OOP III

- 상속(Inheritance)
- 메서드 오버라이딩(Method Overriding)
- 다중 상속(Multiple Inheritance)

상속

상속(Inheritance)이란?

클래스에서 가장 큰 특징은 **상속** 이 가능하다는 것이다.

부모 클래스의 모든 속성이 자식 클래스에게 상속 되므로 코드 재사용성이 높아진다.

활용법

```
class ChildClass(ParentClass):
    <code block>
```

In [ ]:
```
# 인사만 할 수 있는 간단한 Person 클래스가 있습니다.
```

In [ ]:
```
class Person:
    population = 0

    def __init__(self, name='사람'):
        self.name = name
        Person.population += 1

    def talk(self):
        print(f'반갑습니다. {self.name}입니다.')
```

In [ ]:
```
# 김교수 인스턴스를 만들어봅시다.
```

In [ ]:
```
kim = Person('김교수')
```

In [ ]:
```
kim.talk()
```

In [ ]:
```
Person.population
```

In [ ]:
```
# Person 클래스를 상속받아 Student 클래스를 만들어봅시다.
```

In [ ]:
```
class Student(Person):
    def __init__(self, name, student_id):
```

```
        self.name = name
        self.student_id = student_id
```

In [ ]:
```
# 학생을 만들어봅시다.
```

In [ ]:
```
s1 = Student('박학생', '20210127')
```

In [ ]:
```
kim.name
```

In [ ]:
```
s1.name
```

In [ ]:
```
s1.student_id
```

In [ ]:
```
# 부모 클래스에 정의된 메서드를 호출 할 수 있습니다.
```

In [ ]:
```
s1.talk()
```

In [ ]:
```
Person.population
```

이처럼 상속은 공통된 속성이나 메서드를 부모 클래스에 정의하고, 이를 상속받아 다양한 형태의 사람들을 만들 수 있다.

In [ ]:
```
# 진짜 상속관계인지 확인해봅시다. (클래스 상속 검사)
```

In [ ]:
```
issubclass(Student, Person)
```

In [ ]:
```
isinstance(s1, Student)
```

In [ ]:
```
isinstance(s1, Person) # True!
```

In [ ]:
```
type(s1) is Person # False
```

타입 검사 방법

- `isinstance(3, int)` : 상속 관계에 있어도 True
- `type(3) is int` : 해당 클래스인 경우만 True

In [ ]:
```
# 내장 타입들에도 상속 관계가 있습니다.
```

In [ ]:
```
isinstance(True, int) # boolean 값이랑 int랑 비교 => True
```

In [ ]:
```
type(True) is int # False
```

In [ ]:
```
# 그 이유는 boolean은 int를 상속받아 만들어짐
issubclass(bool, int)
```

In [ ]:
```
bool.mro()
```

In [ ]:
```
float.mro()
```

super()

- 자식 클래스에 메서드를 추가로 구현할 수 있다.
- 부모 클래스의 내용을 사용하고자 할 때, `super()` 를 사용할 수 있다.

활용법

```
class ChildClass(ParentClass):
    def method(self, arg):
        super().method(arg)
```

In [ ]:
```
class Person:
    population = 0

    def __init__(self, name):
        self.name = name
        Person.population += 1

    def talk(self):
        print(f'반갑습니다. {self.name}입니다.')

class Student(Person):
    # 학생은 생성할 때, 학번을 추가로 받고 싶어요.....
    def __init__(self, name, student_id):
        # Person 하는 로직 다하고..
        self.name = name
        Person.population += 1
        # 학생거하고...불편..
        self.student_id = student_id
```

In [ ]:
```
p1 = Person('iu')
p2 = Person('jimin')
s1 = Student('kim', '202101')
s2 = Student('kim', '202102')
```

In [ ]:
```
Person.population
```

In [ ]:
```
s2.student_id
```

위의 코드를 보면, 상속을 했음에도 불구하고 동일한 코드가 반복된다.

이를 수정해보자.

In [ ]:
```
class Person:
    population = 0

    def __init__(self, name):
        self.name = name
        Person.population += 1

    def talk(self):
        print(f'반갑습니다. {self.name}입니다.')

class Student(Person):
    # 학생은 생성할 때, 학번을 추가로 받고 싶어요.....
    def __init__(self, name, student_id):
        super().__init__(name) # 여기가 실행되는 것은 부모클래스의 init()실행하고
        # 추가 작업
        self.student_id = student_id
```

[연습] Rectangle & Square

아래의 조건에 만족하는 클래스 `Rentangle` 을 작성하세요.

> Rectangle 클래스는 아래와 같은 속성과 메서드를 갖는다.
>
> - 인스턴스 속성
>   - `length` : 가로 길이
>   - `width` : 세로 길이
> - 인스턴스 메서드
>   - `area` : 직사각형의 넓이를 리턴한다.
>   - `perimeter` : 직사각형의 둘레의 길이를 리턴한다.

