CHAPTER 7.
객체 지향 프로그래밍

section107. 객체(object)

step 1.
알고가기

- 핵심내용
  - 객체는 우리가 보고 생각할 수 있는 모든 것을 말한다.

- 코딩하기 전에
  - 객체 지향 프로그래밍은 객체(object, '오브젝트'라고 읽음)를 '지향'하는 또는 '중심으로'하는 프로그래밍이라는 의미
  - 객체에는 아빠, 엄마, 강아지, 옷, 책상, 자동차, 집, 공원에 존재하는 모든 것, 그리고 수업, 사랑처럼 눈에 보이지 않는 것까지 모두 포함한다.

section107. 객체(object)

step 2.
코딩

```
1:  # section_107.py
2:
3:  name = '휴보'
4:  weight = 45
5:
6:  def speak():
7:      print('안녕하세요. 휴보입니다.')
8:
9:  def move():
10:     print('휴보가 이동한다')
```

section107. 객체(object)

step 3.
코드분석

- 객체를 만드는 방법
  1. 어떤 객체를 만들지 선택한다.
  2. 해당 객체를 특성과 기능으로 나타낸다.
     - 객체의 특성: 현실 객체의 고정적인 요소들이나 상태. 예를 들면, 색, 크기, 온도, 키, 몸무게 등등을 말한다.
     - 객체의 기능: 현실 객체의 움직임과 관련된 것들. 예를 들면, 걷다, 뛰다, 날다, 먹다, 잠자다, 회전하다 등등을 말한다.
  3. 객체의 특성은 변수에 저장하고, 기능은 함수로 구현한다.

section107. 객체(object)

step 3.
코드분석

- 객체를 만들어 보자.
  1. 어떤 객체?: 로봇
  2. 로봇 객체를 특성과 기능으로 나타낸다.
     - 객체의 특성: 제조사, 제조번호, 모델명, 이름, 몸무게 등이 있다. 이름은 '휴보', 몸무게는 45로 설정하기로 한다.
     - 객체의 기능: 말하다, 이동하다, 충전하다 등으로 분석할 수 있다. 이 중 말하다, 이동하다 기능만 구현하기로 한다.
  3. 로봇의 특성을 변수로, 기능은 함수로 구현한다.
     - name = '휴보', weight = 45
     - def speak():, def move():
- 객체 지향 프로그래밍에서는 변수를 '속성(attribute)'이라고 부르고, 함수를 '메서드(method)'라고 부른다.

section107. 객체(object)

step 4.

3줄 요약

• 3줄 요약 •

- 객체는 우리가 보고 느끼고 생각할 수 있는 모든 것을 말한다.
- 현실의 객체를 구현하기 위해서 우선 특성과 기능으로 분석한다.
- 객체의 특성은 변수(속성)로, 기능은 함수(메서드)로 구현한다.

section108. 클래스(class)

step 1.
알고가기

- 핵심내용
  - 클래스는 객체의 원형이다.

- 코딩하기 전에
  - 객체를 코드로 완성하기 위해서는 class 단위로 묶어준다.
  - 객체와 클래스 개념을 구분해야 한다.

section108. 클래스(class)

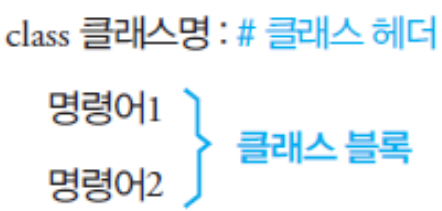
step 2.
코딩

```
1:  # section_108.py
2:
3:  class Robot:
4:      name = '휴보'
5:      weight = 45
6:
7:      def speak(self):
8:          print('안녕하세요. 휴보입니다.')
9:
10:     def move(self):
11:         print('휴보가 이동한다.')
12:
13: robot1 = Robot()
14: robot2 = Robot()
15: robot3 = Robot()
16:
17: print(type(robot1))
18: robot2.speak()
```

```
<class '__main__.Robot'>
안녕하세요. 휴보입니다.
```

section108. 클래스(class)

