# 머신러닝
# (Machine Learning)

## LECTURE IV: 기초 파이썬 프로그래밍 3
## (Basic Python Programming)

**Dai-Gyoung Kim**

*Department of Applied Mathematics*

*Hanyang University ERICA*

# 기초 파이썬 프로그래밍 (Basic Python Programming)

## Contents


- 파이썬 기초
- 파이썬 프로그래밍
  - 파이썬 프로그래밍의 구조와 제어문
  - 파이썬 프로그래밍의 입력과 출력
  - 파이썬 프로그래밍의 핵심: 클래스, 모듈

# ▶ 파이썬 프로그래밍의 핵심: 클래스, 모듈입력과 출력

## ❖ 클래스

- 클래스(class)는 여러 가지 데이터와 테이터를 다루는 함수를 하나의 객체로 관리하게 하는 도구이며, 클래스를 활용하면 훨씬 더 효율적인 프로그램이 가능함.
- 파이썬은 대부분의 기능을 클래스를 통해 구현하며, 프로그램의 효율성을 위해 클래스를 많이 선언함.
- 클래스는 어떤 객체를 만들기 위한 선언적 존재이며, 객체는 클래스에 정의된 내용대로 메모리에 생성한 실제 데이터를 의미함.
- 클래스는 객체를 생성하기 위해 변수 형태의 속성(attribute)과 함수 형태의 메서드(method)를 정의하는 일종의 틀이라 할 수 있음.
- 프로그램으로 표현 가능한 존재를 클래스로 선언하고, 이 클래스를 이용하여 객체를 만들며, 이 객체를 이용하여 프로그램을 작성하는 것을 객체 지향 프로그래밍(OOP: Object-Oriented Programming)이라 함.

# 객체 지향 기본 개념

- 객체 지향 언어에는 Java, C++, C#, Modula-3, Python 등이 있음.
- **객체 지향 프로그래밍**이란 소프트웨어 객체(object)들을 만들고 그 객체들이 메시지들을 주고받으며 상호 작용하도록 하여 원하는 결과를 만들어내는 프로그램을 작성하는 것.

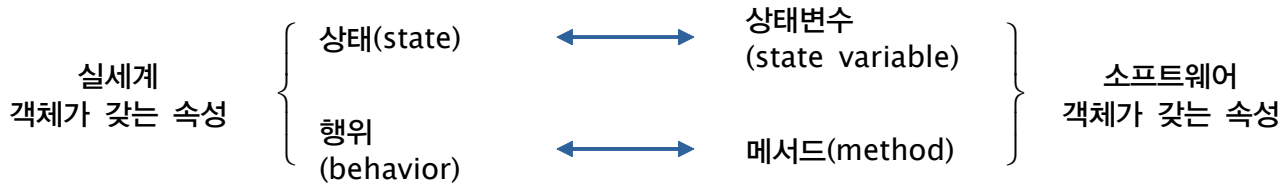
## [객체 지향 언의 기본 구성 요소]

- 클래스(Class)
- 객체(Object)
- 메서드(Method), 메시지(Message)

- 모든 객체 지향 프로그래밍 언어는 클래스를 지원함.
- 클래스는 객체를 만들기 위한 설계도 또는 틀이라고 할 수 있음.

## [상태 변수와 메서드]

- 소프트웨어 객체는 상태 변수들과 메서드를 가짐.




## [메시지 전송]

- 객체 a가 객체 b에게 메시지 m을 보내는 것은 파이썬과 같은 객체 지향 프로그래밍 언어 수준에서는 객체 a가 객체 b의 메서드 m()을 호출하는 것으로 표현할 수 있음.

## [인스턴스(instance)]

- 클래스 X를 설계도로 하여 만들어지는 객체를 X 클래스의 인스턴스라고함.

