Unit 36.

Status	완료
담당자	
마감일	
완료일	@2022년 11월 30일

Unit 36. 클래스 속성 사용하기

- 클래스 상속은 물려받은 기능을 유지한채로 다른 기능을 추가할 때 사용하는 기능
- 기능을 물려주는 클래스를 기반 클래스(base class), 상속을 받아 새롭게 만드는 클래스를 파생 클래스(derived class)라고 한다.
- 보통 기반 클래스는 부모 클래스(parent class), 슈퍼 클래스(superclass)라고 부르고, 파생 클래스는 자식 클래스(child class), 서브 클래스(subclass)라고도 부른다.

36.1 사람 클래스로 학생 클래스 만들기

- 클래스 상속은 클래스를 만들 때 ( )괄호를 붙이고 안에 기반 클래스 이름을 넣는다.

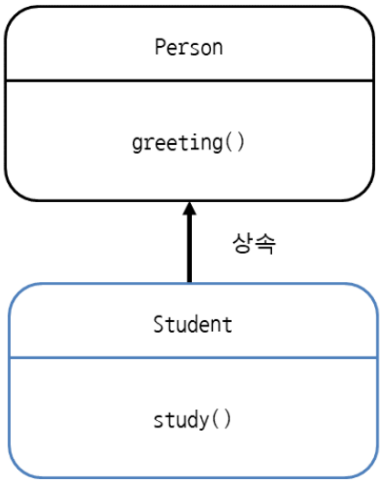
```
class 기반클래스이름:
    코드

class 파생클래스이름(기반클래스이름):
    코드
```

```
class Person:
    def greeting(self):
        print('안녕하세요.')

class Student(Person):
    def study(self):
        print('공부하기')

james = Student()
james.greeting()    # 안녕하세요.: 기반 클래스 Person의 메서드 호출
james.study()       # 공부하기: 파생 클래스 Student에 추가한 study 메서드
```

class Student(Person):과 같이 ( )(괄호) 안에 기반 클래스인 Person 클래스를 넣었다. 이렇게 하면 Person 클래스의 기능을 물려받은 Student 클래스가 된다.

Student 클래스에는 greeting 메서드가 없지만 Person 클래스를 상속받았으므로 greeting 메서드를 호출할 수 있다.

Student 클래스에 추가한 새로운 메서드인 study를 호출했다.



36.2 상속 관계와 포함 관계 알아보기

36.2.1 상속 관계

- 상속은 명확하게 같은 종류이며 동등한 관계일 때 사용합니다.

36.2.2 포함 관계

- 학생 한명의 클래스가 아니라 사람 목록을 관리하는 클래스를 만들때는 리스트 속성에 person 인스턴스를 넣어서 관리하면 된다.

```
class Person:
    def greeting(self):
        print('안녕하세요.')

class PersonList:
    def __init__(self):
        self.person_list = []    # 리스트 속성에 Person 인스턴스를 넣어서 관리

    def append_person(self, person):    # 리스트 속성에 Person 인스턴스를 추가하는 함수
        self.person_list.append(person)
```

36.3 기반 클래스의 속성 사용하기

기반 클래스에 들어있는 인스턴스 속성을 사용

Person 클래스에 hello 속성이 있고, Person 클래스를 상속받아 Student 클래스를 만든다.

```
class Person:
    def __init__(self):
        print('Person __init__')
        self.hello = '안녕하세요.'

class Student(Person):
    def __init__(self):
        print('Student __init__')
        self.school = '파이썬 코딩 도장'

james = Student()
print(james.school)
print(james.hello)    # 기반 클래스의 속성을 출력하려고 하면 에러가 발생함
```

기반 클래스 Person의 **init** 메서드가 호출되지 않았기 때문에 에러가 발생한다.

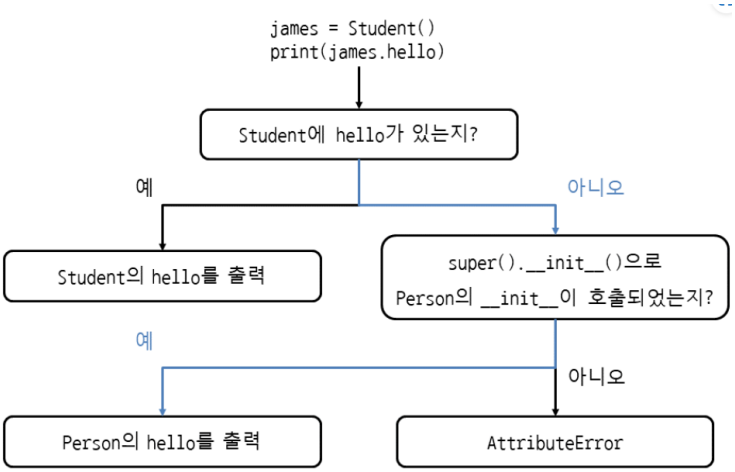
36.3.1 super()로 기반 클래스 초기화하기

이때는 super()를 사용해서 기반 클래스의 **init** 메서드를 호출해준다.

super() 뒤에 .(점)을 붙여서 메서드를 호출

```
class Person:
    def __init__(self):
        print('Person __init__')
        self.hello = '안녕하세요.'

class Student(Person):
    def __init__(self):
        print('Student __init__')
        super().__init__()                # super()로 기반 클래스의 __init__ 메서드 호출
        self.school = '파이썬 코딩 도장'

james = Student()
print(james.school)
print(james.hello)
```

super().**init**()와 같이 기반 클래스 Person의 **init** 메서드를 호출해주면 기반 클래스가 초기화되어서 속성이 만들어진다.