step 3.
코드분석

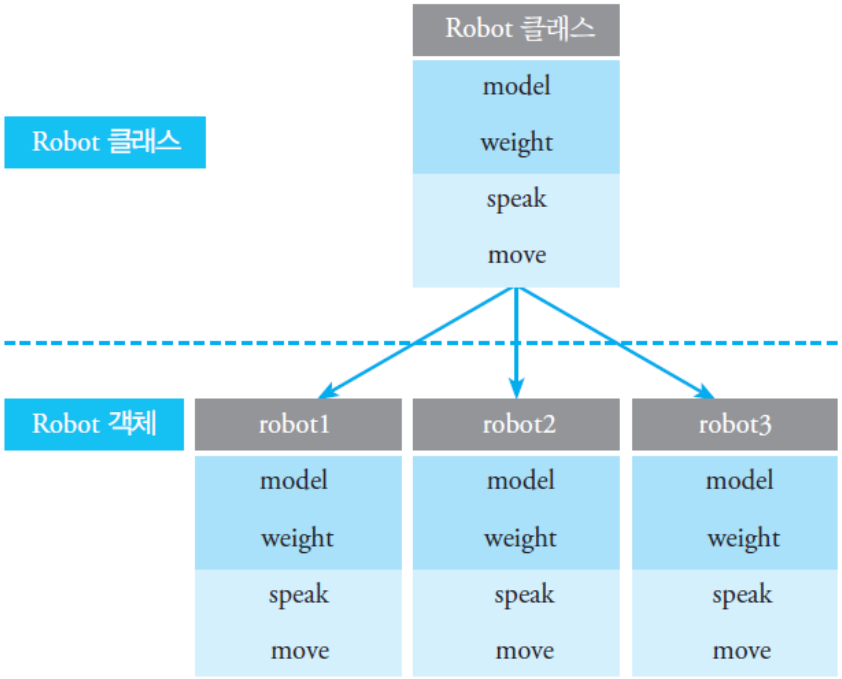


- class Robot:
  - 클래스는 속성과 메서드를 담는 큰 틀이다.
  - 클래스는 클래스 헤더와 클래스 블록으로 구성된다.
  - Robot은 클래스 이름이다. 보통은 클래스 이름의 첫 문자를 대문자로 한다.
- robot1 = Robot()
  - 3번~11번 줄에 거쳐 클래스를 정의했지만 우리는 설계도만 완성한 것이다.
  - Robot 객체를 저장할 변수 robot1을 만들고 등호의 우변에다 Robot()이라고 써줌으로써 로봇 객체 하나를 생성한 것이다.
  - 로봇 객체 robot1을 Robot 클래스의 인스턴스(instance)라고도 한다.

section108. 클래스(class)

step 3.
코드분석

- 설계도인 Robot 클래스를 이용해 무수히 많은 로봇 객체를 만들 수 있다.



[robot1, robot2, robot3는 Robot 객체이면서 Robot 클래스의 인스턴스이다.]

- 이렇게 만들어진 각 객체들은 모두 Robot클래스의 모든 속성과 메서드를 갖게 된다.

section108. 클래스(class)

step 3.
코드분석

- type(robot1)
  - <class '__main__.Robot'>
  - robot1은 Robot 클래스의 인스턴스라는 것을 확인할 수 있다
- robot2.speak()
  - robot2가 가지고 있는 speak() 메서드를 사용해본 것이다.
  - 객체에 부여된 기능이 잘 작동되는 것을 확인할 수 있다.

section108. 클래스(class)

step 4.

3줄 요약

• 3줄 요약 •

- 객체를 만들기 위해서는 클래스를 구현해야 한다.
- 하나의 클래스로 무수히 많은 객체를 만들어 낼 수 있다.
- 객체 지향 프로그래밍은 객체와 객체가 메시지를 주고받으며 상호 작용하면서 작동하도록 코딩하는 것을 말한다.

section109. 파이썬은 객체로 구성된다.

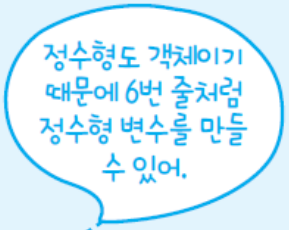
step 1.
알고가기

- 핵심내용
  - 파이썬은 모든 요소가 객체이다.

section109. 파이썬은 객체로 구성된다.

step 2.
코딩

```
1:  # section_109.py
2:
3:  a = 10
4:  print(type(a))
5:
6:  b = int(10)
7:  print(type(b))
8:
9:  c = [1, 2, 3, 4, 5]
10: print(type(c))
11:
12: def func(x):
13:     return x + 1
14: print(type(func))
15:
16: import math
17: print(type(math))
```




```
<class 'int'>
<class 'int'>
<class 'list'>
<class 'function'>
<class 'module'>
```

section109. 파이썬은 객체로 구성된다.

step 3.
코드분석

- a = 10
  - 변수 a에 정수형 데이터 10을 저장한다.
  - <class 'int'>는 변수 a가 정수형을 정의한 int class의 인스턴스임을 의미한다.
  - 즉, 변수도 객체임을 알 수 있다.
- b = int(10)
  - 변수도 객체이므로 객체 생성 방법과 같은 방법으로 만들 수 있다.
- 사용자가 새로운 클래스를 만들어 새로운 자료형을 만들 수도 있다.
- 리스트, 함수, 모듈 등 모든 요소는 파이썬 내부에 이미 클래스로 정의되어 있다.
- 따라서 우리가 리스트, 함수, 모듈을 만들때마다 객체가 생성되는 것이라고 볼 수 있다.

section109. 파이썬은 객체로 구성된다.

step 4.