# 클래스의 기본 구조

```
class 클래스이름:
    """문서 문자열 (생략 가능)"""
    생성자(constructor)
    소멸자(destructor)
    클래스 변수들
    정적 변수들
    클래스 메서드들
    인스턴스 메서드들
```

# 클래스의 필요성

**예제1** • 덧셈 기능만을 가진 간이 계산기 만들기

```
>>> result = 0
>>> def adder(num):
...     global result
...     result += num
...     return result
...
```

```
>>> adder(3)
3
>>> adder(4)
7
```

• 계산기 2대

```
>>> result1 = 0
>>> def adder1(num):
...     global result1
...     result1 += num
...     return result1
...
```

```
>>> result2 = 0
>>> def adder2(num):
...     global result2
...     result2 += num
...     return result2
...
```

```
>>> adder1(3)
3
>>> adder1(4)
7
```

```
>>> adder2(5)
5
>>> adder2(10)
15
```

**예제2** • 클래스를 이용한 덧셈 기능만을 가진 간이 계산기 만들기

```
>>> class Calc:
...     def __init__(self):
...         self.result = 0
...     def adder(self, num):
...         self.result +=num
...         return self.result
...
>>> cal1 = Calc()
>>> cal2 = Calc()
```

```
>>> cal1.adder(2)
2
>>> cal1.adder(7)
7
>>> cal2.adder(3)
3
>>> cal2.adder(7)
10
```

# 클래스의 선언

- 파이썬에서 클래스 선언은 class 예약어를 이용하여 선언함.
- 클래스 내에서는 self라는 변수를 함수의 첫 번째 인수로 받아야 함.
- 클래스에 의해 만들어진 객체를 인스턴스라고 함.

# 클래스 기초 쌓기

- 클래스 변수

```
>>> class Service:
...     secret = "영구는 배꼽이 두 개다."
...
```

✓ 위의 클래스 이름은 Service이다. 이 Service 클래스는 어떤 유용한 정보를 제공하는 한 인터넷 서비스 업체라고 가정하자. 이 업체는 가입한 고객에게만 유용한 정보를 제공한다고 한다.

✓ 위 인터넷 업체에 가입하는 방법은 다음과 같다.

```
>>> fox9 = Service()
```

✓ 위와 같이 하면 fox9라는 아이디로 인터넷 서비스 업체인 Service 클래스를 이용할 수 있다.

```
>>> fox9.secret
'영구는 배꼽이 두 개다.'
```

- 클래스 함수

✓ 이 업체가 고객을 위해 더하기를 해주는 서비스를 추가해 클래스를 업그레이드 하였다.

```
>>> class Service:
...     secret = "영구는 배꼽이 두 개다."
...     def sum(self, a,b):
...         result = a + b
...         print("%s + %s = %s입니다." % (a,b,result))
...
```

```
>>> fox9 = Service()
```

```
>>> fox9.sum(2,3)
2 + 3 = 5입니다.

>>> Service.sum(fox9,2,3)
2 + 3 = 5입니다.
```

✓ 사용자에게 이름을 입력받아서 서비스를 제공할 때 앞부분에 그 이름을 넣어주기.

```
>>> class Service:
...     secret = "영구는 배꼽이 두 개다."
...     def setname(self, name):
...         self.name = name
...     def sum(self, a,b):
...         result = a + b
...         print("%s님! %s + %s = %s입니다." % (self.name,a,b,result))
...
```

```
>>> fox9 = Service()
>>> fox9.setname("홍길동")
>>> fox9.sum(2,3)
홍길동님! 2 + 3 = 5입니다
```

```
>>> fox8 = Service()
>>> fox8.sum(2,3)
Traceback (most recent call last):
  File "<input>", line 1, in <module>
  File "<input>", line 7, in sum
AttributeError: 'Service' object has no attribute 'name'
```

- 생성자 사용하기
  - ✓ 생성자는 클래스의 인스턴스를 생성하고 초기화하는 기능을 가짐.
  - ✓ 생성자 '__init__'의 첫 번째 매개변수는 관례적으로 'self'를 사용함.
  - ✓ 객체를 만드는 순간 'setname' 메서드 기능이 동작하도록 '__init__' 메서드로 초깃값을 설정함.

```
>>> class Service:
...     secret = "영구는 배꼽이 두 개다."
...     def __init__(self, name):
...         self.name = name
...     def sum(self, a,b):
...         result = a + b
...         print("%s님 %s + %s = %s입니다." % (self.name,a,b,result))
...
```

```
>>> fox8 = Service("홍길동")
>>> fox8.sum(2,3)
홍길동님 2 + 3 = 5입니다
```

