

# 학습 내용

2부. 프로그래밍 언어 활용

6장. 모듈과 패키지



7장. 객체지향 프로그래밍

- 1. 객체와 클래스
- 2. 상속과 재정의

8장. 예외 처리

9장. 파일 입/출력 프로그래밍

10장. 데이터베이스 연동

# 1절 객체와 클래스

- 클래스 : 객체를 만들기 위한 틀(template, blueprint)
- 객체 : 클래스의 인스턴스
- 객체의 명사적 특성 : 데이터(Data), 변수(Variable), 필드(Field), 속성(Attribute)
- 객체의 동사적 특성 : 행위(Behavior), 함수(Function), 메서드(Method), 기능(Operation)

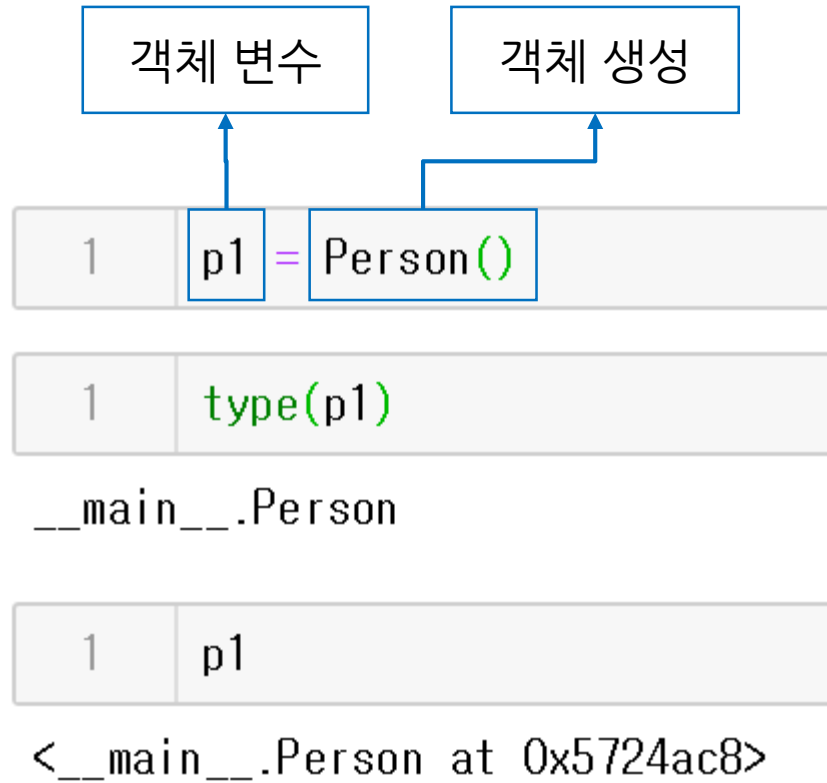
# 클래스 정의

- class 키워드 사용
- 카멜 표기법

```
class ClassName:  
    class_body
```

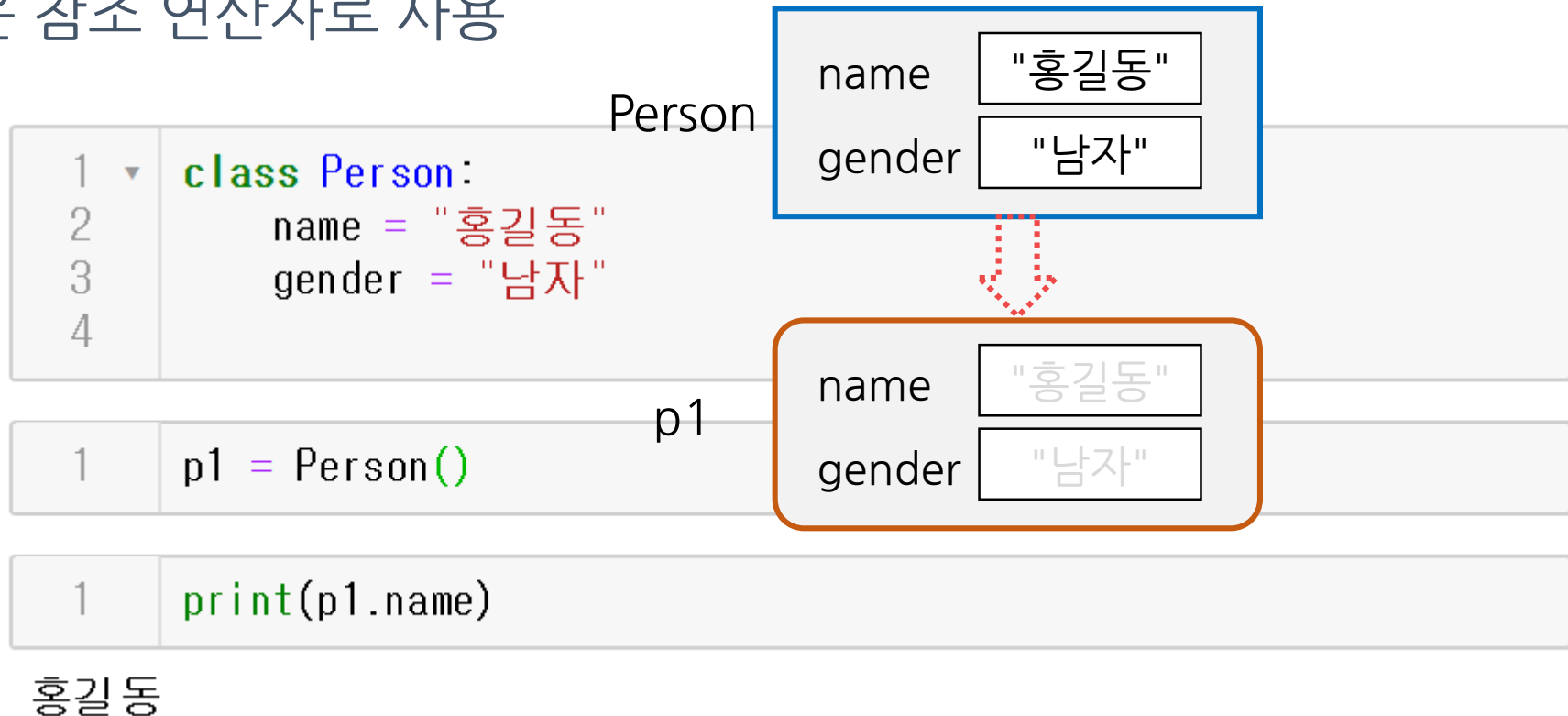
```
1  ▼ class Person:  
2      pass  
3
```