3줄 요약

• 3줄 요약 •

- 파이썬의 모든 요소는 객체이다.
- 〈class 'int'〉는 'int 객체'라는 의미이다.
- 새로운 클래스를 정의하면 새로운 자료형이 생긴 것과 같다.

section110. 가장 간단한 클래스

step 1.
알고가기

- 핵심내용
  - 클래스를 구성하는 변수를 속성(attribute)이라고 한다.

- 코딩하기 전에
  - 구현하고 싶어하는 현실 객체를 특성과 기능으로 분석한 다음 디지털 객체로 구현한다.
  - 이때 현실 객체의 특성은 디지털 객체에서 변수로 구현되며 속성이라고 부른다.

section110. 가장 간단한 클래스

step 2.
코딩

```
1:  # section_110.py
2:
3:  class Robot:
4:    '''가장 심플한 클래스'''
5:      pass
6:
7:  robot1 = Robot()
8:  robot1.name = 'Hubo 2'
9:  robot1.weight = '45 Kg'
10:
11: robot2 = Robot()
12: robot2.name = 'Hubo 2 plus'
13: robot2.build_year = '2011년'
14:
15: print(robot1.name, '-', robot1.weight)
16: print(robot2.name, '-', robot2.build_year)
```

```
Hubo 2 - 45 Kg
Hubo 2 plus - 2011년
```

section110. 가장 간단한 클래스

step 3.
코드분석

- class Robot:
- Robot 클래스를 정의하는 헤더
- robot1 = Robot()
  - Robot 클래스는 설계도일 뿐 아무런 일을 할 수 없다.
  - Robot클래스를 사용하려면 객체를 만들어야 한다. robot1이 Robot 클래스의 인스턴스이자 객체이다.
- robot1.name = 'Hubo 2'
  - 객체를 만들고 난 후 속성을 추가할 수 있다. 즉, 변수를 추가할 수 있다.
- robot1과 robot2는 각각 독립된 객체이다.

section110. 가장 간단한 클래스

step 4.

3줄 요약

• 3줄 요약 •

- 객체의 속성을 추가할 때는 점(.)으로 연결한다.
- 클래스로부터 만들어진 객체들은 독립적인 개체들이다.
- 객체를 생성한 후에 속성을 추가하면 다른 객체에는 영향을 주지 않는다.

section111. __init__()

step 1.
알고가기

- 핵심내용
  - 생성자를 이용하면 객체가 생성될 때부터 값을 가지도록 할 수 있다.

- 코딩하기 전에
  - ___init___() 메서드는 클래스에서 특별한 메서드로서 객체를 생성할 때마다 맨 처음 실행된다.
  - 그래서 생성자(Constructor)라는 이름을 가진다.
  - 객체를 만들 때 맨 처음 실행되기 때문에 변수에 처음 값을 설정하는 용도로 많이 사용한다.

section111. __init__()

step 2.
코딩

```
1:  # section_111.py
2:
3:  class Robot:
4:      '''다양한 로봇을 만드는 클래스'''
5:      def _ _init_ _(self):
6:          self.name = 'pybot'
7:          self.weight = '45kg'
8:
9:  hubo1 = Robot()
10: hubo2 = Robot()
11: print(hubo1.name, hubo1.weight)
12: print(hubo2.name, hubo2.weight)
13:
14: print()
15: hubo1.name = 'minibot'
16: hubo2.weight = '60kg'
17: print(hubo1.name, hubo1.weight)
18: print(hubo2.name, hubo2.weight)
```

```
pybot 45kg
pybot 45kg

minibot 45kg
pybot 60kg
```

section111. __init__()

step 3.
코드분석

- def __init__(self):
  - 메서드는 항상 def 키워드로 정의한다.
  - 관례상 필수적으로 self를 매개변수로 갖는다.
  - self는 객체 자신을 의미한다.
- self.name = 'pybot'
  - 객체 자신을 의미하는 self와 속성명을 점 표기법으로 연결하여 변수를 초기화 한다.
- hubo1 = Robot()
  - __init__() 메서드는 이처럼 객체를 생성할 때마다 매번 실행된다.
  - 따라서 모든 객체는 __init__() 메서드에서 초기화된 속성을 공통으로 갖게 된다.
- 객체 생성 이후 robot1, robot2는 각각 독립적인 객체이다.

section111. __init__()

step 4.