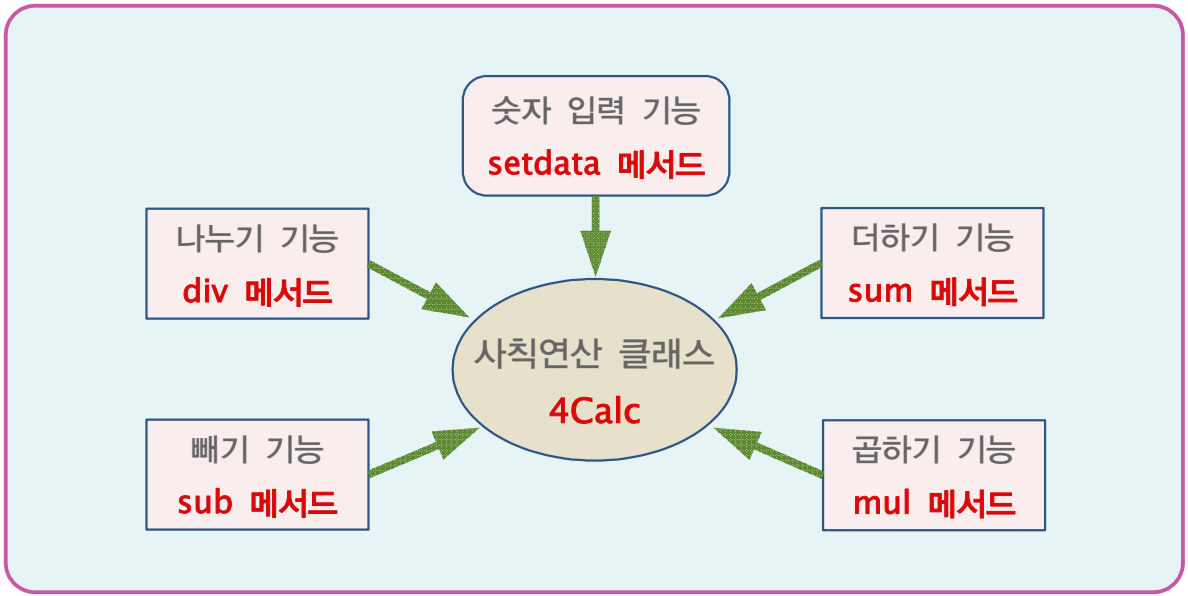
# 클래스 자세히 알기

- 클래스는 인스턴스를 만들어내는 설계도임.
- 클래스의 구조

```
class 클래스 이름(상속 클래스 명):
    <클래스 변수1>
    <클래스 변수N>
     ...................
    def 클래스 함수1(self, 인수1, 인수2, ... ):
        <수행할 문장1>
        <수행할 문장2>
         ...................
    def 클래스 함수2(self, 인수1, 인수2, ... ):
        <수행할 문장1>
        <수행할 문장2>
         ...................
```

# 사칙 연산 클래스 만들기

- 사칙연산 클래스 구상하기



- 클래스 구조 만들기 (단계별 접근)

1) 아무 기능이 없는 간단한 클래스

```
>>> class Calc:
...     pass
...
>>> a = Calc()
>>> type(a)
<class 'Calc'>
```

2) 객체에 숫자 지정할 수 있는 클래스

```
>>> class Calc:
...     def setdata(self, a1, a2):
...         self.first = a1
...         self.second = a2
...
```

✓ setdata 함수는 Calc 클래스의 메서드이다.

```
>>> a = Calc()
>>> a.setdata(3,4)
>>> a.first
3
>>> a.second
4
>>> b = Calc()
>>> b.setdata(5,7)
>>> b.first
5
>>> b.second
7
```

