# 14. 클라스(Class)

## 14-1. 클라스와 메소드 (Class 와 Method)

## Python 의 class 와 method 는 C++, Java 등의 객체지향 프로그래밍의 개념과 동일함.

## 객체지향개발(OOP - Object Oriented Programming) 4가지 특성

- 객체지향개발(Object Oriented Programming)의 특성은 크게 추상화, 캡슐화, 상속성, 다형성이 있다.

#### 1. 추상화(Abstraciton)

- 공통의 속성이나 기능을 묶어 이름을 붙이는 것
- OOP 에서 클래스를 정의하는 것을 추상화라고 할 수 있다.

#### 2. 은닉화 (Encapsulation)

- 변수와 함수를 하나로 묶어서 외부에서의 접근을 막고, 함수를 통해서만 접근 가능하도록 하는 것

#### 3. 상속 (Inheritance)

- 상위 개념의 특징과 메소드를 하위 개념이 물려 받아 재사용

#### 4. 다형성 (Polymorphism)

- 부모 클래스로부터 물려받은 가상 함수를 자식 클래스 내에서 오버라이딩 하여 사용하는 것

## Python 에서는 모든 것을 object 라고 부름.

data (string, integer, float), function, built-in function

## Python 에서 모든 object 는 특정 class 에 속함.

## class 란 객체 (object) 를 정의하는 틀 혹은 설계도와 같은 의미임.

## method 는 class 내에서 정의한 함수임. class 는 속성 (data) 와 메소드(method) 로 구성됨.

```
class ClassName:
    <data>

    def methodName(self):
        <method body>
        return
```

## dir( ) 내장함수는 객체가 어떤 변수와 메소드를 가지고 있는지 반환함.

Python 언어 자체에서 사용하는 특별한 method 들은 __name__ 형식의 이름을 가지고 있음

In [19]:

```
print(type("string"))
```

<class 'str'>

In [20]:

```
print(type(15))
```

<class 'int'>

In [21]:

```
print(type(12.5))
```

<class 'float'>

In [22]:

```
def f(n):
    return n
```

In [23]:

```
print(type(f))
```

<class 'function'>

In [24]:

```
print(type(print))
```

<class 'builtin_function_or_method'>

In [25]:

```
class Car:
    color = "blue"
    max_speed = 100

    def speedCheck(self, n):
        if n > self.max_speed:
            print("Too fast")
        else:
            print("Good speed")
```

In [26]:

```
sonata = Car()
```

In [27]:

```
1 sonata.speedCheck(100)
```

Good speed

In [28]:

```
1 sonata.max_speed
```

Out[28]:

100

In [29]:

```
1 print(type(sonata))
```

<class '__main__.Car'>

In [30]:

```
1 print(type(sonata.speedCheck))
```

<class 'method'>

## 14-2 Python 의 Object Oriented Programming

- Class : 속성(attribute) 과 method 로 구성

  class Classname:

  ```
      def __init__(self, *args):
          self.변수명 (속성 정의)

      def methods(self, *args):
          메소드 정의
  ```

- Constructor (생성자) : **init**()

  class 의 instance 생성 시 자동으로 호출되어 object 를 초기화 하는 method

- Inheritance (상속)

  class Subclass(superclass):

- Encapsulation (은닉화)

  self._변수명

- Polymorphism (다형성)

  Python 은 자체로 다형성의 특성을 가진 언어이다.

  1 + 2 ==> 3
  '1' + '2' =='12'

- Python interpreter 는 + operator 를 만나면 __add__ method 호출

In [42]:

```
a = 1
a.__add__(2)    # numeric 연산
```

Out[42]:

3

In [43]:

```
a = '1'
a.__add__('2')    # string 연산
```

Out[43]:

'12'

## Class example

In [44]:

```
# class 정의
class Person:
    def __init__(self, name):
        self.name = name

    def sayHello(self):
        print(self.name, "님 안녕하세요")

# instance 생성
A = Person("파이썬")

# instance 를 통한 method 호출
A.sayHello()

# instance 를 통한 attribute 접근
print(A.name)
```

파이썬 님 안녕하세요
파이썬

## class 변수와 instance 변수

In [45]:

```
class Person:
    country = 'Korea'
    def __init__(self, country):
        self.country = country

A = Person('USA')

print(Person.country)     # class 변수
print(A.country)          # instance 변수
```

```
Korea
USA
```

### Inheritance (상속)

In [46]:

```
class Korean:        # parent class
    def speak(self):
        print('나는 한국말을 합니다.')

    def sex(self):
        print('부모 성별')

class Man(Korean):      # child class
    def sex(self):
        print('나는 한국 남자 입니다.')

class Woman(Korean):  # child class
    def sex(self):
        print('나는 한국 여자 입니다.')

A = Man()
A.speak()
A.sex()
print()
B = Woman()
B.speak()
B.sex()
```

```
나는 한국말을 합니다.
나는 한국 남자 입니다.

나는 한국말을 합니다.
나는 한국 여자 입니다.
```

## 14-3. built-in class methods

In [47]:

```
1 dir(A)
```

Out[47]:

```
['__class__',
 '__delattr__',
 '__dict__',
 '__dir__',
 '__doc__',
 '__eq__',
 '__format__',
 '__ge__',
 '__getattribute__',
 '__gt__',
 '__hash__',
 '__init__',
 '__init_subclass__',
 '__le__',
 '__lt__',
 '__module__',
 '__ne__',
 '__new__',
 '__reduce__',
 '__reduce_ex__',
 '__repr__',
 '__setattr__',
 '__sizeof__',
 '__str__',
 '__subclasshook__',
 '__weakref__',
 'sex',
 'speak']
```

In [48]:

```
1 print(A)
```

```
<__main__.Man object at 0x000001CB80E12F48>
```

## str, repr

- **str** - print() 문 호출시 string 반환
- **repr** - object 자체 출력시 readable 한 형태 반환
- **repr** 만 정의되어 있고 **str** 이 생략된 경우 **str=repr** 와 같음

In [92]:

```
class Korean:       # parent class

    def speak(self):
        print('나는 한국말을 합니다.')

    def __str__(self):
        return "Print 문 사용시 호출"

    def __repr__(self):
        return "object 반환시 호출 "

A = Korean()
```

In [93]:

```
A.speak()
```

나는 한국말을 합니다.

