

Day 8 - Object Oriented and Class

클래스 - 객체를 만들기 위한 설계도 이다.

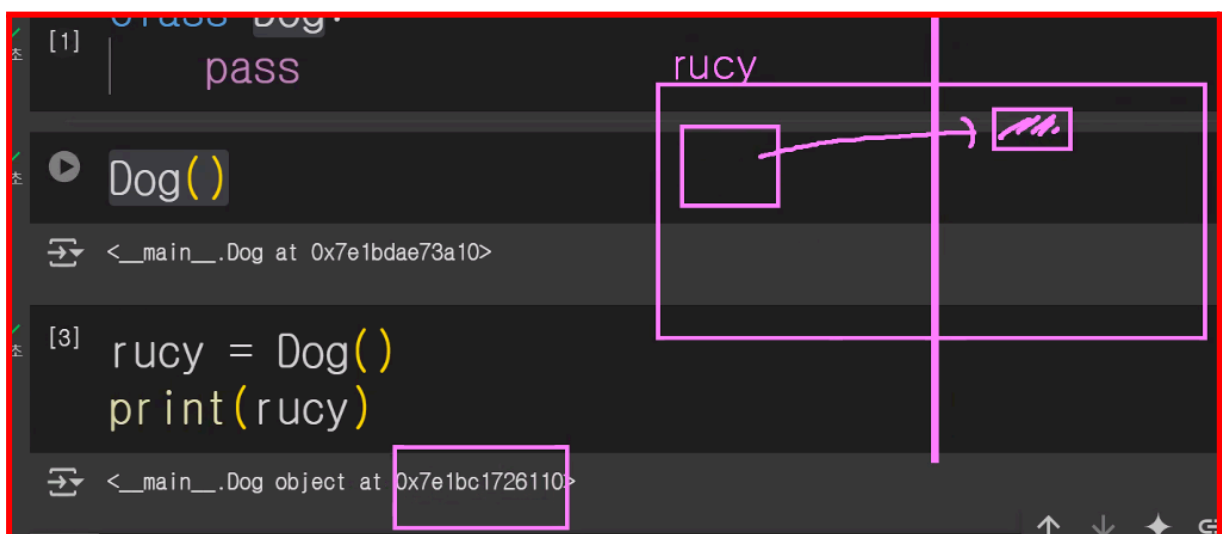
```
class 클래스이름:
    # 클래스 속성(멤버 변수) 정의
    속성1 = 초기값1
    속성2 = 초기값2

    # 생성자 메서드 (생략 가능)
    def __init__(self, 매개변수1, 매개변수2, ...):
        # 인스턴스 속성 초기화
        self.속성1 = 매개변수1
        self.속성2 = 매개변수2

    # 메서드(멤버 함수) 정의
    def 메서드1(self, 매개변수1, 매개변수2, ...):
        # 메서드 동작 정의
        pass

    def 메서드2(self, 매개변수1, 매개변수2, ...):
        # 메서드 동작 정의
        pass
```

- 클래스 안에 있는 변수와 함수는:
 - 속성(멤버 변수)
 - 메서드(멤버 함수) 라고 부른다.



- rucy가 dog를 가리키고 있는 객체가 만들어진다.

객체 생성 초기화:

The screenshot shows a Python IDE with a code editor on the left and a memory diagram on the right. The code in the editor is:

```
rucy name = '루시'  
rucy.age = 15  
rucy.family = '포메'  
print(rucy.name)  
print(rucy.age)  
print(rucy.family)
```

The variable `rucy` is highlighted with a yellow box. The memory diagram on the right shows a box labeled `Rucy` with arrows pointing to the memory locations of `name`, `age`, and `family`. The `name` box contains '루시', `age` contains 15, and `family` contains '포메'.

- 루시의 경우에는:
- 처음에는 클래스 변수로 가르키고 있다가.
- 값을 바꾸는 순간 인스턴스 변수의 값이 바뀐다.

생성자 (Constructor)

The screenshot shows a Python IDE with a code editor and a console. The code in the editor is:

```
class Dog:  
    def __init__(self):  
        print(self, 'init 호출!')
```

The `self` parameter in the `__init__` method is highlighted with a yellow circle. Below the code, the console shows the execution:

```
rucy = Dog()  
  
<__main__.Dog object at 0x7e1bb252af10> init 호출!
```

- () 는 `init`를 호출하는 함수를 나타낸다
- 없으면 내부적으로 빈 생성자를 만들고 그걸 호출한다.
- 만약 만들면 무조건 `init`을 호출한다
- `self`라는 변수나 무조건 변수 하나는 만들어줘야 한다.
- `init`라는 메서드는 하나만 만든다
 - 객체마다 메서드를 만들면 메모리 낭비이다 (어쨌든 변수가 아니라 실행시키는거라 상관없다)

```
[14] class Dog:
      def __init__(self):
          print(self, 'init 호출!')

rucy = Dog()

<__main__.Dog object at 0x7e1bb252af10> init 호출!

[16] PPomi = Dog()

<__main__.Dog object at 0x7e1bb2522590> init 호출!
```


- 메모리 주소가 다른걸 알수 있다.
- 하지만 메서드는 하나를 쓴다 (self)를 통해서
- 각 객체를 만들때마다 다른 메모리를 쓴다.