In [ ]:
```
# 아래에 코드를 작성하세요.
```

In [ ]:
```
class Rectangle:

    def __init__(self, length, width):
        self.length = length
        self.width = width

    def Rectangle_area(self):
        area = self.length * self.width
        print(area)

    def Rectangle_width(self):
        perimeter = (self.length + self.width) * 2
        print(perimeter)
```

In [ ]:
```
# Rectangle 클래스로부터 인스턴스를 하나 만들어 가로 길이 4, 세로 길이 8인 직사각형의 넓이와
```

In [ ]:
```
nemo = Rectangle(4,8)
```

In [ ]:
```
nemo.Rectangle_area()
```

In [ ]:
```
nemo.Rectangle_width()
```

In [ ]:
```
# Rectangle 클래스를 상속받아 Sqaure 클래스를 만들어 주세요.
# Square 클래스는 Rectangle 클래스에서 상속받은 속성 외 추가 속성을 가지고 있지 않습니다.
```

In [ ]:
```
class Square(Rectangle):

    def __init__(self, length, width):
        super().__init__(length, width)
```

In [ ]:
```
# Square 클래스로부터 인스턴스를 하나 만들어 가로/세로 길이4가 4인 직사각형의 넓이와 둘레 길
```

In [ ]:
```
momo = Square(4,4)
```

In [ ]:
```
momo.Rectangle_area()
```

In [ ]:
```
momo.Rectangle_width()
```

메서드 오버라이딩

> Method Overriding(메서드 재정의): 자식 클래스에서 부모 클래스의 메서드를 재정의하는 것

- 상속 받은 메서드를 재정의 할 수도 있다.
- 상속 받은 클래스에서 **같은 이름의 메서드**로 덮어쓴다.

In [ ]:
```
# Person 클래스의 상속을 받아 군인처럼 말하는 Soldier 클래스를 만들어봅시다.

class Person:
    def __init__(self, name, age, number, email):
        self.name = name
        self.age = age
        self.number = number
        self.email = email

    def talk(self):
        print(f'안녕, {self.name}')
```

In [ ]:
```
class Soldier(Person):
    def __init__(self, name, age, number, email, level):
        super().__init__(name, age, number, email)
        self.level = level

    def talk(self):
        if self.level == '참모총장':
            print('내밑으로 집합.')
```

```
        else:
            print(f'충성! {self.level} {self.name}입니다. ^^7')
```

In [ ]:
```
#
p = Person('일반인', 10, '010123', '1banin@gmail.com')
p.talk()
```

In [ ]:
```
goodgun2 = Soldier('굳건이', 25, '010123456', 'goodgun2@rok.kr', '이병')
```

In [ ]:
```
goodgun2.talk()
```

In [ ]:
```
star = Soldier('4스타', 50, '010123456', 'zzang@rok.kr', '참모총장')
```

In [ ]:
```
star.talk()
```

상속관계에서의 이름공간

- 기존의 인스턴스 -> 클래스 순으로 이름 공간을 탐색해나가는 과정에서 상속관계에 있으면 아래와 같이 확장된다.
- 인스턴스 -> 클래스 -> 전역
- 인스턴스 -> 자식 클래스 -> 부모 클래스 -> 전역

[연습] Person & Animal (메서드 오버라이딩)

> 사실 사람은 포유류입니다.
>
> Animal Class를 만들고, Person Class 가 상속받도록 구성해봅시다.
>
> (변수나, 메서드는 자유롭게 만들어보세요.)

In [ ]:
```
# 아래에 코드를 작성해주세요.
```

In [ ]:
```

```

다중 상속

두개 이상의 클래스를 상속받는 경우, 다중 상속이 된다.

In [ ]:
```
# Person 클래스를 정의합니다.
```

In [ ]:
```
class Person:

    def __init__(self, name):
        self.name = name
```

```
    def talk(self):
        print('사람입니다.')
```

In [ ]:
```
# Mom 클래스를 정의합니다.
```

In [ ]:
```
class Mom(Person):
    gene = 'XX'

    def swim(self):
        print('첨벙첨벙')
```

In [ ]:
```
# Dad 클래스를 정의합니다.
```

In [ ]:
```
class Dad(Person):
    gene = 'XY'

    def walk(self):
        print('씩씩하게 걷기')
```

In [ ]:
```
mommy = Mom('박엄마')
mommy.swim()
mommy.gene
```

In [ ]:
```
daddy = Dad('김아빠')
daddy.walk()
daddy.gene
```

In [ ]:
```
daddy.talk()
```

In [ ]:
```
daddy.swim()
```

In [ ]:
```
# FirstChild 클래스를 정의합니다.
```

In [ ]:
```
class FirstChild(Mom, Dad):

    def cry(self):
        print('응애')

    def walk(self):
        print('아자아장')
```

In [ ]:
```
# FirstChild 의 인스턴스 객체를 확인합니다.
```

In [ ]:
```
baby = FirstChild('이아가')
```

In [ ]:
```
# cry 메서드를 실행합니다.
```

In [ ]:
```
baby.cry()
```

In [ ]:
```
# swim 메서드를 실행합니다.
```

In [ ]:
```
baby.swim()
```

In [ ]:
```
# walk 메서드를 실행합니다.
```

In [ ]:
```
baby.walk() # baby 다시 정의 오버라이딩!
```

In [ ]:
```
# gene 은 누구의 속성을 참조할까요?
```

In [ ]:
```
baby.gene
```

In [ ]:
```
# 그렇다면 상속 순서를 바꿔봅시다.
```

In [ ]:
```
class Boy(Dad, Mom):

    def cry(self):
        print('으아아앙')
```

In [ ]:
```
# Boy 의 인스턴스 객체를 확인합니다.
```

In [ ]:
```
boy = Boy('이애기')
```

In [ ]:
```
# cry 메서드를 실행합니다.
```

In [ ]:
```
boy.cry()
```

In [ ]:
```
# walk 메서드를 실행합니다.
```

In [ ]:
```
boy.walk()
```

In [ ]:
```
# swim 메서드를 실행합니다.
```

In [ ]:
```
boy.swim()
```

In [ ]:
```
# gene 은 누구의 속성을 참조할까요?
```

In [ ]:
```
boy.gene
```

- method resolution order => 상속의 순서에 따라서 어떤 메서드를 실행할지

In [ ]:
```
Boy.mro()
```

In [ ]:
```
FirstChild.mro()
```