### 3) 사칙연산 기능 만들기

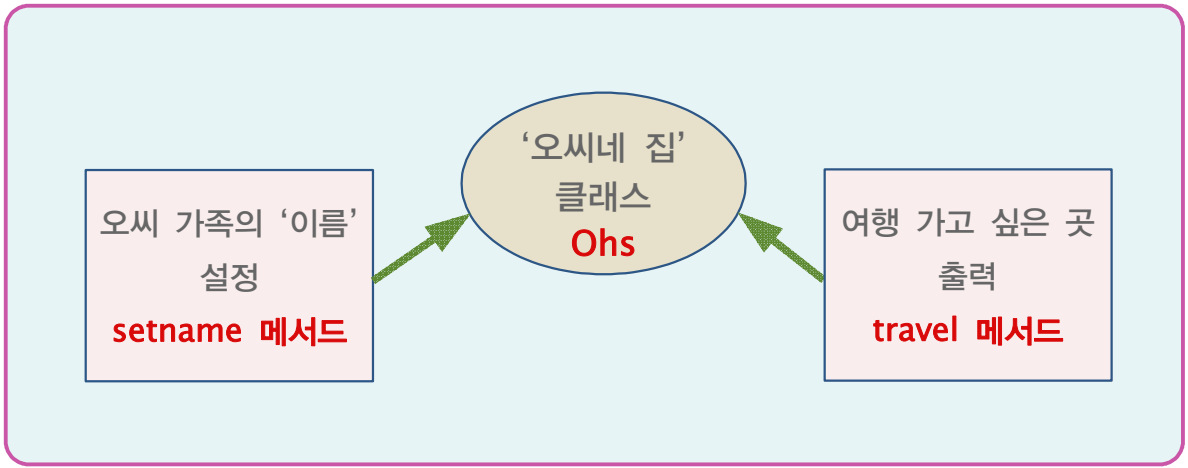
```
>>> class FourCal:
...     def setdata(self, a1,a2):
...         self.first = a1
...         self.second = a2
...     def sum(self):
...         result = self.first + self.second
...         return result
...     def sub(self):
...         result = self.first - self.second
...         return result
...     def mul(self):
...         result = self.first * self.second
...         return result
...     def div(self):
...         result = self.first / self.second
...         return result
...
```

```
>>> a = FourCal()
>>> b = FourCal()
>>> a.setdata(4,2)
>>> b.setdata(3,7)

>>> a.sum(), a.sub(), a.mul(), a.div()
(6, 2, 8, 2.0)

>>> b.sum(), b.sub(), b.mul(), b.div()
(10, -4, 21, 0.42857142857142855)
```

- 클래스에 의해 생성된 객체들은 다른 객체들과 완전히 다른 저장 공간을 가지고 독립적으로 동작함.

# '오씨네 집' 클래스 만들기

- 클래스 구상하기




1) 클래스 이름은 'Ohs'로 하고, 'oh1'이라는 인스턴스를 만들기.

```
>>> oh1 = Ohs()
```

2) ‘oh1.lastname’을 출력하면 ‘오’라는 성을 출력하도록 만들기.

```
>>> print(oh1.lastname)
'오'
```

3) 이름을 설정하면 ‘oh1.fullname’이 성을 포함한 이름을 출력하도록 만들기.

```
>>> oh1.setname("일숙")
>>> print(oh1.fullname)
'오일숙'
```

4) 여행 가고 싶은 장소를 입력하면 다음과 같이 출력해 주는 ‘travel’함수를 만들기.

```
>>> oh1.travel("제주도")
'오일숙, 제주도 여행을 가다.'
```

- 위 사항을 포함하는 'Ohs' 클래스는 다음과 같다.

```
>>> class Ohs:
...     lastname = "오"
...     def setname(self, name):
...         self.fullname = self.lastname + name
...     def travel(self, where):
...         print("%s, %s 여행을 가다." %(self.fullname, where))
...

>>> oh1 = Ohs()
>>> oh1.setname("일숙")
>>> oh1.travel("제주도")
오일숙, 제주도 여행을 가다.
```

- 초깃값 설정하기
  - ✓ 이 클래스에서 다음과 같이 실행하면 오류가 발생함.

```
>>> oh1 = Ohs()
>>> oh1.travel("제주도")
Traceback (most recent call last):
  File "<input>", line 1, in <module>
  File "<input>", line 6, in travel
AttributeError: 'Ohs' object has no attribute 'fullname'
```

  - ✓ 위 오류를 방지하기 위해 'oh2' 객체를 만드는 순간 'setname' 메서드 기능이 동작하도록 '__init__' 메서드로 초깃값을 설정하도록 함.

```
>>> class Ohs:
...     lastname = "오"
...     def __init__(self, name):
...         self.fullname = self.lastname + name
...     def travel(self, where):
...         print("%s, %s 여행을 가다." %(self.fullname, where))
...
```

```
>>> oh1 = Ohs()
Traceback (most recent call last):
  File "<input>", line 1, in <module>
TypeError: __init__() missing 1 required positional argument:
'name'
```

```
>>> oh1 = Ohs("일숙")
>>> oh1.travel("태국")
오일숙, 태국 여행을 가다.
```