3줄 요약

• 3줄 요약 •

- __init__( )은 객체 생성 시 변수를 초기화하는 목적으로 사용된다.
- __init__( )은 객체 생성 시 가장 먼저 자동으로 실행되는 특별한 메서드이다.
- __init__( )은 기본 매개변수를 가지며 관례적으로 self를 사용한다.

section112. __init__()에 매개변수 추가하기

step 1.
알고가기

- 핵심내용
  - 생성자에 매개변수를 추가하여 객체마다 특별한 값을 설정할 수 있다.

- 코딩하기 전에
  - 클래스 정의시 ___init___() 메서드에 매개변수를 추가하면 객체를 생성할 때 특별한 값을 설정할 수 있다.

section112. __init__()에 매개변수 추가하기

step 2.
코딩

```
1:  # section_112.py
2:
3:  class Robot:
4:    "다양한 로봇을 만드는 클래스"
5:      def __init__(self, name, build_year):
6:          self.maker = 'KAIST'
7:          self.isWalking = True
8:          self.name = name
9:          self.build_year = build_year
10:
11: hubo1 = Robot('shane', 2016)
12: print(hubo1.maker, hubo1.isWalking, hubo1.name,
          hubo1.build_year)
13:
14: hubo2 = Robot('albert', 2018)
15: print(hubo2.maker, hubo2.isWalking, hubo2.name,
          hubo2.build_year)
```

self를 제외하면 두 개의 매개 변수를 가진 메서드야.

```
KAIST True shane 2016
KAIST True albert 2018
```

section112. __init__()에 매개변수 추가하기

step 3.
코드분석

- def __init__(self, name, build_year):
  - 객체를 생성할 때마다 name과 build_year를 설정하도록 만든다.
- self.name = name
  - 객체의 속성인 name을 객체 생성시 전달받은 name값으로 초기화한다.
  - 우변의 name 값은 객체 생성할 때마다 달라질 수 있다.
- hubo1 = Robot('shane', 2016)
  - hubo1 객체를 생성할 때 'shane'과 2016을 전달했다.
  - 따라서 객체의 name과 build_year속성이 이 값으로 설정된다.

section112. __init__()에 매개변수 추가하기

step 4.

3줄 요약

• 3줄 요약 •

- __init__( )에 self 이외에 필요한 매개 변수를 추가할 수 있다.
- 객체를 생성할 때 self를 제외한 매개 변수의 개수만큼 값을 입력해 줘야 한다.
- 객체 생성 시 입력한 값들은 객체 고유의 값이 된다.

section113. 메서드

step 1.
알고가기

- 핵심내용
  - 메서드는 클래스의 기능을 구현하는 부분이다.

- 코딩하기 전에
  - 이제는 객체의 상태를 바꾸거나 데이터 처리를 하기 위해 메서드를 정의한다.
  - __init__() 이외에 우리가 클래스에 필요하다고 생각하는 메서드를 추가할 수 있다.

section113. 메서드

step 2.
코딩

```
1:  # section_113.py
2:
3:  class Robot:
4:      '''다양한 로봇을 만드는 클래스'''
5:      def _ _init_ _(self, name, build_year):
6:          self.maker = 'KAIST'
7:          self.isWalking = True
8:          self.name = name
9:          self.build_year = build_year
10:         self.xpos = 0
11:         self.ypos = 0
12:
13:     def move(self):
14:         if self.isWalking:
15:             self.xpos += 1
16:             self.ypos += 1
17:
18:     def curPosition(self):
19:         print('현재 좌표: ({}, {})'.format(self.xpos, self.ypos))
20:
21: hubo = Robot('shane', 2016)
22: hubo.move()
23: hubo.move()
24: hubo.move()
25: hubo.curPosition()
```

section113. 메서드

step 3.
코드분석

- def move(self):
  - 메서드를 정의하는 방법은 함수를 정의하는 방법과 같다.
  - self는 관례적으로 항상 입력하도록 되어 있다.
- hubo.move()
  - 객체가 가진 메서드를 호출할 때는 객체변수와 메서드명을 점 표기법으로 사용한다.
  - 객체변수.메서드명()

section113. 메서드

step 4.

