09

상속 및 클래스의 추가적인 구문

염 희 균

목차

시작하기 전에

[핵심 키워드] : 상속, isinstance(), 클래스 변수, 클래스 함수

[핵심 포인트]

클래스의 부가적인 기능에 대해 알아본다.

상속(inheritance)

- '물려 받다'라는 뜻
  - 어떤 클래스를 기반으로 그 속성과 기능을 물려받아 새로운 클래스 만드는 것
  - 형식

    class 클래스 이름 (상속할 클래스 이름) :

  - 예: >>> class MoreFourCal(FourCal) :
  - 상속은 기존 클래스는 그대로 놔둔 채 새로운 기능을 확장 시킬 때 주로 사용한다.

상속

```
01  # 부모 클래스를 선언합니다.
02  class Parent:
03      def __init__(Self):
04          Self.value = "테스트"
05          print("Parent 클래스의 __init()__ 메소드가 호출되었습니다.")
06      def test(self):
07          print("Parent 클래스의 test() 메소드입니다.")
08
09  # 자식 클래스를 선언합니다.
10  class Child(Parent):
11      def __init__(Self):
12          Parent.__init__(self)
13          print("Child 클래스의 __init()__ 메소드가 호출되었습니다.")
14
15  # 자식 클래스의 인스턴스를 생성하고 부모의 메소드를 호출합니다.
16  child = Child()
17  child.test()
18  print(child.value)
```

실행결과

```
Parent 클래스의 __init()__ 메소드가 호출되었습니다.
Child 클래스의 __init()__ 메소드가 호출되었습니다.
Parent 클래스의 test() 메소드입니다.
테스트
```

상속 예:

- 기존 클래스의 속성과 메소드를 그대로 물려 받는다.
- class AttackUnit (공격 유닛) : 일반 유닛 클래스를 상속 받는다.

```
class Unit:
    def __init__(self, name, hp):
        self.name = name # 멤버 변수
        self.hp = hp
```

```
class AttackUnit(Unit):
    def __init__(self, name, hp, damage) :
        Unit.__init__(self, name, hp)
        self.damage = damage
```

다중 상속

- 2개의 클래스의 기능을 상속 받을 수 있다.
  - 형식

```
class 클래스 이름 (상속할 클래스 이름1, 상속할 클래스이름2 ) :
```

다중 상속 예

- FlyableAttackUnit
  - class Flyable :드랍쉽(공중 유닛, 공격력은 없음)
  - class AttackUnit

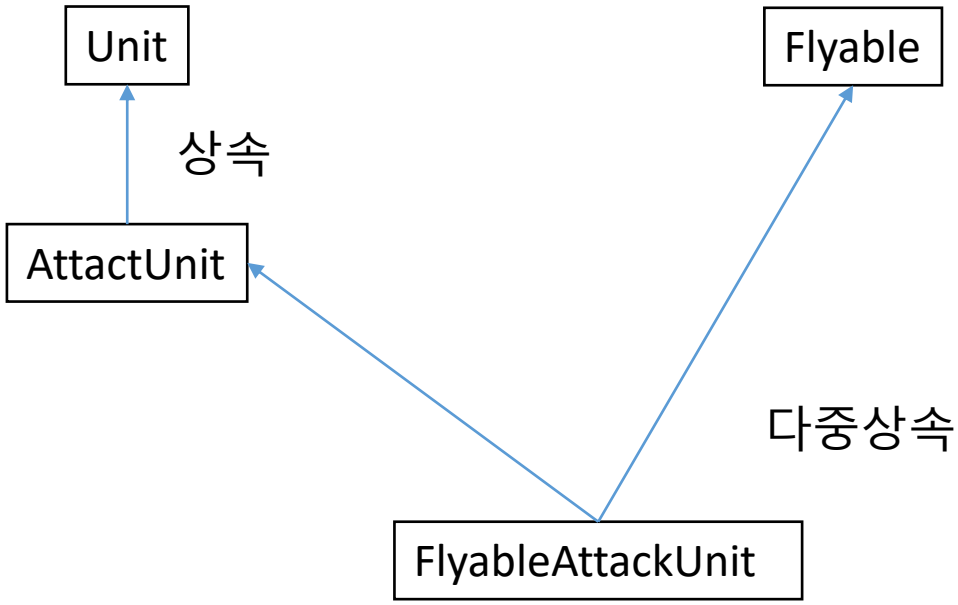
```
class Flyable:
    def __init__(self, flying_speed) :
        self.flying_speed = flying_speed

    def fly(self, name, location):
        print("{} : {} 방향으로 날아갑니다. [속도 : {}]"\
            .format(name, location, self.flying_speed))


class FlyableAttackUnit(AttackUnit, Flyable):
    def __init__(self, name, hp, damage, flying_speed):
        AttackUnit.__init__(self, name, hp, damage)
        Flyable.__init__(self,flying_speed)

# 발키리 : 공중 공격유닛, 한번에 14발 미사리 발사
valkyrie = FlyableAttackUnit("발키리", 200, 6, 5)
valkyrie.fly(valkyrie.name, "3시")
```

