머신러닝/딥러닝을 위한

# 파이썬(Python)

– class • exception • with –

# 클래스 – class

- 파이썬 클래스는 class 키워드를 사용하여 자신만의 데이터타입을 만들 수 있음

```
class 클래스이름:
    def    __init__(self, 인수, ...):    # 생성자
    def    메서드이름(self, 인수, ...)  # 메서드
```

```
class Person:

    def __init__(self, name):
        self.name = name
        print(self.name + " is initialized")

    def work(self, company):
        print(self.name + " is working in " + company)

    def sleep(self):
        print(self.name + " is sleeping")

# Person instance 2개 생성
obj = Person("PARK")

# method call
obj.work("ABCDEF")
obj.sleep()

# 속성에 직접 접근, 기본적으로 파이썬에서는 모두 public
print("current person object is ", obj.name)
```

```
PARK is initialized
PARK is working in ABCDEF
PARK is sleeping
current person object is  PARK
```

- 파이썬 클래스에서는 __init__ 메서드가 생성자(constructor) 역할을 수행하여, 인스턴스가 만들어 질 때 한번만 호출됨
- 파이썬에서는 클래스 메서드의 첫번째 인수로 '자신의 인스턴스'를 나타내는 self 를 반드시 기술해야 함
- 기본적으로 파이썬에서는 메서드와 속성 모두 public

# 클래스 - class

- 파이썬은 기본적으로 모든 멤버가 public 이기 때문에, 외부에서 직접 접근 가능함

- 멤버변수(variable), 멤버메서드(method)를 __멤버변수, __멤버메서드 형태로 선언한다면 private으로 설정 할 수 있음

```
class PrivateMemberTest:

    def __init__(self, name1, name2):

        self.name1 = name1
        self.__name2 = name2    # private member variable
        print("initialized with " + name1 + " ," + name2)

    def getNames(self):
        self.__printNames()
        return  self.name1, self.__name2

    def __printNames(self):    # private member method
        print(self.name1, self.__name2)

# 인스턴스 생성
obj = PrivateMemberTest("PARK", "KIM")

print(obj.name1)
print(obj.getNames())
print(obj.__printNames())  # error 발생
print(obj.__name2)         # error 발생
```

```
initialized with PARK ,KIM
PARK
PARK KIM
('PARK', 'KIM')

---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
<ipython-input-4-bb55048fba43> in <module>()
     19 print(obj.name1)
     20 print(obj.getNames())
---> 21 print(obj.__printNames())  # error 발생
     22 print(obj.__name2)         # error 발생

AttributeError: 'PrivateMemberTest' object has no attribute '__printNames'
```

# 클래스 – class

- 외부함수와 클래스 method name 이 같은 경우

  => C++, Java 언어의 this 처럼 self 를 통해 method 호출

  => self 를 붙이지 않으면 동일한 이름의 외부 함수 호출됨

```
def print_name(name):

    print("[def] ", name)

class SameTest:

    def __init__(self):
        # 아무것도 않하기 때문에 pass
        pass

    # 외부 함수와 동일한 이름으로 method 정의
    def print_name(self, name):
        print("[SameTest] ", name)

    def call_test(self):

        # 외부 함수 호출
        print_name("KIM")

        # 클래스 내부 method 호출
        self.print_name("KIM")

# create SameTest object
obj = SameTest()

# call function print_name
print_name("LEE")

# call method print_name
obj.print_name("LEE")

# call method call_test
obj.call_test()
```

```
[def]  LEE
[SameTest]  LEE
[def]  KIM
[SameTest]  KIM
```

# 예외처리 – exception

- 파이썬 exception은 try...except 문을 사용함. try 블록에서 에러가 발생 시 except 문으로 이동하여 예외 처리 수행

- 발생된 exception을 그냥 무시하기 위해서는 보통 pass 문을 사용하며, 또한 개발자가 에러를 던지기 위해서는 raise문을 사용함

- try 문은 또한 finally 문을 가질 수도 있는데, finally 블럭은 try 블럭이 정상적으로 실행되든, 에러가 발생하여 except이 실행되든 항상 마지막에 실행됨

```
def calc(list_data):

    sum = 0

    try:
        sum = list_data[0] + list_data[1] + list_data[2]

        if sum < 0:
            raise Exception("Sum is minus")

    except IndexError as err:
        print(str(err))
    except Exception as err:
        print(str(err))
    finally:
        print(sum)
```

```
calc([1, 2])  # index error 발생
```

```
list index out of range
0
```

```
calc([1, 2, -100])  # 인위적인 exception 발생
```

```
Sum is minus
-97
```

# with 구문

- 일반적으로 파일 (file) 또는 세션 (session) 을 사용하는 순서는 다음과 같음

  open() => read() 또는 write()  => close()

- 그러나 파이썬에서 with 구문을 사용하면 명시적으로 리소스 close() 를 해주지 않아도 자동으로 close() 해주는 기능이 있음. with 블록을 벗어나는 순간 파일 파일이나 세션 등의 리소스를 자동으로 close 시킴

  => 딥러닝 프레임워크인 TensorFlow 의 session 사용시 자주 사용됨

```
# 일반적인 방법

f = open("./file_test", 'w')

f.write("Hello, Python !!!")

f.close()
```

```
# with 구문을 사용한 방법
# with 블록을 벗어나는 순간 파일 객체 f가 자동으로 close 됨.

with open("./file_test", 'w') as f:
    f.write("Hello, Python !!!")
```

[예제 1] 다음의 조건을 만족하는 Point 클래스를 작성하시오

Point 클래스는 생성자를 통해 (x, y) 좌표를 입력받는다.
setx(x), sety(y) 메서드를 통해 x 좌표와 y 좌표를 따로 입력받을 수도 있다.
get() 메서드를 호출하면 튜플로 구성된 (x, y) 좌표를 반환한다.
move(dx, dy) 메서드는 현재 좌표를 dx, dy만큼 이동시킨다.

[예제 2] 다음의 Error 원인과 해결방안을 기술하시오

```
class MyTest:

    def hello():

        print("MyTest hello method")
```

```
obj = MyTest()

obj.hello()
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-3-d9d34e58be7e> in <module>
      1 obj = MyTest()
      2
----> 3 obj.hello()

TypeError: hello() takes 0 positional arguments but 1 was given
```

[예제 3] 다음의 조건을 만족하는 MyCalc 클래스를 작성한후, MyCalc 객체 obj를 생성한 후에 4개의 메서드를 호출하는 코드를 작성하시오

임의의 2개의 숫자를 객체 생성시에 입력으로 받아 사칙연산(+,-,*,/)을 수행하여 결과를 return해주는 4개의 메서드  summation, subtract, multiply, divide 를 가지고 있다

[예제 4]  클래스 MyFile의 save_file 메서드는 임의의 문자열을 입력 파라미터로 받아, 문자열을 현재 디렉토리에 text_file.txt 파일로 저장하는 기능을 가지고 있다.
MyFile 클래스의 save_file 메서드를 구현하시오