Day 9 - Python Inheritance

객체지향 특징

1. 상속
2. 캡슐화
3. 다형성
4. 추상화

상속 (Inheritance)

```
class Parent:
    pass

class Child(Parent):
    pass
```

- 파이썬 상속은 기존 클래스(부모 클래스)의 속성과 메서드를 새로운 클래스(자식 클래스)가 물려받아 코드의 재사용성과 확장성을 높이며, 자식 클래스에서 부모 클래스의 기능을 수정하거나 추가할 수 있는 기능을 제공한다. 또한 파이썬의 모든 클래스는 object라는 클래스로부터 상속받습니다.
- 파이썬에서는 자식 클래스의 매개변수칸 ()가로 안에 부모 클래스를 넣어서 상속시킨다.

상속 예시)

```
[3]  class Animal:
         def __init__(self, name, age):
             self.name = name
             self.age = age

         def eat(self, food):
             print(f'{self.name} {food} 먹습니다.')

         def sleep(self, hour):
             print(f'{self.name} {hour}시간 동안 잠을 잡니다.')
```

```
     animal = Animal('동물', 10)
     animal.eat('먹이')
     animal.sleep(10)
```

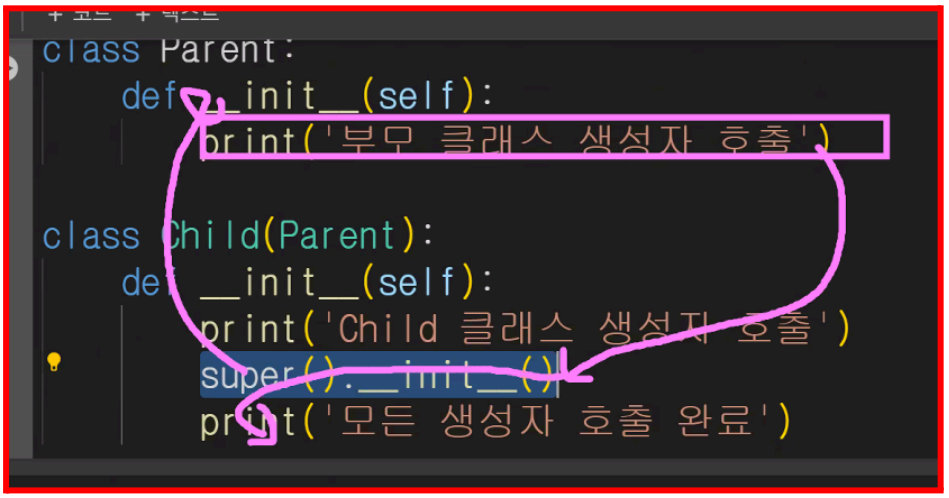
```
     동물 먹이 먹습니다.
     동물 10시간 동안 잠을 잡니다.
```

```
[8]  class Dog(Animal):
         pass            #animal을 상속한 빈 클래스를 만들어보자
```

```
[11] rucy = Dog('루시', 15)
     rucy.eat('사료')        #animal 기능을 다 가져온다.
     rucy.sleep(12)
```

- Dog 클래스는 animal 클래스를 상속 받았으므로 모든 animal class의 기능을 다 사용할 수 있다.
- Animal 클래스를 상속 받았기 때문에 animal 클래스의 생성자 매개변수를 전달해야 함(자식 클래스의 생성자가 없을 경우)

클래스 상속시 생성자 호출 순서:

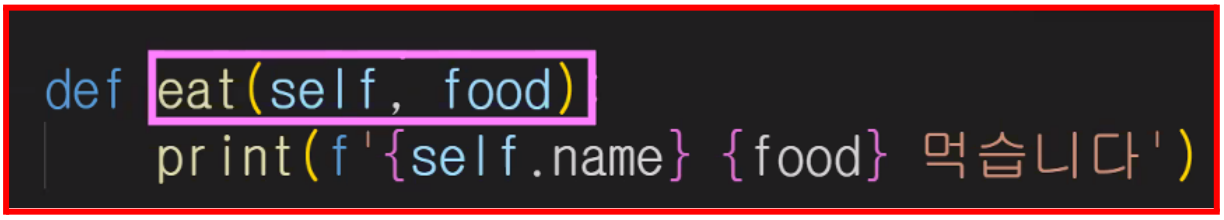


- super().__init__() 으로 부모 생성자를 호출 할 수 있다.
- 매개변수가 다르면 부모생성자를 호출시켜줘야 부모 매개변수를 사용할 수 있다.

Object 클래스:

- 파이썬에서 모든 클래스의 부모 클래스 역할을 하는 기본 클래스.
- 모든 객체는 이 클래스를 상속받는다.

메서드 오버라이딩:



함수의 매개변수와 이름이 같은걸 시그니처 라고 한다.