3줄 요약

• 3줄 요약 •

- 클래스의 메서드는 객체의 기능을 구현하는 부분이다.
- 메서드를 이용해서 객체의 속성 값을 변경하거나 데이터를 처리한다.
- 메서드를 사용할 때는 점(.)을 이용해 '객체.메서드명( )'과 같이 사용한다.

section114. __str__() 메서드

step 1.
알고가기

- 핵심내용
  - ___str___()은 객체를 대표하는 문자열을 출력하기 위한 메서드이다.

section114. __str__() 메서드

step 2.
코딩

```
1:  # section_114.py
2:
3:  class Robot:
4:      '''다양한 로봇을 만드는 클래스'''
5:      def __init__(self, name, build_year):
6:          self.maker = 'KAIST'
7:          self.isWalking = True
8:          self.name = name
9:          self.build_year = build_year
10:         self.xpos = 0
11:         self.ypos = 0
12:
13:     def move(self):
14:         if self.isWalking:
15:             self.xpos += 1
16:             self.ypos += 1
17:
18:     def curPosition(self):
19:         print('현재 좌표: ({}, {})'.format(self.xpos, self.ypos))
20:
21:     def __str__(self):
22:         sentence = '이름:{}, 제조년:{}, 현재위치: ({}, {})'\
23:         .format(self.name, self.build_year, self.xpos, self.ypos)
24:         return sentence
25:
26: hubo = Robot('shane', 2016)
27: print(hubo)
```

section114. __str__() 메서드

step 3.
코드분석

- __str__() 메서드도 __init__() 메서드와 같이 파이썬이 지정해 놓은 특별한 메서드이다.
- return sentence
  - __str__() 메서드는 객체를 대표하는 문자열을 반환하도록 설계되어 있어서 꼭 return문으로 마무리 해
- print(hubo)
  - 객체의 이름만 넣으면 파이썬이 알아서 __str__()메서드를 실행시키도록 되어 있다.
  - 이렇게 간단한 방법으로 클래스의 변수들을 출력할 수 있기 때문에 __str__() 메서드를 종종 디버깅용으로 사용한다.

section114. __str__() 메서드

step 4.

3줄 요약

• 3줄 요약 •

- __str__( )은 객체를 대표하는 문자열을 출력하기 위한 메서드이다.
- __str__( )은 return문을 사용해야 한다.
- __str__( )을 디버깅에 활용하면 좋다.

section115. 클래스 변수와 인스턴스 변수

step 1.
알고가기

- 핵심내용
  - 클래스 변수는 모든 객체가 공유하는 변수이다.

- 코딩하기 전에
  - Robot 클래스를 이용해서 여러 개의 객체를 만들 수 있다.
  - 그리고 각 객체들은 독립된 개체로서 값이 바뀌어도 다른 객체에 영향을 주지 않는다.
  - 반면에 클래스 변수는 모든 객체들이 공유하는 변수이다.

section115. 클래스 변수와 인스턴스 변수

step 2.
코딩

```
1:  # section_115.py
2:
3:  class Robot:
4:      '''다양한 로봇을 만드는 클래스'''
5:      population = 0
6:      maker = 'KAIST'
7:
8:      def __init__(self, name, build_year):
9:          self.isWalking = True
10:         self.name = name
11:         self.build_year = build_year
12:         self.xpos = 0
13:         self.ypos = 0
14:         Robot.population += 1
15:
16: hubo1 = Robot('shane', 2016)
17: hubo2 = Robot('albert', 2018)
18: hubo3 = Robot('sol', 2018)
19: print(Robot.maker, Robot.population, hubo1.population)
20:
21: Robot.maker = 'POSTECH'
22: print(hubo2.maker, hubo2.population)
```

클래스 변수야. 모든 객체들이 공유하는 변수지. 위치를 봐. 클래스의 메서드 밖에서 만들어졌어.

self.가 붙은 변수들은 인스턴스 변수야. 모든 객체들이 독립적으로 소유하게 돼.

클래스 변수는 '클래스명.클래스변수명' 처럼 사용해.

section115. 클래스 변수와 인스턴스 변수

step 3.
코드분석

- population = 0
  - 클래스 변수는 클래스 안에서 메서드 밖에 만들어진 변수를 말한다.
  - 클래스 변수 값의 변화는 모든 객체들에 영향을 준다.
  - 반면에 메서드 안에서 self와 결합하여 만들어진 변수들(isWalking, name, build_year 등)은 인스턴스 변수(Instance Variable)라고 한다.
  - 인스턴스 변수 값의 변화는 다른 객체에 영향을 주지 않는다.
- Robot.maker = 'POSTECH'
  - 클래스 변수를 사용할 때는 클래스 이름을 사용하는 것이 좋다.
  - 클래스 변수 maker는 모든 객체가 공유하기 때문에 모든 객체의 maker 값이 변경된다.

section115. 클래스 변수와 인스턴스 변수

step 4.

