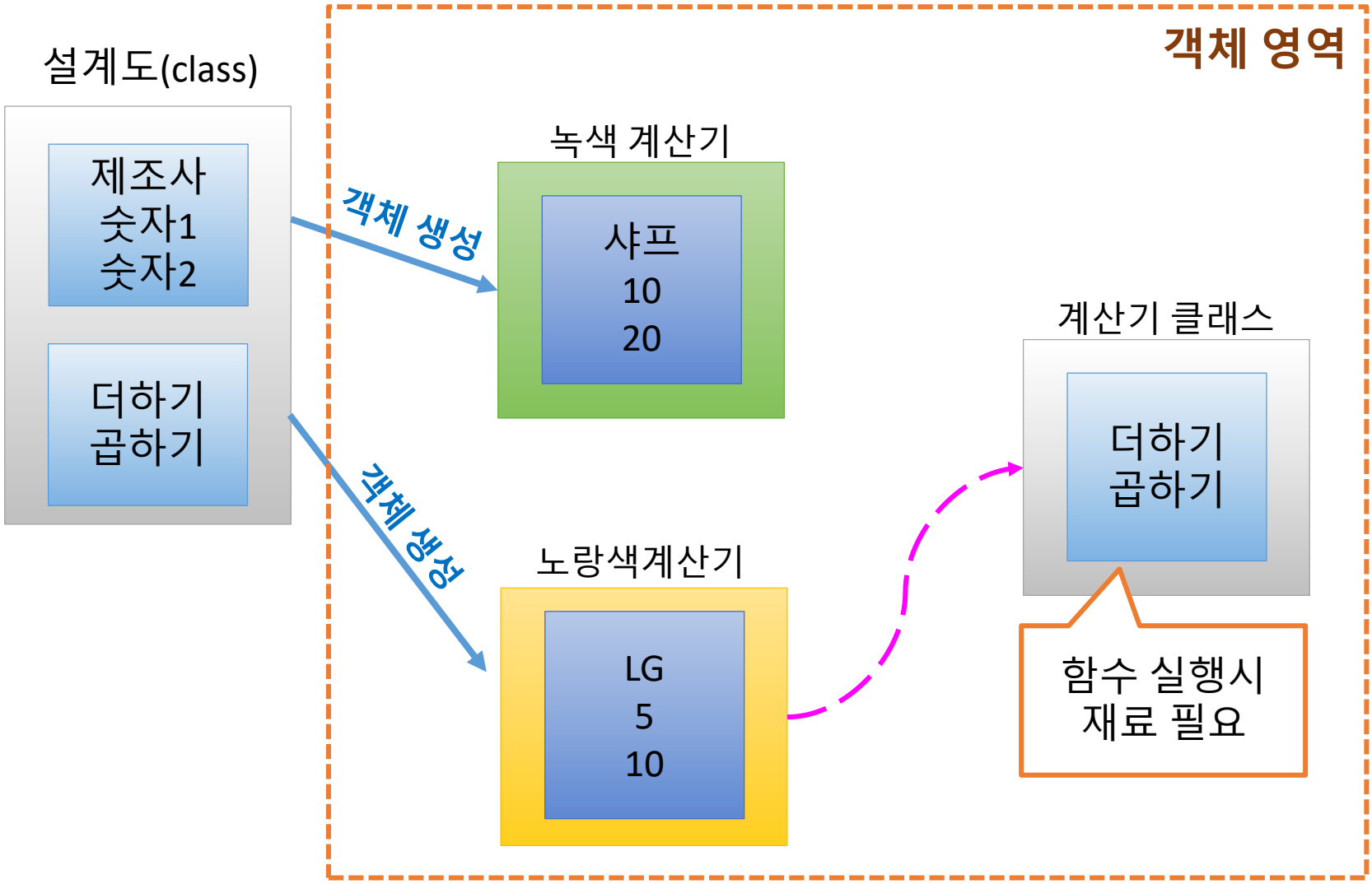
◆ 인스턴스 생성

객체 메서드
호출 코드
해석 후



PYTHON 클래스

◆ 인스턴스 생성



PYTHON 클래스

◆ 클래스(CLASS) 구성

- 속성 & 메서드 확인

인스턴스명._ _ dict _ _

클래스명._ _ dict _ _

```
print(f'p1.__dict__ : {p1.__dict__}')
print(f'Product.__dict__ : {Product.__dict__}')
```

```
p1.__dict__ : {'pname': 'Cake', 'pmaker': 'Home', 'price': 10000}

Product.__dict__ : {'__module__': '__main__', '__init__': <function
Product.__init__ at 0x000001F3C58B2DC8>, 'displayInfo': <function
Product.displayInfo at 0x000001F3C58B2D38>, '__dict__':
<attribute '__dict__' of 'Product' objects>, '__weakref__': <attribute
'__weakref__' of 'Product' objects>, '__doc__': None}
```