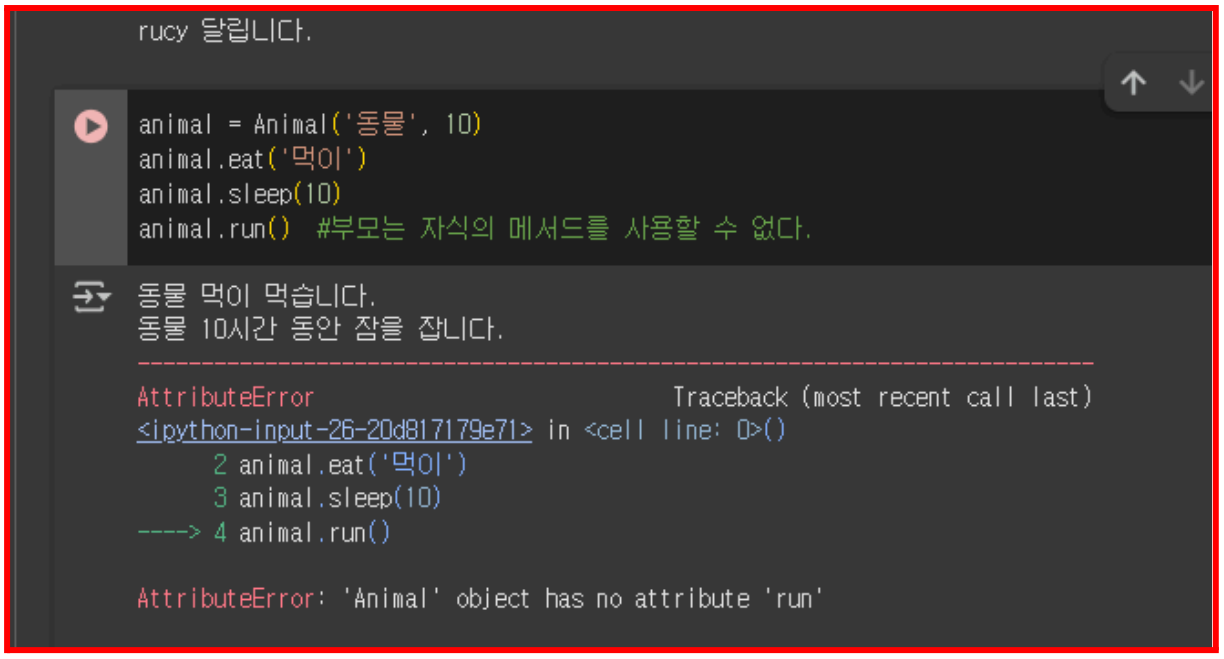
오버라이딩 예시)

```
class Animal:
    def __init__(self, name, age):
        self.name = name
        self.age = age

    def eat(self, food):
        print(f'{self.name} {food} 먹습니다.')

    def sleep(self, hour):
        print(f'{self.name} {hour}시간 동안 잠을 잡니다.')

class Dog(Animal):
    def run(self):
        print(f'{self.name} 달립니다.')

    def eat(self, food):
        print(f'{self.name} {food} 를 아주 맛있게 먹습니다.')

    def superEat(self, food):
        super().eat(food)
```

```
rucy = Dog('rucy', 15)
rucy.eat('사료') # 부모메서드를 오버라이드 한 메서드 사용가능
rucy.superEat('사료') # super()를 사용해서 부모메서드 사용가능
rucy.sleep(12)

rucy.run() # 자식 메서드 사용가능
```

```
rucy 사료 를 아주 맛있게 먹습니다.
rucy 사료 먹습니다.
rucy 12시간 동안 잠을 잡니다.
rucy 달립니다.
```

- 부모 메서드를 오버라이딩해서 사용가능하다.
- 또한 super()를 사용해서 부모 메서드도 동시에 사용가능하다.



- 부모는 자식 메서드를 사용할 수 없다.

다중 상속:

- 파이썬은 둘 이상의 부모 클래스로부터 상속받는것이 가능하다.
- 하지만 충돌이 날 수 있기때문에 주의해야한다. (복잡성이 높아진다)
- 파이썬은 상속할때 어떤 부모 클래스가 우선시되야할지 법칙이 있다.

```python
class Animal:
    def __init__(self, name, age):
        self.name = name
        self.age = age

    def eat(self, food):
        print(f'{self.name} {food} 먹습니다.')

    def sleep(self, hour):
        print(f'{self.name} {hour}시간 동안 잠을 잡니다.')

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

    def study(self, hour):
        print(f'{self.name} {hour} 동안 공부를 합니다.')

    def sleep(self, hour):
        print(f'{self.name} {hour}시간 동안 꿀잠을 잡니다.')

class Kim(Animal, Human):   #Animal 과 Human 둘다 상속받음.
    pass
```