36.3.2 기반 클래스를 초기화하지 않아도 되는 경우

파생 클래스에서 **init** 메서드를 생략한다면 기반 클래스의 __init__이 자동으로 호출되므로 super()는 사용하지 않아도 된다.

```
class Person:
    def __init__(self):
        print('Person __init__')
        self.hello = '안녕하세요.'

class Student(Person):
    pass

james = Student()
print(james.hello)
```

36.4 메서드 오버라이딩 사용하기

메서드 오버라이딩이란 파생 클래스에서 기반 클래스의 메서드를 새로 정의하는 메서드

```
class Person:
    def greeting(self):
        print('안녕하세요.')

class Student(Person):
    def greeting(self):
        print('안녕하세요. 저는 파이썬 코딩 도장 학생입니다.')

james = Student()
james.greeting()
```

오버라이딩(overriding)은 무시하다, 우선하다라는 뜻을 가지고 있는데 말 그대로 기반 클래스의 메서드를 무시하고 새로운 메서드를 만든다는 뜻

여기서는 Person 클래스의 greeting 메서드를 무시하고 Student 클래스에서 새로운 greeting 메서드를 만들었다.

위와 같이 파생 클래스와 기반 클래스가 중복될 때는 super( )로 기반 클래스의 메서드를 호출

```
class Person:
    def greeting(self):
        print('안녕하세요.')

class Student(Person):
    def greeting(self):
        super().greeting()    # 기반 클래스의 메서드 호출하여 중복을 줄임
        print('저는 파이썬 코딩 도장 학생입니다.')

james = Student()
james.greeting()
```

이처럼 메서드 오버라이딩은 원래 기능을 유지하면서 새로운 기능을 덧붙일 때 사용한다.

36.5 다중 상속 사용하기

다중 상속은 여러 기반 클래스로부터 상속을 받아서 파생 클래스를 만드는 방법

클래스를 만들 때 ( )(괄호) 안에 클래스 이름을 ,(콤마)로 구분해서 넣는다.

```
class 기반클래스이름1:
    코드

class 기반클래스이름2:
    코드

class 파생클래스이름(기반클래스이름1, 기반클래스이름2):
    코드
```

```
class Person:
    def greeting(self):
        print('안녕하세요.')

class University:
    def manage_credit(self):
        print('학점 관리')

class Undergraduate(Person, University):
    def study(self):
        print('공부하기')

james = Undergraduate()
james.greeting()         # 안녕하세요.: 기반 클래스 Person의 메서드 호출
james.manage_credit()    # 학점 관리: 기반 클래스 University의 메서드 호출
james.study()            # 공부하기: 파생 클래스 Undergraduate에 추가한 study 메서드
```

class Undergraduate(Person, University):와 같이 괄호 안에 Person과 University를 콤마로 구분해서 넣었습니다. 이렇게 하면 두 기반 클래스의 기능을 모두 상속받는다.

36.5.1 다이아몬드 상속

```
class A:
    def greeting(self):
        print('안녕하세요. A입니다.')

class B(A):
    def greeting(self):
        print('안녕하세요. B입니다.')

class C(A):
    def greeting(self):
        print('안녕하세요. C입니다.')

class D(B, C):
    pass

x = D()
x.greeting()    # 안녕하세요. B입니다.
```

이걸 잘 보면 b와 c는 a를 상속받고 d는 a를 상속받은 b와 c를 상속받는다 이러면 어떨 때는 b를 어떨 때는 c의 메서드를 호출하는 큰 문제가 있다 다이아몬드 상속에는

36.5.2 메서드 탐색 순서 확인하기

이런 문제 때문에 다이아몬드 상속에 대한 해결책을 제시하고 있는데 파이썬에서는 메서드 탐색 순서를 따른다.

클래스 D에 메서드 mro를 사용해보면 메서드 탐색 순서가 나온다.

• 클래스.mro()

```
>>> D.mro()
[<class '__main__.D'>, <class '__main__.B'>, <class '__main__.C'>, <class '__main__.A'>, <class 'object'>]
```

36.6 추상 클래스 사용하기

추상 클래스는 메서드의 목록만 가진 클래스이며 상속받는 클래스에서 메서드 구현을 강제하기 위해 사용한다.

추상 클래스를 만들려면 import로 abc모듈을 가져와야 한다

클래스의 ( )(괄호) 안에 metaclass=ABCMeta를 지정하고, 메서드를 만들 때 위에 @abstractmethod를 붙여서 추상 메서드로 지정한다.

```
from abc import *

class 추상클래스이름(metaclass=ABCMeta):
    @abstractmethod
    def 메서드이름(self):
        코드
```

```
from abc import *

class StudentBase(metaclass=ABCMeta):
    @abstractmethod
    def study(self):
        pass

    @abstractmethod
    def go_to_school(self):
        pass

class Student(StudentBase):
    def study(self):
        print('공부하기')

james = Student()
james.study()
```

위의 코드를 실행하면 에러가 발생한다 그 이유는 추상 클래스 StudentBase에서는 추상 메서드로 study와 go_to_school을 정의했다. 하지만 StudentBase를 상속받은 Student에서는 study 메서드만 구현하고, go_to_school 메서드는 구현하지 않았으므로 에러가 발생한다.

추상 클래스를 상속받았다면 @abstractmethod가 붙은 추상 메서드를 모두 구현해야한다

```
from abc import *

class StudentBase(metaclass=ABCMeta):
    @abstractmethod
    def study(self):
        pass

    @abstractmethod
    def go_to_school(self):
        pass

class Student(StudentBase):
    def study(self):
        print('공부하기')

    def go_to_school(self):
        print('학교가기')

james = Student()
james.study()
james.go_to_school()
```