PYTHON 클래스

◆ 클래스(CLASS) 구성

생성자(Constructor)

- 객체(인스턴스) 생성 시 호출
- 인스턴스 변수 초기화
- **def _ _init_ _ (self)**

소멸자(Destructor)

- 객체(인스턴스) 제거 시 호출
- 인스턴스 변수 해제
- **def _ _del_ _ (self)**

PYTHON 클래스

◆ 클래스(CLASS) 구성

- 스페셜 메서드 / 매직 메서드

파이썬 시스템에서 **자동 호출되는 메서드**

형태 : **_ _메서드명_ _()**

PYTHON 클래스

◆ 클래스(CLASS) 구성

```
class CC:
    name="계산기"

    def add(self, first, second):
        return first + second

    def sub(self, first, second):
        return first - second
```

```
class CCC:
    name = "계산기"

    def __init__(self, first, second):
        self.first=first
        self.second=second

    def add(self):
        return self.first + self.second

    def sub(self):
        return self.first - self.second
```

```
c=CC()
print( "{0} + {1} = {2}".format(10, 20, c.add(10, 20)))
print( "{0} - {1} = {2}".format(10, 20, c.sub(10, 20)))

ccc=CCC(1000, 3000)
print( "{0} + {1} = {2}".format(ccc.first, ccc.second, ccc.add()))
print( "{0} - {1} = {2}".format(ccc.first, ccc.second, ccc.sub()))
```

PYTHON 클래스

◆ 클래스(CLASS)

- 변수 종류

인스턴스 변수

- 인스턴스 마다 존재하는 변수
- 객체변수명으로 읽기 & 변경

비공개 변수 : __변수명

- 객체변수명으로 보이지 않음
- 클래스 내에서만 사용 가능

```
class CCC:

    def __init__(self, datas, age):
        self.data=datas
        self._ _age=age

    def getsum(self):
        sum=0
        for i in range(0,len(self.data)):
            sum += self.data[i]
        return sum
```

PYTHON 클래스

◆ 클래스(CLASS)

- **변수 종류**

클래스 변수

- 인스턴스 공유하는 변수
- 접근 : 클래스명.변수명

비공개 변수 : __변수명

- 클래스명으로 보이지 않음
- 클래스 내부에서만 사용 가능

```
class CCC:
    __share = 1000

    def __init__(self, datas, age):
        self.data=datas
        self.__age=age

    def getsum(self):
        sum=0
        for i in range(0,len(self.data)):
            sum += self.data[i]
        return sum
```

PYTHON 클래스

◆ 클래스(CLASS)

- **오버로딩(overloading)**

→ 함수이름 동일

→ 매개변수 개수, 타입, 순서가 다른 함수 정의

→ OOP의 다형성 중 하나

PYTHON 클래스

◆ 클래스(CLASS)

- **연산자 오버로딩(overloadring)**

  → **객체**에서 **연산자를 클래스 목적에 맞게 기능 부여** 사용

  → 함수이름 앞뒤에 언더스코어(_ _) 두개 연속으로 붙은 함수

형태 : def _ _함수이름_ _():

매직함수명	연산자
def __add__(self, other)	+
def __sub__(self, other)	-
def __mul__(self, other)	*
def __truediv__(self, other)	/
def __floordiv__(self, other)	//
def __mod__(self, other)	%
def __pow__(self, other)	**

PYTHON 클래스

◆ 클래스(CLASS)

- 연산자 오버로딩

```
# 클래스 생성 ------------------------
class A:
    def __init__(self,num):
        self.num=num

# 인스턴스 생성 ----------------------
a=A(10)
b=A(20)

# + 연산 및 출력 ---------------------
print(a+b)
```

```
-------------------------------------------------------------------------
Traceback (most recent call last):
  File "C:/PyChamProject/DAY03/src/EX_Class.py", line 103, in <module>
    print(a+b)
TypeError: unsupported operand type(s) for +: 'A' and 'A'
```

PYTHON 클래스

◆ 클래스(CLASS)

➢ 연산자 오버로딩

```
# 클래스 생성 ---------------------------
class A:
    def __init__(self,num):
        self.num=num

    def __add__(self, other):
        return self.num+other.num

# 인스턴스 생성 ---------------------------
a=A(10)
b=A(20)

# + 연산 및 출력 ---------------------------
print(a+b)
```

PYTHON 클래스

◆ 클래스(CLASS)

➢ Setter & Getter 메서드

- 변수의 값 설정 ➔ setter method
- 변수의 값 읽기 ➔ getter method

PYTHON 클래스

◆ 상속(inheritance)