# 객체 생성



## 2절. 변수와 메소드

- 클래스를 사용하는 가장 큰 이유
  - 객체를 이용해 데이터를 저장하기 위해 → 변수
  - 객체 고유의 기능을 갖기 위해 → 메서드
- .은 참조 연산자로 사용



# 객체에 멤버 추가

```

1  ▼ class Person:
2  ▼     def print_info():
3      print("Person 객체입니다.")
4

```

```

1  p1 = Person()

```

```

1  p1.print_info()

```

‘Person 객체입니다’ 라는 문장을 출력할 것 같았지만 실제로는 에러가 발생

**TypeError**

Traceback (most recent ca

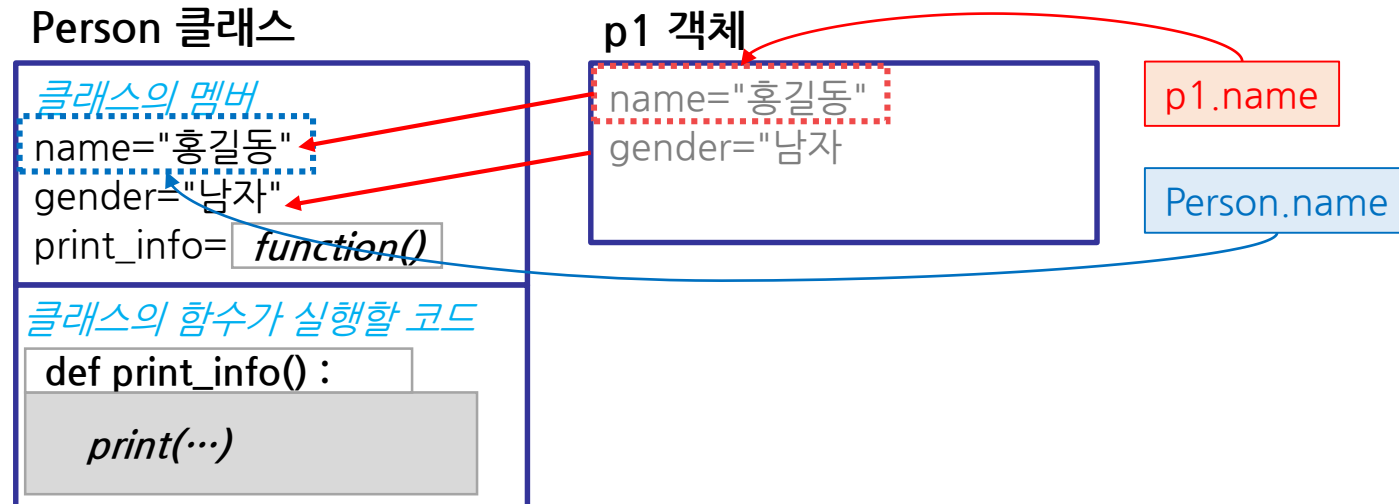
ll last)

<ipython-input-10-123b1e7cbd59> in <module>()

----> 1 p1.print\_info()

**TypeError:** print\_info() takes 0 positional arguments but 1 was give  
n

# 객체를 이용한 참조와 클래스를 이용한 참조



# 인스턴스 메서드

- 객체를 이용해 참조할 수 있는 메서드
- 인스턴스 메서드의 첫 번째 인자는 self여야 함
- self 인자는 객체의 멤버(변수 또는 메서드)에 접근하기 위해 사용

```
1 class Person:  
2     def print_info(self):  
3         print("Person 객체입니다.")  
4
```