Instance 변수

```
<__main__.Dog object at 0x7e1bb2367c50> init 호출!

class Dog:
    def __init__(self):
        self.name = '무명'
        self.age = 0

rucy = Dog()
```



- 객체가 복사되면서 만들어지기 때문에 instance 변수라고 부른다.
- rucy = Dog()
 - 새 dog 객체를 만든다.

```

35] PPomi = Dog()
    print(PPomi.name)
    print(PPomi.age)
    PPomi.name = '뽀미'
    PPomi.age = 8
    print(PPomi.name)
    print(PPomi.age)

```

```

무명
0
뽀미
8

```

- 무명, 0 → member var,
- 뽀미, 8 → instance var

ex)

```

[37] class Dog:
      def __init__(self, name, age, family='족보없음'): #self는 모든 객체가 공통으로 사용하기 때문에 이게 누구의 이름인지
          self.name = name
          self.age = age
          self.family = family

# TypeError: Dog.__init__() missing 2 required positional arguments: 'name' and 'age'
# rucy = Dog() # 매개변수를 안넣어서 애러가 난다

rucy = Dog('루시', 15, '포메')
print(rucy)
print(rucy.name)
print(rucy.age)
print(rucy.family)

<__main__.Dog object at 0x7a443e975e90>
루시
15
포메

PPomi = Dog('뽀미', 8, '폼피츠')
print(PPomi)
print(PPomi.name)
print(PPomi.age)
print(PPomi.family)

<__main__.Dog object at 0x7a443e96fb10>
뽀미
8
폼피츠

```

- Dog class 안에서 객체 2개를 만든 상태이다 (rucy, PPomi)

```

<__main__.Dog object at 0x7e1bb236dd90>
루시
15
포메

[34] PPomi = Dog('뽀미', 8, '폼피츠')
      print(PPomi)
      print(PPomi.name)
      print(PPomi.age)
      print(PPomi.family)

<__main__.Dog object at 0x7e1bb2365e90>
뽀미
8
폼피츠

```

- 각각 객체가 따로따로 보관하기 때문에 instance 변수 2개가 현재 만들어져있다.

Class variable vs instance variable?

- **Class variable:**
 - 클래스 전체가 사용할수 있는 변수
 - 한번 만들면 각 객체마다 만들필요는 없다.
 - 객체 만들고 각 인스턴스마다 `__init__ (self)` 이지랄 필요 x
 - 한번 클래스 만들때 만들어놓으면 필요할때마다 변경만 해주면 된다.

```

# 클래스 변수(속성)
print(Dog.name)
print(Dog.age)
print(Dog.family)

```

- 만약에 인스턴스가 없으면 클래스 변수에 직접적으로 접근이 가능하다.
- **instance variable:**
 - 객체를 만든 후 → 각 상황마다 만든 / 변경된 변수

__init__(Self)란?

```
+ 코드 + 텍스트
38] class Dog:
    def __init__(self, name, age, family='족보없음'):
        self.name = name
        self.age = age
        self.family = family

# rucy = Dog() # TypeError: Dog.__init__() missing 2 re
rucy = Dog('루시', 15, '프네')
print(rucy)
print(rucy.name)
print(rucy.age)
print(rucy.family)
```

- 저장소는 2개가 별도로 만들어지지만 메서드는 공통으로 사용한다.
- 각 객체에서 정보를 init으로 보내면 누가 보냈는지 알 수 없기때문에 self라는 공간에 메모리 주소를 보냄으로서 누가 보냈는지 알 수 있고, 그 메모리 주소에 따라서 변경이 가능하다

객체지향의 은닉성 원칙:

```
def __init__(self):
    self.num = 0
def increment(self):
    self.num += 1
def current_value(self):
    return self.num
def decrement(self):
    self.num -= 1
def set_value(self, num):
    self.num = num
def reset(self):
    self.num = 0
```

- 직접 값을 변수에 대입하지 말고 특정 메서드를 이용해서 대입을 시키자.
- 값을 대입할 때는 메소드를 이용해서 대입해라 (보안 측면에서 필요하다.)
- 그냥 변수에 대입시키는거랑 뭐가 차이가나지?
 - 메소드 안에는 다양한 알고리즘을 넣을 수 있다.
 - 직접 대입을 피함으로 위험을 방지시킬 수 있다.