3줄 요약

• 3줄 요약 •

- 클래스 변수는 클래스의 메서드 밖에서 만들어진 변수이다.
- 클래스 변수는 클래스의 모든 인스턴스들이 공유하는 변수이다.
- 클래스 변수를 사용할 때는 '클래스명.클래스 변수'처럼 사용한다.

section116. 데이터 숨기기

step 1.
알고가기

- 핵심내용
  - 파이썬은 네임 맹글링 방법으로 변수의 사용을 제한할 수 있다.

- 코딩하기 전에
  - 클래스를 정의할 때 객체 사용 시 쉽게 접근하지 못하게 데이터를 숨기고 싶은 속성들이 있을 수 있다.
  - 파이썬은 네임 맹글링 기법을 통해 데이터 숨기기를 구현한다.
  - 이런 속성을 프라이빗 멤버라고도 한다.
  - 그러나 파이썬 철학에 따라 프라이빗 멤버도 변경된 이름으로 접근할 수는 있다.

section116. 데이터 숨기기

step 2.
코딩

```
1:  # section_116.py
2:
3:  class Robot:
4:      '''다양한 로봇을 만드는 클래스'''
5:      def _ _init_ _(self, name, build_year):
6:          self.name = name
7:          self._ _build_year = build_year
8:          self.xpos = 0
9:          self.ypos = 0
10:
11:     def getYear(self):
12:         return self._ _build_year
13:
14: hubo = Robot('shane', 2016)
15:
16: hubo.name = 'albert'
17: print(hubo.name)
18: #print(hubo._ _build_year)
19: print(hubo.getYear())
20:
21: print()
22: print(dir(hubo))
```

section116. 데이터 숨기기

step 3.
코드분석

- 프라이빗 멤버
  - 두 개의 밑줄(_)을 변수명/함수명 앞에 붙인다.
    - 예) __name
  - 변수명/함수명 뒤에는 밑줄(_)을 하나까지만 붙일 수 있다.
    - 예) __name_
- self.__build_year = build_year
  - 숨기고 싶은 변수(프라이빗 멤버)가 있으면 이와 같이 코딩한다.
  - 프라이빗 멤버는 변수명이 변경되어 저장되기 때문에 쉽게 접근하지 못한다.
  - 이것이 네임 맹글링을 이용한 데이터 숨기기이다.

section116. 데이터 숨기기

step 4.

3줄 요약

• 3줄 요약 •

- 변수명/함수명 앞에 '__'가 붙은 변수는 사용하지 말자.
- 변수명/함수명 앞에 '_'를 붙은 변수도 사용하지 말자.
- '__'가 앞에 붙은 멤버명은 네임 맹글링이 적용된다.

section117. 상속

step 1.
알고가기

- 핵심내용
  - 상속은 자식 클래스가 부모 클래스의 멤버들을 물려받는 것을 말한다.

- 코딩하기 전에
  - 클래스와 클래스 사이에 관계를 맺어주는 방식 중 부모-자식 관계를 맺어주는 형태를 상속이라고 한다.
  - 상속 관계를 맺어주면 자식은 부모가 가진 속성과 메서드들을 그대로 물려받는다.
  - 상속의 장점은 기존에 있던 클래스를 다시 사용하는 방법이기 때문에 코드 중복을 줄이고 단순하게 만들 수 있다.
  - 수정할 때도 한 번 만 수정하면 모든 자식 클래스에 적용이 되니까 유지보수하기도 편리하다.

section117. 상속

step 2.
코딩

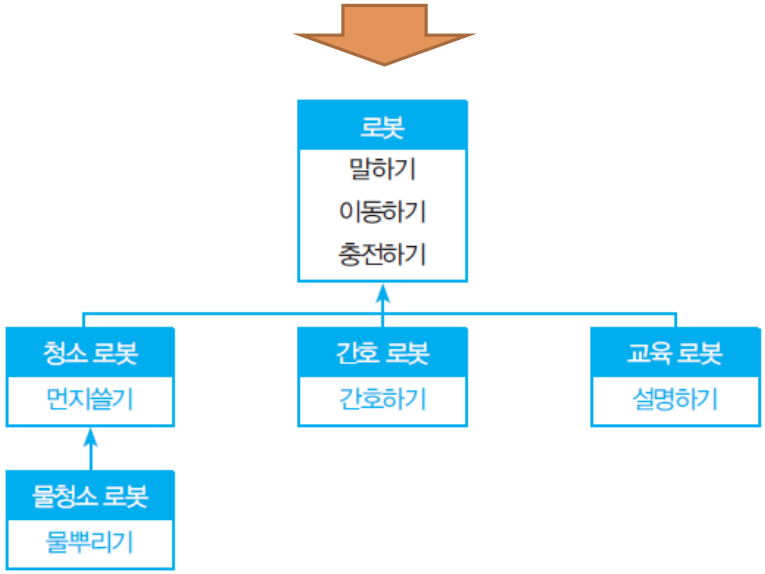
```
1:  # section_117.py
2:
3:  class Robot:
4:      '''다양한 로봇을 만드는 클래스'''
5:      def speak(self):
6:          print('로봇이 말한다.')
7:
8:      def move(self):
9:          print('로봇이 이동한다.')
10:
11:     def charge(self):
12:         print('로봇이 충전한다.')
13:
14: class CleanRobot(Robot):
15:     def broom(self):
16:         print('청소로봇이 먼지를 쓸어 담는다.')
17:
18: robot = Robot()
19: robot.move()
20:
21: clean = CleanRobot()
22: clean.broom()
23: clean.move()
```