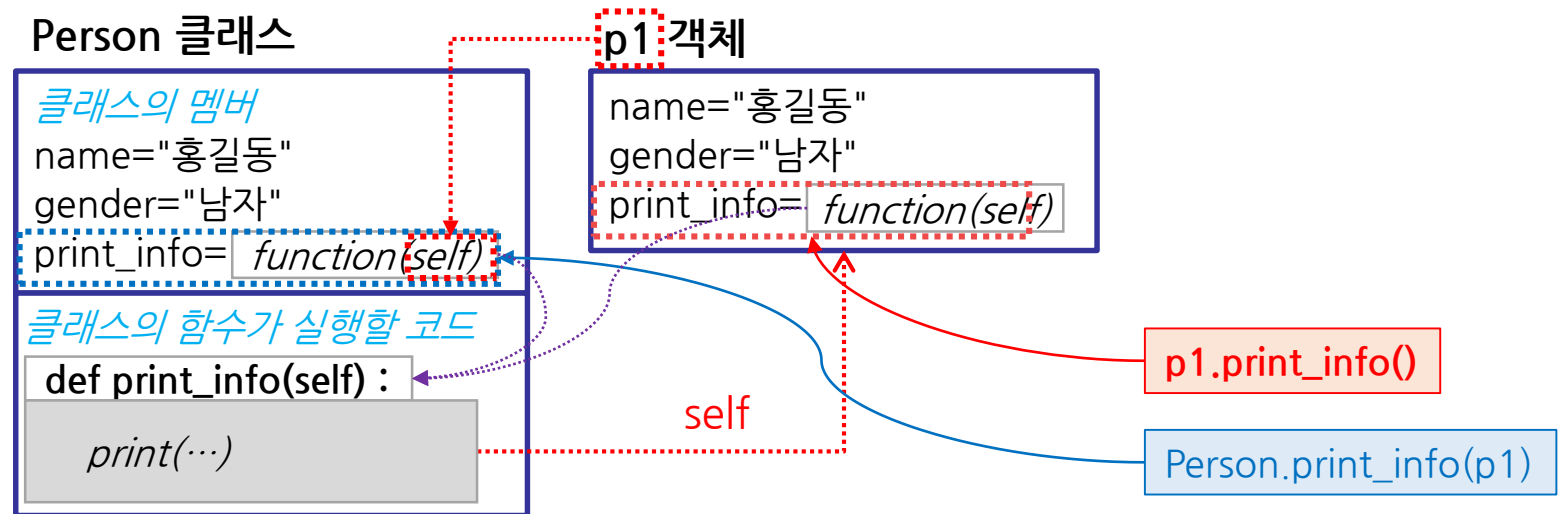
```
1 p1 = Person()
```

```
1 p1.print_info()
```

Person 객체입니다.



# 객체를 이용한 참조와 클래스를 이용한 참조



# 클래스의 멤버변수 접근

```

1 class Person:
2     name = "홍길동"
3     gender = "남자"
4     def print_info(self):
5         print("{}님은 {}입니다.".format(name, gender))
6

```

```

1 p1 = Person()

```

```

1 p1.print_info()

```

- ❖ 인스턴스 메서드는 객체를 통해 참조하는 메서드.
- ❖ 객체가 갖는 멤버에 접근하려면 객체 자신을 참조할 수 있는 그 무엇이 있어야 함

**NameError**

Traceback (most recent call last)

```
<ipython-input-16-123b1e7cbd59> in <module>()
```

```
----> 1 p1.print_info()
```

```
<ipython-input-14-057c19c3811b> in print_info(self)
```

```
3     gender = "남자"
```

```
4     def print_info(self):
```

```
----> 5         print("{}님은 {}입니다.".format(name, gender))
```

```
6
```

**NameError**: name 'name' is not defined

# self

- self를 인스턴스 메서드의 인자로 정의해서 자신 객체의 멤버를 참조할 수 있도록 하는 것
- self 대신에 파이썬의 키워드가 아니라면 다른 단어를 사용할 수 있음
- 자신 객체임을 명백히 알리기 위해 self를 사용할 것을 권장

```
1 class Person:
2     name = "홍길동"
3     gender = "남자"
4     def print_info(self):
5         print("{}님은 {}입니다.".format(self.name, self.gender))
6
```