메서드 타입

Class method:

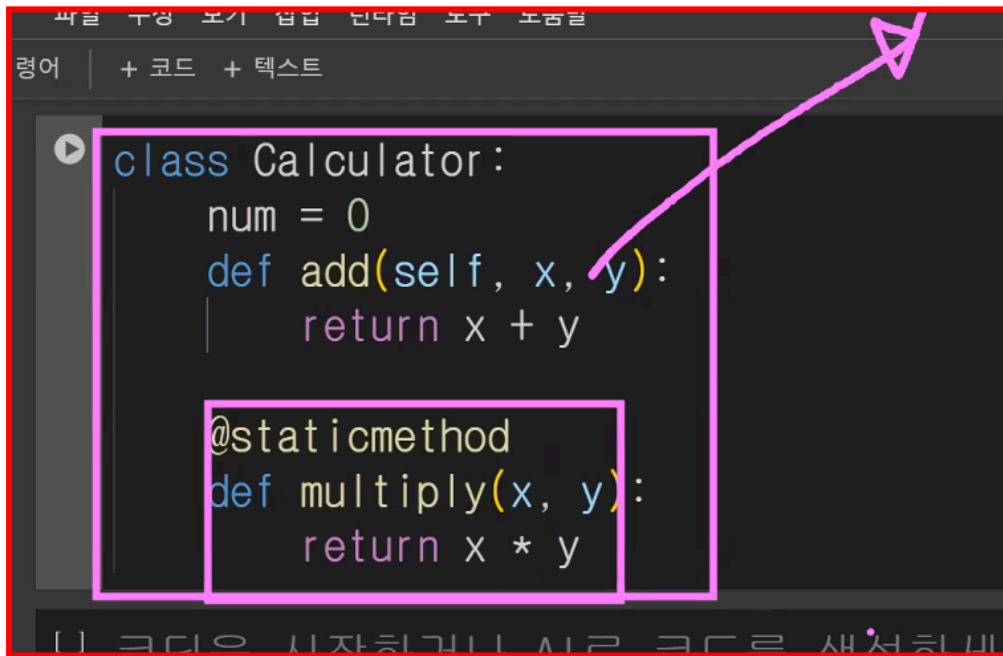
니다. 정적 메서드는 @staticmethod 네코레이터를 사용하여 정의합니다.

```
class Calculator:
    num = 0
    def add(self, x, y):
        return x + y

    @classmethod
    def subtract(cls, x, y):
        return x - y - cls.num
```

- cls는 클래스 주소를 가르킨다 (self가 객체의 주소를 가리키는 것처럼)
- 클래스 변수를 사용 가능하다.

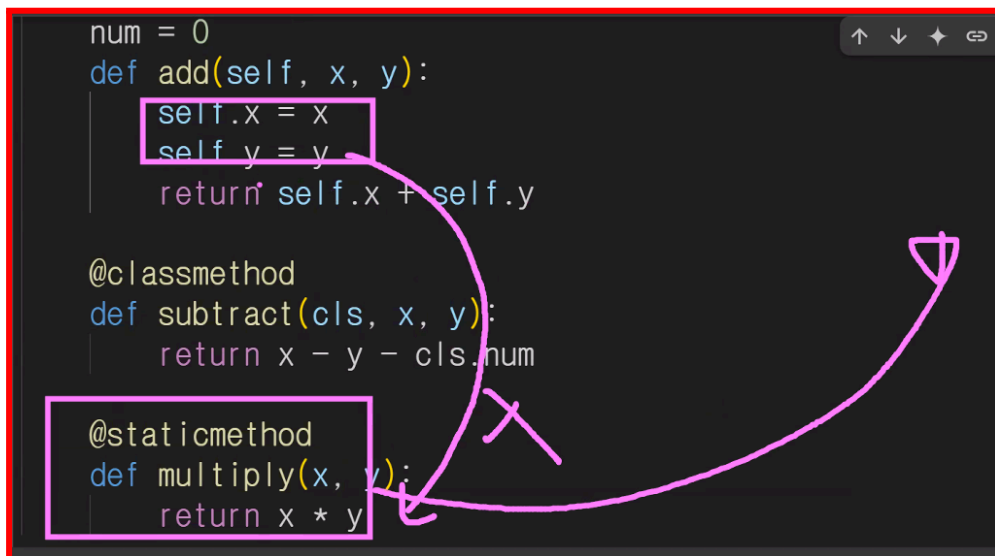
Static method:



```
class Calculator:
    num = 0
    def add(self, x, y):
        return x + y

    @staticmethod
    def multiply(x, y):
        return x * y
```

- 객체생성하기 전에는 add를 사용 못하지만
- multiply 같은 경우는 객체생성 없이 사용가능하다.



```
num = 0
def add(self, x, y):
    self.x = x
    self.y = y
    return self.x + self.y

@classmethod
def subtract(cls, x, y):
    return x - y - cls.num

@staticmethod
def multiply(x, y):
    return x * y
```

- add와 multiply는 다른 메모리 힙에 올라가기때문에 서로 통신이 되지 않는다.