## 클래스의 상속

클래스의 상속(inheritance)는 어떤 클래스를 만들 때 다른 클래스의 기능을 물려받을 수 있게 만드는 것임.

```
class 상속받을 클래스명(상속할 클래스 명):
```

- ‘오씨네 집’이라는 ‘Ohs’ 클래스를 상속 받아 ‘김씨네 집’이라는 ‘Kims’ 클래스를 만들기.

```
>>> class Kims(Ohs):
...     lastname = "김"
...
>>> kim1 = Kims("일동")
>>> kim1.travel("독도")
김일동, 독도 여행을 가다.
```

- 메서드 오버라이딩(overriding)
  - ✓ 상속 받을 대상인 클래스의 메서드와 이름은 그 상속자와 같지만 그 행동을 다르게 할 경우.
  - ✓ ‘김씨네 집’ 클래스가 ‘오씨네 집’에서 받은 클래스의 ‘travel’ 함수를 다음과 같이 다르게 적용하고자 함.

```
>>> kim1 = Kims("일동")
>>> kim1.travel("독도", 3)
김일동, 독도 여행을 3일 간다.
```

```
>>> class Kims(Ohs):
...     lastname = "김"
...     def travel(self,where, day):
...         print("%s, %s 여행을 %d일 간다" %(self.fullname, where, day))
...
```

## ❖ 모듈

- 모듈(Module)이란 함수, 변수 또는 클래스를 모아 놓은 파일이며 '.py' 확장자를 가짐.
- 파이썬 프로그램이나 라이브러리는 모듈이라는 단위로 구성됨.
- 모듈의 기능은 함수나 변수, 클래스들을 모아 놓아 선언하지 않고 바로 가져다 쓸 수 있도록 하는 것임.

## 간단한 모듈 만들고 불러오기

```
# mod1.py
def sum(a,b):
    return a+b
```

```
C:\Users\home\PycharmProjects\untitled> dir
......
2019-09-6  오후 03:55                40 mod1.py
......
```

- ✓ 모듈 파일 이름은 모듈 파일 안의 첫 번째 줄에서 주석을 알리는 해시 문자 '#' 다음의 파일 이름과 일치해야함.
- ✓ 'import'를 사용하여 모듈 파일을 탑재함.

```
>>> import mod1
>>> print(mod1.sum(3,4))
7
```

- import 문의 형태는 다음과 같음.

import *모듈파일_이름*

import *모듈파일_이름* as *모듈파일_이름의 별칭*

- mod1.py 파일에 함수 추가하기

```
# mod1.py
def sum(a,b):
    return a+b

def safe_sum(a,b):
    if type(a) != type(b):
        print("더할 수 있는 것이 아님!")
        return
    else:
        result = sum(a,b)
    return result
```

```
>>> import mod1
>>> print(mod1.safe_sum(3,4))
7

>>> print(mod1.safe_sum('a','b'))
ab

>>> print(mod1.safe_sum(3,'a'))
더할 수 있는 것이 아닙니다!
None

>>> print(mod1.sum(1,2))
3
```

# 모듈함수를 사용하는 또 다른 방법

from *모듈이름* import *모듈함수*

```
>>> from mod1 import sum
>>> sum(3,4)
7
>>> from mod1 import safe_sum
>>> safe_sum(1,"a")
더할 수 있는 것이 아닙니다!

>>> from mod1 import sum, safe_sum
>>> from mod1 import *
```

# 클래스나 변수 등을 포함한 모듈

```
# mod2.py
PI = 3.141592
class Math:
    def solv(self, r):
        return PI*(r**2)

def sum(a,b):
    return(a,b)
```

```
>>> import mod2
>>> print(mod2.PI)
3.141592
>>> a = mod2.Math()
>>> print(a.solv(2))
12.566368
>>> print(mod2.sum(mod2.PI, 4.4))
7.541592
```

# 새 파일 안에서 이전에 만든 모듈 불러오기

```
# modtest.py
import mod2
result = mod2.sum(3,4)
print(result)
```

```
>>> import modtest
7
```