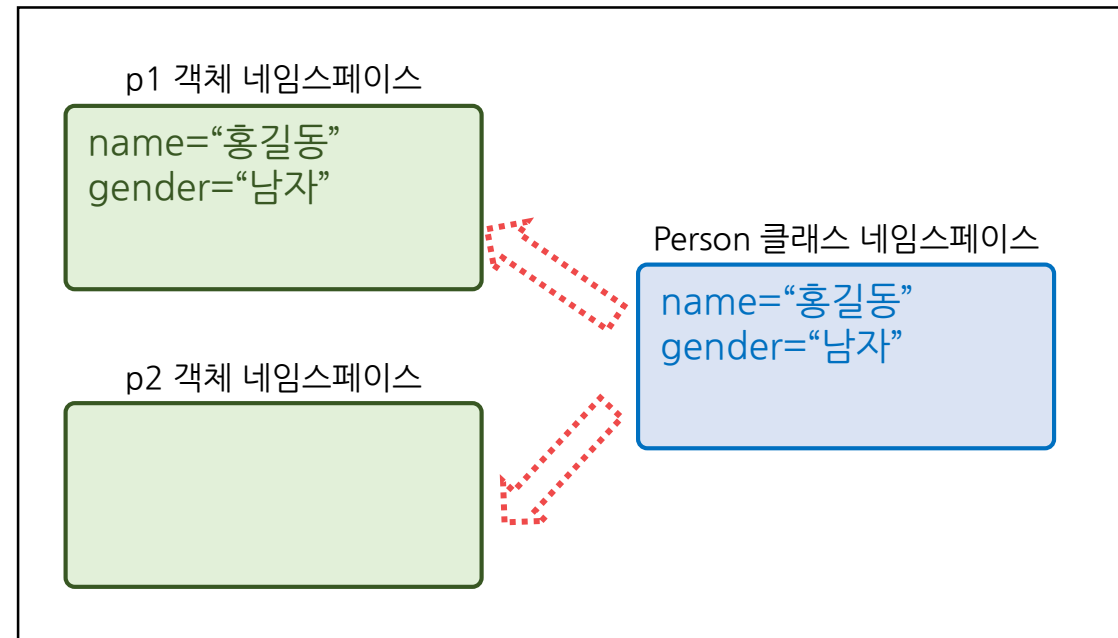
```
1 p1 = Person()
```

```
1 p1.print_info()
```

홍길동님은 남자입니다.

# 네임스페이스

- 파이썬은 클래스 객체와 인스턴스 객체의 이름 공간(namespace)이 분리되어 있음
- 클래스 객체와 인스턴스 객체의 이름공간이 다르다는 의미
- 파이썬은 동적으로 인스턴스 멤버를 추가하는 것이 가능
- 인스턴스 객체를 통해 변수나 함수의 이름을 찾는 순서
  - 1. 인스턴스 영역
  - 2. 클래스 영역
  - 3. 전역 영역



## class 메서드와 static 메서드

```

1 class Person:
2     name = "홍길동"
3     gender = "남자"
4     def print_info(self):
5         print("{}님은 {}입니다.".format(self.name, self.gender))
6
7     @classmethod
8     def do_(cls):
9         print("이름 : {}, 성별 : {}".format(cls.name, cls.gender))
10
11    @staticmethod
12    def that_():
13        print("이름은 {}이고, 성별은 {}입니다."
14              .format(Person.name, Person.gender));

```

```
1 Person.do_()
```

이름 : 홍길동, 성별 : 남자

객체를 만들지 않고 클래스 이름으로 참조 가능

```
1 Person.that_()
```

이름은 홍길동이고, 성별은 남자입니다.

## class 메서드와 static 메서드

```
1 p1 = Person()
2 p1.do_()
```

이름 : 홍길동, 성별 : 남자

```
1 p1.name = "이순신"
2 p1.do_()
```

이름 : 홍길동, 성별 : 남자

```
1 p1.that_()
```

이름은 홍길동이고, 성별은 남자입니다.

```
1 Person.do_()
```

이름 : 홍길동, 성별 : 남자

```
1 Person.that_()
```

이름은 홍길동이고, 성별은 남자입니다.