다중 상속



메소드 오버라이딩

- 자식 클래스에서 부모 클래스의 메소드를 다시 정의하는 것

```
class Unit:
    def __init__(self, name, hp, speed):
        self.name = name # 멤버 변수
        self.hp = hp
        self.speed = speed

     def move(self, location):
            print("[지상 유닛 이동]")
            print("{} : {} 방향으로 이동합니다. [속도: {}]"\
                .format(self.name, location, self.speed) )
```

```
class AttackUnit(Unit):
    def __init__(self, name, hp, speed, damage) :
        Unit.__init__(self, name, hp, speed)
        self.damage = damage
```

메소드 오버라이딩

```
class Flyable:
    def __init__(self, flying_speed) :
        self.flying_speed = flying_speed

    def fly(self, name, location):
        print("{} : {} 방향으로 날아갑니다. [속도 : {}]"\
            .format(name, location, self.flying_speed))


class FlyableAttackUnit(AttackUnit, Flyable):
    def __init__(self, name, hp, damage, flying_speed):
        AttackUnit.__init__(self, name, hp, 0, damage) # 지상 speed=0 으로 처리

        Flyable.__init__(self,flying_speed)

     def move(self, location):
        print("[공중 유닛 이동]")
        self.fly(self.name, location)
```

메소드 오버라이딩

```
# 벌처 유닛 : 지상 유닛, 기동성이 좋음
vulture = AttackUnit("벌처", 80, 10, 20)

# 배틀크루저: 공중 유닛, 체력도 좋음, 공격력도 좋음
battlecuiser = FlyableAttackUnit("배틀크루저",500, 25,3)

vulture.move("11시")
# battlecuiser.fly(battlecuiser.name, "9시")
# 재정의(오버라이딩) 한 메소드 호출
battlecuiser.move("9시")
```

어떤 클래스의 인스턴스인지 확인하기

- isinstance() 함수
  - 객체가 어떤 클래스로부터 만들어졌는지 확인

```
isinstance(인스턴스, 클래스)
```

```
# 클래스를 선언합니다.
class Student:
    def __init__(self):
        pass

# 학생을 선언합니다.
student = Student()

# 인스턴스 확인하기
print("isinstance(student, Student):", isinstance(student, Student))
```

```
isinstance(students[0], Student): True
```

어떤 클래스의 인스턴스인지 확인하기

- isinstance() 함수의 다양한 활용
  - 예시 – 리스트 내부에 여러 종류의 인스턴스 들어있을 때, 인스턴스들을 구분하며 속성과 기능 사용

```
01  # 학생 클래스를 선언합니다.
02  class Student:
03      def study(self):
04          print("공부를 합니다.")
05
06  # 선생님 클래스를 선언합니다.
07  class Teacher:
08      def teach(self):
09          print("학생을 가르칩니다.")
10
11  # 교실 내부의 객체 리스트를 생성합니다.
12  classroom = [Student(), Student(), Teacher(), Student(), Student()]
13
14  # 반복을 적용해서 적절한 함수를 호출하게 합니다.
15  for person in classroom:
16      if isinstance(person, Student):
17          person.study()
18      elif isinstance(person, Teacher):
19          person.teach()
```

실행결과

```
공부를 합니다.
공부를 합니다.
학생을 가르칩니다.
공부를 합니다.
공부를 합니다.
```

클래스 변수와 메소드

- 클래스 변수
  - class 구문 바로 아래의 단계에 변수를 선언

```
class 클래스 이름:
    클래스 변수 = 값
```

  - 클래스 변수에 접근

```
클래스 이름.변수 이름
```

클래스 변수와 메소드

- 활용 예시

```
01  # 클래스를 선언합니다.
02  class Student:
03      count = 0
04
05      def __init__(self, name, korean, math, english, science):
06          # 인스턴스 변수 초기화
07          self.name = name
08          self.korean = korean
09          self.math = math
10          self.english = english
11          self.science = science
12
13          # 클래스 변수 설정
```

실행결과

```
1번째 학생이 생성되었습니다.
2번째 학생이 생성되었습니다.
3번째 학생이 생성되었습니다.
4번째 학생이 생성되었습니다.
5번째 학생이 생성되었습니다.
6번째 학생이 생성되었습니다.

현재 생성된 총 학생 수는 6명입니다.
```

클래스 변수와 메소드

```
14            Student.count += 1
15            print("{}번째 학생이 생성되었습니다.".format(Student.count))
16
17    # 학생 리스트를 선언합니다.
18    students = [
19        Student("윤인성", 87, 98, 88, 95),
20        Student("연하진", 92, 98, 96, 98),
21        Student("구지연", 76, 96, 94, 90),
22        Student("나선주", 98, 92, 96, 92),
23        Student("윤아린", 95, 98, 98, 98),
24        Student("윤명월", 64, 88, 92, 92)
25    ]
26
27    # 출력합니다.
28    print()
29    print("현재 생성된 총 학생 수는 {}명입니다.".format(Student.count))
```

클래스 내부와 외부에서 클래스 변수에 접근할 때는 모두 Student.count 형태(클래스이름.변수이름)를 사용합니다.