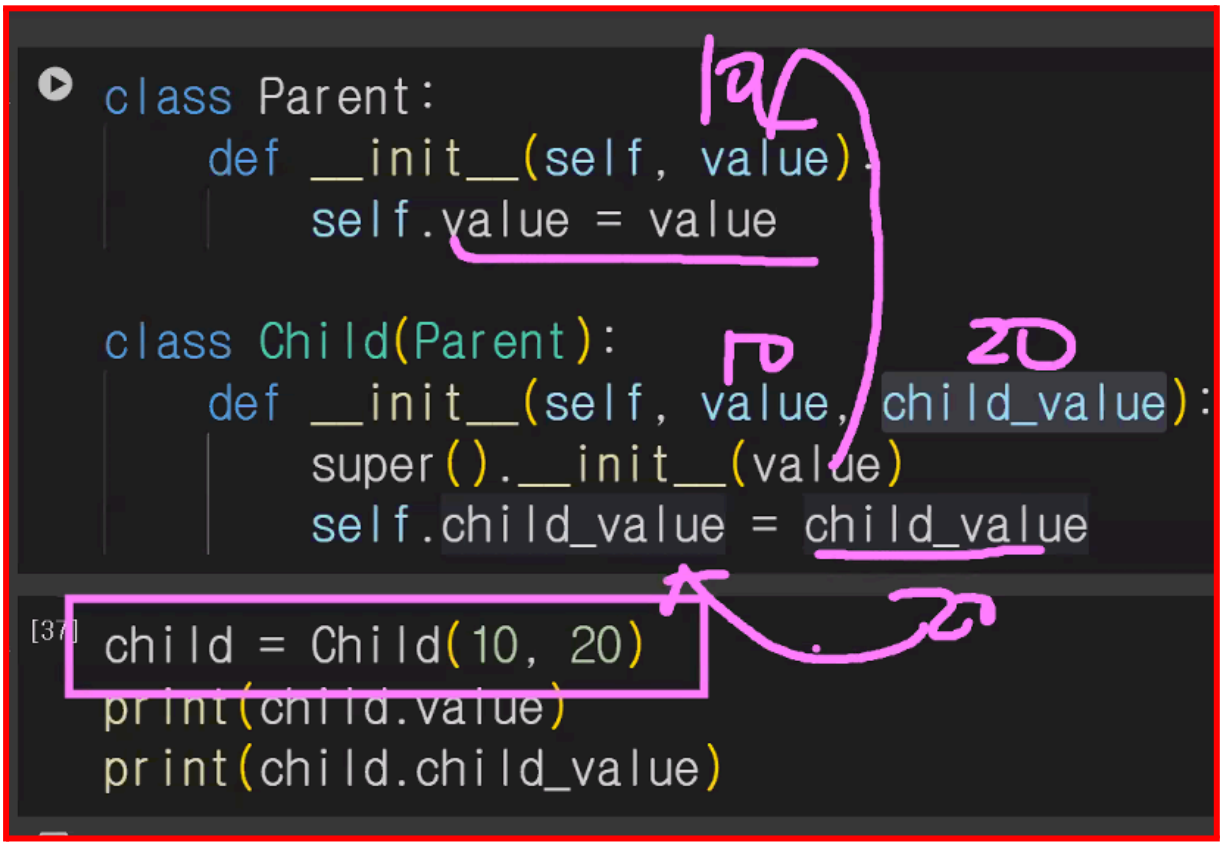
```python
kim = Kim('김사과', 20)
kim.eat('밥')   # eat은 animal 에만 있다.
kim.study(2)    # study는 human 에만 있다.
kim.sleep(8)    # sleep은 둘다 가지고 있는데 animal에서 가져온다.

print(Kim.mro()) # 상속의 순서를 반환시켜준다.
```

- MRO(Method Resolution Order)를 이용해서 상속의 순서를 알수 있다.

```
print(Kim.mro()) # 상속의 순서를 반환시켜준다.

김사과 밥 먹습니다.
김사과 2 동안 공부를 합니다.
김사과 8시간 동안 잠을 잡니다.
[<class '__main__.Kim'>, <class '__main__.Animal'>, <class '__main__.Human'>, <class 'object'>]
```

  - 이경우 Animal → Human → object 순서이다. (이 순서대로 적용된다)

super메서드:



- 부모 클래스의 변수, 생성자를 부를때 super()를 사용한다.
- 이 경우 value는 부모 클래스의 변수, child_value는 자식 클래스의 변수이다.

MRO(Method Resolution Order):

- 다중 상속을 이용할 때, 메서드나 속성을 찾는 순서를 정의하는 규칙

```
[51] class Base:
         def hello(self):
             print('Base의 hello()')
             print('Base 클래스의 hello() 메서드')

     class A(Base):
         def hello(self):
             print('A의 hello()')
             super().hello()
             print('A 클래스의 hello() 메서드')

     class B(Base):
         def hello(self):
             print('B의 hello()')
             super().hello()
             print('B 클래스의 hello() 메서드')

     class Child(A, B):
         def hello(self):
             print('Child의 hello()')
             super().hello()
             print('Child 클래스의 hello() 메서드')
```

```
child = Child()
child.hello()
```

```
Child의 hello()
A의 hello()
B의 hello()
Base의 hello()
Base 클래스의 hello() 메서드
B 클래스의 hello() 메서드
A 클래스의 hello() 메서드
Child 클래스의 hello() 메서드
```

```
Child.mro()
```

```
[__main__.Child, __main__.A, __main__.B, __main__.Base, object]
```

- Child → A → B → Base → Object 순서로 적용된다.
- Child 입장에서 생각해야한다 (A, B 둘다 부모이기때문에)
    - A 의 부모는 Base인데? 가 아니다.