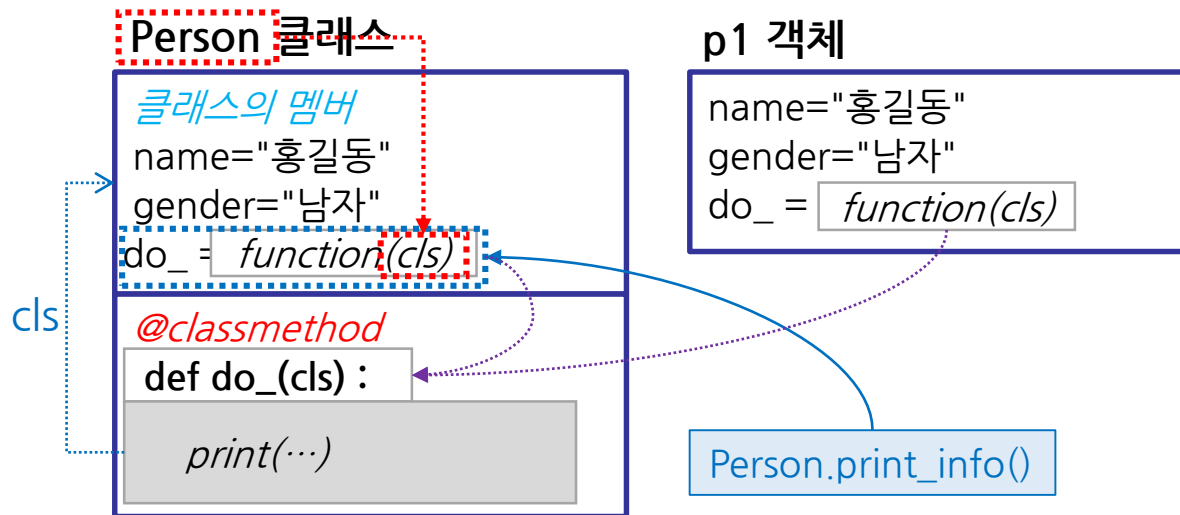
```
1 p1.name
```

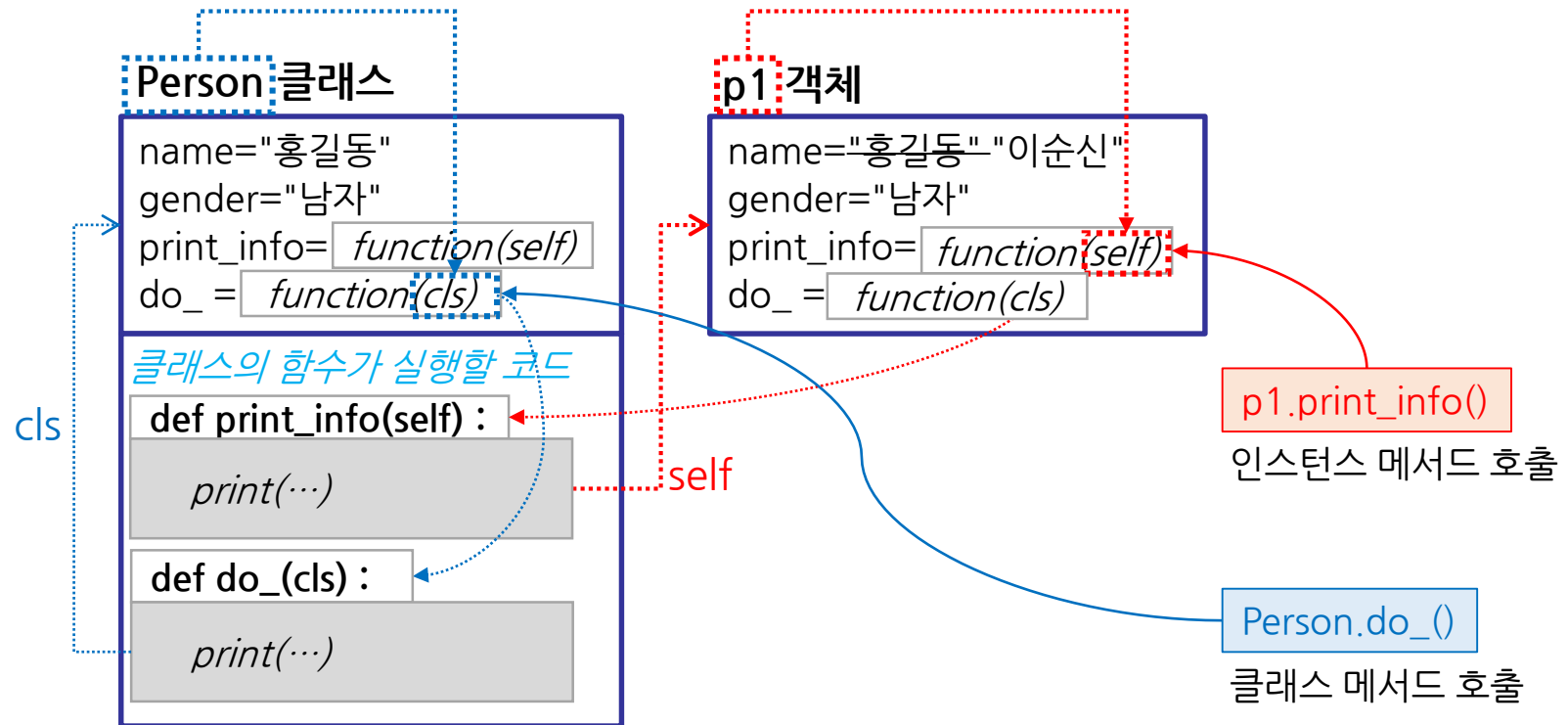
'이순신'

```
1 p1.print_info()
```

이순신님은 남자입니다.