클래스 변수와 메소드

- 클래스 함수
  - 클래스가 가진 함수
  - '클래스가 가진 기능' 명시적으로 나타냄
  - 데코레이터 (decorator) : @classmethod

클래스 함수 만들기

```
class 클래스 이름:
    @classmethod
    def 클래스 함수(cls, 매개변수):
        pass
```

클래스 함수 호출하기

```
클래스 이름.함수 이름(매개변수)
```

클래스 변수와 메소드

- 활용 예시 – Student.print()

```
01  # 클래스를 선언합니다.
02  class Student:
03      # 클래스 변수
04      count = 0
05      students = []
06
07      # 클래스 함수
08      @classmethod
09      def print(cls):
10          print("------ 학생 목록 ------")
11          print("이름\t총점\t평균")
12          for student in cls.students:
13              print(str(student))
14          print("------- ------- -------")
15
```

Student.students라고 해도 상관없지만,
여기서는 매개변수로 받은 cls를 활용합니다.

클래스 변수와 메소드

```
16      # 인스턴스 함수
17      def __init__(self, name, korean, math, english, science):
18          self.name = name
19          self.korean = Korean
20          self.math = math
21          self.english = English
22          self.science = science
23          Student.count += 1
24          Student.students.append(self)
25
26      def get_sum(self):
27          return self.korean + self.math +\
28              self.english + self.science
29
30      def get_average(self):
```

클래스 변수와 메소드

```
31          return self.get_sum() / 4
32
33      def __str__(self):
34          return "{}\t{}\t{}".format(\
35              self.name,\
36              self.get_sum(),\
37              self.get_average())
38
39  # 학생 리스트를 선언합니다.
40  Student("윤인성", 87, 98, 88, 95)
41  Student("연하진", 92, 98, 96, 98)
42  Student("구지연", 76, 96, 94, 90)
43  Student("나선주", 98, 92, 96, 92)
44  Student("윤아린", 95, 98, 98, 98)
45  Student("윤명월", 64, 88, 92, 92)
46  Student("김미화", 82, 86, 98, 88)
47  Student("김연화", 88, 74, 78, 92)
48  Student("박아현", 97, 92, 88, 95)
49  Student("서준서", 45, 52, 72, 78)
50
51  # 현재 생성된 학생을 모두 출력합니다.
52  Student.print()
```

실행결과

```
------ 학생 목록 ------
이름        총점        평균
윤인성      368      92.0
연하진      384      96.0
구지연      356      89.0
나선주      378      94.5
윤아린      389      97.25
윤명월      336      84.0
김미화      354      88.5
김연화      332      83.0
박아현      372      93.0
서준서      247      61.75
------  ------  ------
```

키워드로 정리하는 핵심 포인트

- isinstance() : 어떤 클래스의 인스턴스인지 확인할 때 사용하는 함수

- **클래스 변수, 클래스 함수** : 클래스 이름 뒤에 마침표 찍고 바로 사용할 수 있는 클래스가 갖는 변수와 함수

- **상속** : 어떤 클래스 기반으로 그 속성과 기능을 물려받아 새로운 클래스 만드는 것

확인문제

- 슬라이드 #10 코드의 compare_func.py를 수정해서 Student 객체를 숫자와 비교했을 때 학생의 성적 평균과 비교가 일어나게 해보세요.

```
test = Student("A", 90, 90, 90, 90)
print(test == 90)           # → True
print(test != 90)           # → False
print(test >  90)           # → False
print(test >= 90)           # → True
print(test <  90)           # → False
print(test <= 90)           # → True
```

```
# 클래스를 선언합니다.
class Student:
    def __init__(self, name, korean, math, english, science):
        self.name = name
        self.korean = korean
        self.math = math
```

확인문제

```
        self.english = english
        self.science = science

    def get_sum(self):
        return self.korean + self.math +\
            self.english + self.science

    def get_average(self):
        return self.get_sum() / 4

    def __    __(self, value):
        return self.get_average()
    def __    __(self, value):
        return self.get_average()
    def __    __(self, value):
        return self.get_average()
    def __    __(self, value):
        return self.get_average()
    def __    __(self, value):
        return self.get_average()
    def __    __(self, value):
        return self.get_average()
```

확인문제

```
# 학생을 선언합니다.
test = Student("A", 90, 90, 90, 90)

# 출력합니다.
print("test == 90:", test == 90)
print("test != 90:", test != 90)
print("test >  90:", test >  90)
print("test >= 90:", test >= 90)
print("test <  90:", test <  90)
print("test <= 90:", test <= 90)
```