section117. 상속

step 3.
코드분석

- 상속의 장점
  - 코드 중복을 줄이고, 유지보수에 용이하다.

[구현하려는 로봇의 종류]



[로봇들 사이의 상속 관계]

section117. 상속

step 3.
코드분석

- class Robot:
  - Robot 클래스 정의
  - 3개의 메서드로 구성된다.
- class CleanRobot(Robot):
  - 자식클래스 CleanRobot 클래스 정의
  - Robot 클래스를 상속받기 위해 괄호를 사용한다.
  - 상속이 이뤄지면 자식클래스는 부모클래스의 모든 멤버들을 물려받는다. 단, 프라이빗 멤버는 상속되지 않는다.
  - CleanRobot 클래스는 Robot 클래스의 3개 메서드와 broom() 메서드를 갖게 된다.
- clean.move()
  - 자식클래스의 객체를 이용해 상속받은 메서드를 실행시킬 수 있다.

section117. 상속

step 4.

3줄 요약

• 3줄 요약 •

- 상속이란 클래스와 클래스 사이에 부모–자식 관계를 만들어 주는 것이다.
- 상속 관계가 생기면 자식 클래스는 부모의 속성과 메서드를 그대로 물려받는다.
- 상속 관계를 사용하면 코드를 재사용할 수 있고 유지 보수가 편리하다.

section118. 상속 - super()

step 1.
알고가기

- 핵심내용
  - 상속은 자식 클래스가 부모 클래스의 멤버들을 물려받는 것을 말한다.

- 코딩하기 전에
  - super()는 부모클래스를 지칭하는 내장함수이다.

section118. 상속 - super()

step 2.
코딩

```
1:  # section_118.py
2:
3:  class Robot:
4:      '''다양한 로봇을 만드는 클래스'''
5:      def __init__(self, name, pos):
6:          self.name = name
7:          self.pos = pos
8:
9:      def move(self):
10:         print('로봇이 이동한다.')
11:
12:     def charge(self):
13:         print('로봇이 충전한다.')
14:
15: class CleanRobot(Robot):
16:     def __init__(self, name, pos, filtertype):
17:         super().__init__(name, pos)
18:         self.filtertype = filtertype
19:
20:     def broom(self):
21:         print('청소 로봇이 먼지를 쓸어 담는다.')
22:
23:     def __str__(self):
24:         word = '{}를 장착한 {}, {}에서 시작'\
25:                .format(self.filtertype, self.name, self.pos)
26:         return word
27:
28: clean = CleanRobot('깔끔이로봇', (20, 45), '먼지필터')
29: print(clean)
30: clean.move()
31: clean.broom()
```

section118. 상속 – super()

step 3.
코드분석

- def __init__(self, name, pos, filtertype):
  - 자식클래스의 생성자이다.
  - 3개의 인수를 받아 초기화를 한다.
  - 이 중 name과 pos는 부모로부터 상속받은 속성이다.
- super().__init__(name, pos)
  - 부모로부터 상속받은 속성인 name, pos를 초기화할 때 부모클래스의 생성자를 이용하는 방법이다.
  - 자식클래스의 __init__() 메서드에서 중복된 코드를 줄이기 위해 부모클래스의 __init__() 메서드를 호출할 수 있다.

section118. 상속 - super()

step 4.

3줄 요약

• 3줄 요약 •

- super( )는 부모 클래스를 의미하는 내장 함수이다.
- super( )는 자식 클래스에서 사용할 수 있다.
- super( )를 사용하면 코드를 줄일 수 있다.

section119. 상속 - 오버라이딩

step 1.
알고가기

- 핵심내용
  - 오버라이딩은 부모 클래스로부터 물려받은 메서드를 재정의하는 것이다.