# 객체를 이용한 참조와 클래스를 이용한 참조



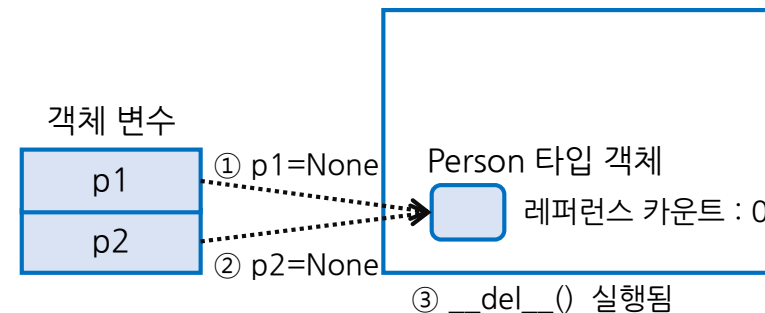
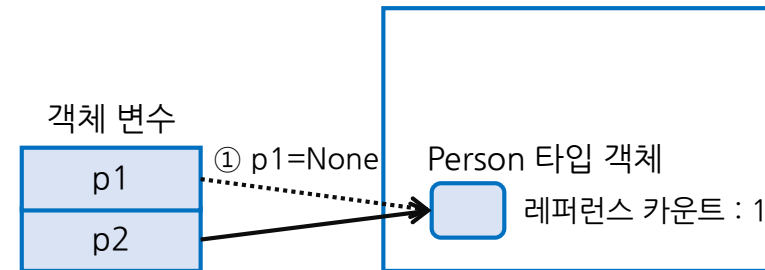
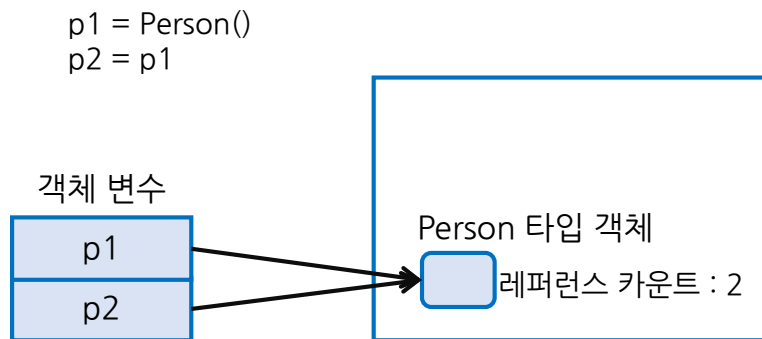




### 3절. 생성자와 소멸자

생성자는 객체가 생성될 때 자동으로 실행되며, 생성 시 필요한 코드를 포함할 수 있습니다. 생성자의 이름은 `__init__()`입니다.

소멸자는 객체가 소멸될 때 자동으로 실행되며, 소멸 시 필요한 코드를 포함할 수 있습니다. 소멸자의 이름은 `__del__()`입니다. 객체는 인스턴스 객체의 레퍼런스 카운트가 0이 될 때 소멸됩니다.



# 생성자와 소멸자

## ● 생성자

- 생성자(Constructor)는 객체가 생성될 때 자동으로 실행
- 생성 시 필요한 코드를 포함할 수 있음
- 생성자의 이름은 `__init__()`

## ● 소멸자

- 객체가 소멸될 때 자동으로 실행
- 소멸 시 필요한 코드를 포함할 수 있음
- 소멸자의 이름은 `__del__()`
- 인스턴스 객체의 레퍼런스 카운트가 0이 될 때 소멸

# 생성자와 소멸자를 갖는 클래스

```
class Person:
    def __init__(self):
        print("Person 객체를 생성합니다.")
        self.name = "홍길동"
        self.gender = "남자"

    def __del__(self):
        print("Person 객체를 소멸시킵니다.")

    def print_info(self):
        print("{}님은 {}입니다.".format(self.name, self.gender))
```

```
p1 = Person()
```

Person 객체를 생성합니다.

```
p1.print_info()
```

홍길동님은 남자입니다.

# 객체 소멸시키기

```
p1 = Person()
```

기존 객체 변수에 새로운 객체를 할당

Person 객체를 생성합니다.  
Person 객체를 소멸 시킵니다.

```
p2 = Person()
```

Person 객체를 생성합니다.

```
p2 = None
```

객체에 None을 할당

Person 객체를 소멸 시킵니다.

```
p3 = Person()
```

Person 객체를 생성합니다.

```
del p3
```

del 명령으로 변수를 삭제

Person 객체를 소멸 시킵니다.

# 생성자를 이용한 인스턴스 변수 초기화

```
class Person:
    def __init__(self, name, gender):
        print("Person 객체를 생성합니다.")
        self.name = name
        self.gender = gender

    def __del__(self):
        print("Person 객체를 소멸시킵니다.")

    def print_info(self):
        print("{}님은 {}입니다.".format(self.name, self.gender))
```

```
p1 = Person("홍길서", "여자")
```

Person 객체를 생성합니다.

```
p1.print_info()
```

홍길서님은 여자입니다.

# 생성자 중복정의 불가

```
class Person:
```

```
def __init__(self, name):
    print("Person 객체를 생성합니다.")
    self.name = name
    self.gender = "여자"
```

```
def __init__(self, name, gender):
    print("Person 객체를 생성합니다.")
    self.name = name
    self.gender = gender
```

```
def __del__(self):
    print("Person 객체를 소멸시킵니다.")
```

```
def print_info(self):
    print("{}님은 {}입니다.".format(self.name, self.gender))
```

```
p1 = Person("홍길동")
```

Person 객체를 소멸시킵니다.

**TypeError**

```
<ipython-input-42-54d73f26a8ae> in <module>
----> 1 p1 = Person("홍길동")
```

**TypeError:** \_\_init\_\_() missing 1 required argument

```
p1 = Person("홍길서", "여자")
```

Person 객체를 생성합니다.