In [94]:

```
A
```

Out[94]:

object 반환시 호출

In [95]:

```
print(A)
```

Print 문 사용시 호출

## 다형성 (polymorphism) 구현

**super()**

- 자식클래스에서 부모클래스의 내용을 사용하고 싶은 경우
- super().부모클래스내용

In [96]:

```python
class Person:

    class_var = '클라스변수'

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

    def show_salary(self):
        print("Salary is unknown")

class Employee(Person):
    def __init__(self, name, age, salary):    # Person 의 __init__() override
        super().__init__(name, age)           # Person 의 __init__() 내용을 상속
        self.salary = salary

    def show_salary(self):                    # Person 의 show_salary() override
        print( "{}'s salary is {} and age is {}".format(self.name, self.salary, self.age))

A = Person('Tom', 30)
A.show_salary()

B = Employee('Tom', 30, 10000)
B.show_salary()
```

```
Salary is unknown
Tom's salary is 10000 and age is 30
```

# Class 내의 method 종류

- instance method
  - 인스턴스를 첫번째 인자로 받음
  - 관습적으로 self 로 표시
- class method
  - 클래스를 첫번째 인자로 받음
  - 관습적으로 cls 로 표시
  - 모든 인스턴스가 공유하는 클래스 변수와 같은 데이터를 생성, 변경 또는 참조하기 위한 메소드
- static method
  - 첫번째 인자 자동 생성 되지 않음 - class 내의 일반 함수
  - class 내의 utility 성격의 method 로 사용
  - class, instance 에서 모두 접근 가능

In [99]:

```python
class Employee(object):

    raise_amount = 1.0  # 연봉 인상율 클래스 변수

    def __init__(self, first, last, pay):
        self.first = first
        self.last = last
        self.pay = pay

    def apply_raise(self):
        self.pay = int(self.pay * self.raise_amount)

    def full_name(self):
        return '{} {}'.format(self.first, self.last)

    def get_pay(self):
        return '현재 "{}"의 연봉은 "{}"입니다.'.format(self.full_name(), self.pay)

    # class 메소드 데코레이터를 사용하여 class 메소드 정의
    @classmethod
    def change_raise_amount(cls, amount):
        cls.raise_amount = amount
        print('인상율 "{}"가 적용 되었습니다.'.format(amount))

    # static 메소드 데코레이터를 사용하여 static 메소드 정의
    @staticmethod
    def calc_tax(amount, tax_rate):
        print(amount * tax_rate)

emp_1 = Employee('Sanghee', 'Lee', 50000)
emp_2 = Employee('Minjung', 'Kim', 60000)

# 연봉 인상 전
print(emp_1.get_pay())
print(emp_2.get_pay())

# 연봉 인상율 변경
Employee.change_raise_amount(1.2)    # Employee.raise_amount = 1.2 와 동일한 결과

# 연봉 인상
emp_1.apply_raise()
emp_2.apply_raise()

# 연봉 인상 후
print(emp_1.get_pay())
print(emp_2.get_pay())

Employee.calc_tax(1000, 0.05)   # static method 도 instance 를 통해 접근
emp_1.calc_tax(1000, 0.04)
```

```
현재 "Sanghee Lee"의 연봉은 "50000"입니다.
현재 "Minjung Kim"의 연봉은 "60000"입니다.
인상율 "1.2"가 적용 되었습니다.
현재 "Sanghee Lee"의 연봉은 "60000"입니다.
현재 "Minjung Kim"의 연봉은 "72000"입니다.
50.0
40.0
```

# 연습문제

- Car class 를 parent class 로 만들고 Sonata 와 Volvo 를 child class 로 구성
- CODE HERE 부분을 채워서 출력된 결과와 동일하도록 code 완성

In [15]:

```
class Car:
    def __init__(self, color='red', power=2000):
        self.color = color
        self.power = power
        self.speed = 0

    def forward(self, speed):
        self.speed += speed
        return '앞으로 전진 : 시속 {} km'.format(self.speed)

    def backward(self):
        pass

class Sonata(Car):
    def __init__(self, color, power, size):

        # CODE HERE

    def backward(self, speed):

        # CODE HERE

class Volvo(Car):
    def __init__(self, color, power, size, price):

        # CODE HERE

    def forward(self, speed):

        # CODE HERE

    def backward(self):

        # CODE HERE

sonata = Sonata('black', 1800, 5)
volvo = Volvo('white', 2500, 7, 5000000)

print(sonata.color)
print(sonata.forward(100))
print(sonata.backward(30))

print(volvo.color)
print(volvo.price)
print(volvo.forward(100))
print(volvo.backward())
```

```
black
앞으로 전진 : 시속 100 km
뒤로 후진 70 km/hour
white
5000000
앞으로 전진 : 시속 200 km
후진 불가
None
```