- 코딩하기 전에
  - 부모클래스가 가진 메서드 중에서 같은 이름을 사용하고 싶지만, 다른 기능을 가진 메서드로 사용하고 싶을 때 오버라이딩을 한다.

section119. 상속 - 오버라이딩

step 2.
코딩

```
1:  # section_119.py
2:
3:  class Robot:
4:      '''다양한 로봇을 만드는 클래스'''
5:      def move(self):
6:          print('로봇이 이동한다.')
7:
8:      def charge(self):
9:          print('로봇이 충전한다.')
10:
11: class CleanRobot(Robot):
12:     def move(self):
13:         print('청소 로봇이 이동한다.')
14:
15:     def charge(self):
16:         print('청소 로봇이 충전한다.')
17:
18: robot = Robot()
19: robot.move()
20: robot.charge()
21:
22: clean = CleanRobot()
23: clean.move()
24: clean.charge()
```

section119. 상속 - 오버라이딩

step 3.
코드분석

- def move(self):
  print('로봇이 이동한다.')
  - 부모클래스에 정의된 move() 메서드이다.
- class CleanRobot(Robot):
  - CleanRobot은 Robot 클래스를 상속받았다.
- def move(self):
  print('청소로봇이 이동한다.')
  - 자식클래스에도 부모클래스와 동일한 이름을 가진 메서드를 만들 수 있다.
  - 메서드의 코드는 자식클래스에서 마음대로 바꿀 수 있다.
  - 이렇게 부모의 메서드를 재정의해서 사요하는 방법을 오버라이딩이라고 한다.

section119. 상속 - 오버라이딩

step 4.

3줄 요약

• 3줄 요약 •

- 오버라이딩은 부모 클래스로부터 물려받은 메서드를 자식 클래스에서 재정의하는 것이다.
- 자식 클래스 입장에서는 부모의 메서드를 그냥 써도 되고, 오버라이딩해서 써도 된다.
- 다형성은 같은 이름의 메서드가 다른 형태로 동작하는 것을 말하며, 오버라이딩과 관련이 있다.

section120. 포함 관계

step 1.
알고가기

- 핵심내용
  - 한 클래스가 다른 클래스의 멤버로 포함되는 관계를 포함관계라고 한다.

- 코딩하기 전에
  - 클래스와 클래스와의 관계를 설정하는 방법 중 상속관계 외에 한 가지 방법이 더 있는데, 클래스와 클래스 사이에 포함관계를 맺어주는 것이다.
  - 포함관계는 두 클래스가 논리적으로 서로 소유의 관계일 때 설정해주면 좋다.
  - 예) 로봇클래스와 모터클래스, 자동차클래스와 엔진클래스

section120. 포함 관계

step 2.
코딩

```
1:  # section_120.py
2:
3:  import random
4:
5:  class Motor:
6:      def __init__(self):
7:          self.distance = 0
8:
9:      def forward(self):
10:         print('앞으로 이동한다.')
11:         self.distance += 1
12:
13:     def backward(self):
14:         print('뒤로 이동한다.')
15:         self.distance -= 1
16:
17: class Robot:
18:     '''다양한 로봇을 만드는 클래스'''
19:     def __init__(self):
20:         self.drive = Motor()
21:
22:     def __str__(self):
23:         return '이동거리: {}'.format(self.drive.distance)
24:
25: robot = Robot()
26:
27: for i in range(10):
28:     if random.randint(0,1):
29:         robot.drive.forward()
30:     else:
31:         robot.drive.backward()
32:
33: print(robot)
```

section120. 포함 관계

step 3.
코드분석

- class Motor:
  - Motor 클래스 정의
- class Robot:
  - Robot 클래스 정의
- self.drive = Motor()
  - 포함관계를 설정하는 방법이다.
  - Robot 클래스의 속성 중 drive를 Motor 클래스의 객체변수로 사용한다.
  - drive는 Robot 클래스의 멤버이자, Motor 클래스의 인스턴스이다.
  - 이제 drive를 이용해 Motor 클래스를 이용할 수 있다.
- robot.drive.forward()
  - robot 객체로 Motor클래스의 forward()를 사용하는 방법이다.

section120. 포함 관계

step 4.

3줄 요약

• 3줄 요약 •

- 클래스와 클래스의 관계를 설정하는 방법에는 상속 관계와 포함 관계가 있다.
- 포함 관계는 한 클래스가 다른 클래스의 멤버로 포함되는 것을 말한다.
- 포함 관계로 설정했을 때 변수나 메서드를 사용하는 방법을 알아두자.