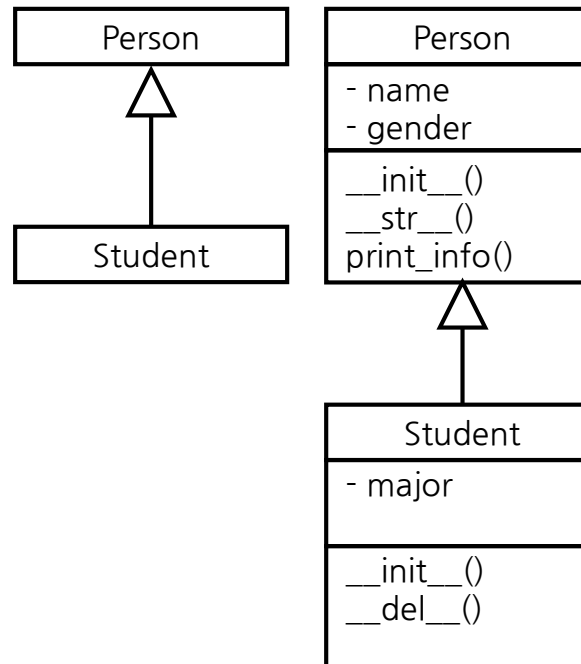
마지막 정의된 생성자만 유효함

## 4절. 상속과 재정의

- 상속(Inheritance)은 객체 재사용의 한 방법
- 상속을 이용하면 부모 클래스의 모든 속성들을 자식 클래스로 물려줄 수 있다

```
class SubClassName(SuperClassName) :
```

Student is a Person



## 상속

```

1  ▼ class Person:
2  ▼     def __init__(self, name, gender):
3         self.name = name
4         self.gender = gender
5
6  ▼     def __str__(self):
7         return "name: {0}, gender: {1}".format(self.name, self.gender)
8
9  ▼     def print_info(self):
10         print("{}님은 {}입니다.".format(self.name, self.gender))
11

```

```

1  ▼ class Student(Person):
2  ▼     def __init__(self, name, gender, major):
3         self.name = name
4         self.gender = gender
5         self.major = major
6
7  ▼     def __del__(self):
8         pass

```

```

1  issubclass(Student, Person)

```

True



# 부모클래스의 생성자 사용

- Student 클래스의 생성자는 Person 클래스의 생성자를 이용하여 인스턴스 변수를 초기화 가능
- 부모클래스의 생성자를 호출하여 자식클래스의 변수들을 쉽게 초기화
- 상속관계가 있을 경우 자식 클래스는 부모 클래스의 변수와 메서드를

```
1 class Student(Person):  
2     def __init__(self, name, gender, major):  
3         Person.__init__(self, name, gender)  
4         self.major = major
```

```
1 s1 = Student("홍길남", "남자", "경제학")  
2 s1.print_info()
```

홍길남님은 남자입니다.

# 재정의

- 부모 클래스에서 정의한 함수를 자식클래스에서 다시 정의

```

1  ▼ class Student(Person):
2  ▼     def __init__(self, name, gender, major):
3         Person.__init__(self, name, gender)
4         self.major = major
5
6  ▼     def __del__(self):
7         pass
8
9  ▼     def print_info(self):
10 ▼         print("{}님은 {}이며, 전공은 {}입니다."
11             .format(self.name, self.gender, self.major))
12

```

함수 이름과 파라미터가 부모클래스의 함수와 같아야 함

```

1  issubclass(Student, Person)

```

```

1  s2 = Student("홍길서", "여자", "컴퓨터공학")
2  s2.print_info()

```

홍길서님은 여자이며, 전공은 컴퓨터공학입니다.

# super()

- super()는 부모 클래스의 멤버를 참조

```

1  ▼ class Student(Person):
2  ▼     def __init__(self, name, gender, major):
3         Person.__init__(self, name, gender)
4         self.major = major
5
6  ▼     def __del__(self):
7         pass
8
9  ▼     def __str__(self):
10         return super().__str__() + ", major:{}".format(self.major)
11
12 ▼     def print_info(self):
13 ▼         print("{}님은 {}이며, 전공은 {}입니다."
14             .format(self.name, self.gender, self.major))
15

```

\_\_str\_\_() 메서드에서  
super().\_\_str\_\_()을 통해 Person의  
\_\_str\_\_() 메서드를 호출 가능

```

1  s3 = Student("홍길북", "남자", "심리학")

```

```

1  print(s3)

```

name: 홍길북, gender: 남자, major:심리학

# 정적 변수

- 클래스의 변수이름 앞에 \_\_ (under score 두 개)를 붙이면 내부적으로 **클래스명.클래스명 변수명** 형식으로 참조

```

1 class Student(Person):
2     __count = 0
3
4     def __init__(self, name, gender, major):
5         Student._Student__count += 1
6         Person.__init__(self, name, gender)
7         self.major = major
8
9     def __del__(self):
10        Student._Student__count -= 1
11
12    def __str__(self):↔
13
14
15    def print_info(self):↔
16
17
18
19    @classmethod
20    def get_count(cls):
21        return Student._Student__count

```

클래스 내의 어떤 메서드에서도 동일한 이름으로 참조할 수 있음

```

1 s1 = Student("홍길동", "남자", "컴퓨터공학")
2 Student.get_count()

```

1

```

1 s2 = Student("홍길서", "여자", "컴퓨터공학")
2 Student.get_count()