- 클래스 확대 및 **기존 클래스 재사용 & 기능 확장**
- 부모 클래스가 가진 것 모두 자식 클래스에서 사용

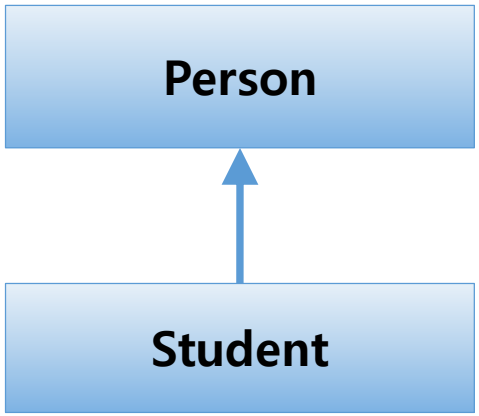
- 부모 클래스로부터 상속받은 함수를 **재정의** 가능
  - **오버라이딩(Overriding)**

형식 ➔ **class 자식클래스명 (부모클래스명)**

PYTHON 클래스

◆ 상속(inheritance)

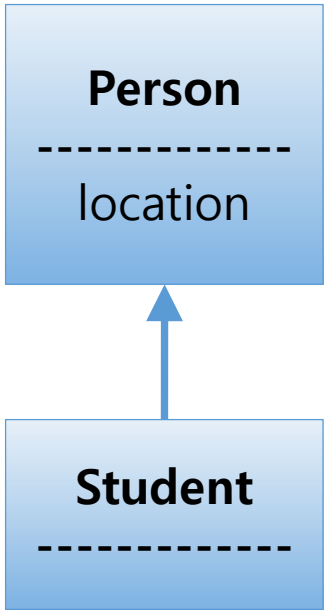
```
class Person:
    pass

class Student(Person):
    pass
```




```
# 상속 관계 확인하기 ----------------------------------------------------
print(f"Student는 Person의 자식? { issubclass(Student, Person)}")
```

PYTHON 클래스

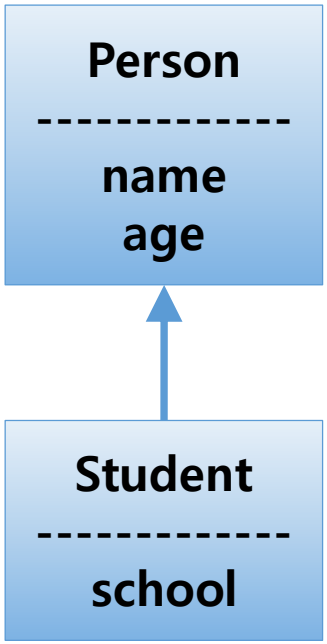
◆ 상속(inheritance)




```
class Person:
    def __init__(self):
        self.location='Korea'

class Student(Person):

    def showInfo(self):
        print(f"Location : {self.location}")
```

PYTHON 클래스

◆ 상속(inheritance)




```
class Person:
    def __init__(self, name, age):
        print("Person __init__() ")
        self.name=name
        self.age=age

class Student(Person):
    def __init__(self, name, age, school):
        print("Student __init__() ")
        #self.name = name
        #self.age = age
        #Person.name=name
        #Person.age=age
        #Person.__init__(self,name,age)
        super().__init__(name, age)
        self.school=school

    def showInfo(self):
        print(f"name : { self.name}₩tschool : {self.school}")
```

PYTHON 클래스

◆ 상속(inheritance)

➢ 오버라이딩(Overring)

- 함수 구현 부분만 다시 **재정의**
- 상속관계에서 **부모에서 상속 받은 메소드**에 한정

PYTHON 클래스

◆ 상속(inheritance)

```
# 클래스 생성 ----------------------------
class Point:
    def __init__(self, x, y):
        self.x=x
        self.y=y

    def show_point(self):
        return  "({0}, {1})" .format(self.x, self.y)
```

```
# 클래스 생성 ----------------------------
class DPoint(Point):
    def __init__(self, x, y, z):
        #self.x=x
        #self.y=y
        super().__init__(x, y):
        self.z=z

    def show_point(self):
        return  "({0}, {1}, {2})" .format(self.x, self.y, self.z)
```

오버라이딩

PYTHON 클래스

◆ Object 클래스

- 모든 클래스의 부모 클래스
- 자동으로 상속받게 되는 클래스

```
__class__                                   object
__str__(self)                               object
__annotations__                             object
__delattr__(self, name)                     object
__dir__(self)                               object
__eq__(self, o)                             object
__format__(self, format_spec)               object
__getattribute__(self, name)                object
__hash__(self)                              object
__init_subclass__(cls)                      object
__ne__(self, o)                             object
__new__(cls)                                object
```