```

2

## 5절. 연습문제(실습형)

### 1. 다음 조건을 만족하는 클래스를 작성하세요

- ✓ 도형(Shape)클래스와 삼각형(Triangle) 클래스를 만들어야 합니다.
- ✓ 도형(Shape)클래스
  - 생성자를 통해 x, y좌표를 초기화할 수 있습니다. x, y좌표의 기본값은 0, 0입니다.
  - x, y좌표를 이동시킬 수 있는 move 함수가 있습니다.
  - x, y좌표값 정보를 문자열로 리턴하는 \_\_str\_\_ 함수가 있습니다
  - 도형의 면적을 구하는 함수(calc\_area)가 있습니다. 다만 도형 클래스의 calc\_area()는 구현되어 있지 않아 호출하면 오류가 발생합니다(파이썬은 추상메소드가 없음. 추상메소드 역할을 구현)
  - 정적메소드(staticmethod)를 하나 이상 구현합니다.
- ✓ '삼각형(Triangle)클래스는 도형(Shape)클래스이다'관계가 성립해야 합니다
  - ✓ 생성자를 통해, width, height, x, y좌표를 초기화하고 x, y 좌표의 기본값은 0, 0입니다
  - ✓ 삼각형의 면적을 구하는 메소드 calc\_area를 재정의합니다
  - ✓ 삼각형의 정보를 문자열로 반환하는 \_\_str\_\_() 함수가 있습니다.
  - ✓ 삼각형 객체가 몇 개가 만들어져 있는지 정보를 저장하는 클래스 변수를 추가하고 생성자에서 객체를 만들 때마다 증가시키고 소멸자에서 객체를 소멸시킬 때마다 감소합니다.

# 연습문제(서술형)

1. 다음 중 올바른 클래스 정의는?

- ① `class Person:`
- ② `class Person():`
- ③ `p1 = Person()`
- ④ `def Person:`

2. 다음 중 `Person` 클래스의 객체를 생성하는 올바른 방법은?

- ① `p1 = Person`
- ② `p1 = Person()`
- ③ `Person.p1`
- ④ `p1 = Person._init_()`

## 연습문제(서술형)

3. 다음 중 인스턴스 메소드를 선언하는 가장 올바른 방법은?

① `def print_info():`

② `@instancemethod`  
`def print_info():`

③ `@method`

`def print_info():`

④ `def print_info(self):`

4. 다음 중 생성자와 소멸자에 대해 잘 못 설명한 것은?

① 생성자는 중복(Overloading) 정의해서 사용할 수 있다.

② 소멸자는 객체 참조수가 0이 되면 실행된다.

③ 생성자의 원형은 `init_(self)`이다.

④ 생성자는 객체의 변수(또는 속성)를 초기화하는데 사용한다.

## 연습문제(서술형)

5. 다음 중 “Student is a Person”을 클래스 상속으로 바르게 표현한 것은?

- ① `class Person(Student):`
- ② `class Student(Person):`
- ③ `class Person is Student:`
- ④ `class Student: Person:`

6. 다음 중 상속과 재정의(Overriding)에 대해 잘 못 설명한 것은?

- ① 부모클래스의 메소드를 자식클래스에서 정의하는 것을 ‘재정의’라고 한다.
- ② 메소드의 재정의는 상속을 전제로 한다.
- ③ 재정의는 메소드의 이름은 같고, 매개변수의 수는 다르게 정의해야 한다.
- ④ `super()`를 이용하면 부모의 멤버를 호출할 수 있다.



# 연습문제(서술형)

7. 다음 클래스의 메소드를 호출 하는 방법 중 잘 못된 것은?

```
class SomeClass:  
    def method_a(self):  
        print("method_a")  
  
    @classmethod  
    def method_b(cls):  
        print("method_b")
```

```
obj = SomeClass()
```

- ① obj.method\_a()
- ② obj.method\_b()
- ③ SomeClass.method\_a()
- ④ SomeClass.method\_b()

# 연습문제(서술형)

8. 두 클래스와 코드의 실행결과 다음과 같이 출력되도록 해야한다면 빈 칸에 넣을 수 없는 것을 고르세요.

```
class Super:
    def do_(self, a): print
        t("super.do_")
```

```
class Sub(Super):
    def do_(self, a):
        print("sub.do_")
```

```
s1 = Sub()
s1.do_(10)
```

```
super.do_
sub.do_
```

- ① Super.do\_(self, a)
- ② super().do\_(a)
- ③ super.do\_(a)

## 연습문제(서술형)

9. 다음 중 파이썬의 특별할 용도로 정의되어 있는 속성과 메소드에 대한 설명 중 잘못된 것은?

- ① `__dict__` 속성을 이용하면 클래스의 멤버를 확인할 수 있다.
- ② `__str__()` 메소드는 `print()` 함수의 인자로 객체를 전달할 때 호출되는 메소드이다. 이 메소드는 매개변수를 가질 수 없다.
- ③ `__doc__` 속성은 독스트링을 갖는다.
- ④ `__init__()` 메소드는 생성자를 정의할 때 사용한다.