[객체지향 프로그래밍:(OOP:Object Oriented Programming)]

객체 지향 프로그래밍 (OOP : Object Oriented Programming) 개념

- **OOP란?** : **"관련된 데이터와 처리부분를 모아서 다룬다"** 개념
- **데이터와 데이터 처리 함수(메서드)**를 하나의 묶음으로 관리
- 프로그램에서 사용되는 모든 것들을 객체(object)로 정의
- 객체를 만들기 위해 **'클래스(class)'** 라는 도구 제공
- 클래스로부터 객체를 생성하는 것을 **인스턴스(instance)**를 만든다라고 함
- *업체에서 제공하는 API가 대부분 클래스를 이용한 객체지향 프로그래밍 방식으로 개발*

< 객체 지향의 개념 >

1. `Abstraction(추상화)` : 핵심적인 개념/기능을 간추려 내는 것.
2. `Encapsulation(캡슐화)` : 데이터+알고리즘(코드), 데이터를 감추고 외부 세계와의 상호작용은 메소드를 통해서 함.
3. `Inheritance(상속)` : 하위 개념이 상위 개념의 속성 및 동작 등을 물려받는 것 (기존 코드를 재활용)
4. `Polymorphism(다형성)` : 하나의 메소드나 클래스가 있을 때 이것들이 다양한 방법으로 동작 (오버라이딩, 오버로딩)

클래스(class) ?

- 파이썬 클래스는 타입을 만들어내는 도구입니다. int, float, str, list, tuple (Built-in Class) 과 같이 사용자 정의 타입을 만들 수 있습니다
- **파이썬 클래스를 이용 프로그래머가 원하는 새로운 데이터 타입을 만들 수 있다**.
- 객체지향 프로그래밍에서 객체를 만들려면 객체를 바로 만들지 못하고, 항상 `클래스(class)라는 것을 만든 후에 그 클래스를 이용하여 객체`를 만들어야 한다.
- <클래스>는 다양한 데이터 타입을 만드는 `"청사진(설계도)"` 또는 `템플릿` 이고, 만들어진 `객체의 데이터 타입에 따라서 메소드`가 정해짐.
- class 서로 연관된 `데이터와 데이터를 처리하는 함수(메소드)` 를 하나의 집합에 모아놓은 것을 말한다. (데이터+메서드)

[built-in class vs. user-defined class]

[Built-in Class] 파이썬 9 가지 타입 (Built-in type)

- <class 'int'>, <class 'float'>, <class 'list'> ....등등 --> built-in Class

- **<클래스>** 는 다양한 데이터 타입의 객체를 만드는 **"청사진(설계도)"** 이고, 만들어진 각각 객체 타입 따라, 각각의 메소드 가 정해짐.
- 클래스는 (데이터+메서드)를 묶은 형태로 만들어 진다

- built-in Date Type

>>> a = 10
>>> type(a)
<class 'int'>
	>>> d = True
	>>> type(d)
	<class 'bool'>
		>>> g = (1,3,5)
		>>> type(g)
		<class 'tuple'>

>>> b = 3.5
>>> type(b)
<class 'float'>
	>>> e = 'hello'
	>>> type(e)
	<class 'str'>
		>>> h = {2,4,6,8}
		>>> type(h)
		<class 'set'>

>>> c = 3+7j
>>> type(c)
<class 'complex'>
	>>> f = [1,2,3,4]
	>>> type(f)
	<class 'list'>
		>>> i = {'baseball':9, 'soccer':11}
		>>> type(i)
		<class 'dict'>

- 타입에 따라, 각각 사용되는 연산자와 메소드(함수) 다르다.

- int, float, str, bool, list, tuple, dict 은 built-in Class들이다
- int(),float(), list(),..등등 는 객체를 만드는 Class 임

- 객체는 메모리에 저장된 데이터를 포함하는 어떤 상자 라고 생각

- 예: a = 100 : 변수 a가 가리키는 값 100 은 int라는 Class로 만들어진 객체 (a = int(123))

객체(Object) : 클래스의 틀로 찍어낸 실체

- **파이썬의 모든것들(숫자, 문자, 함수 등)은 여러 속성과 행동을 가지고 있는 객체이다**
- 파이썬에서는 **객체(object)라는 단위로 메모리에 저장** 한다 --> (**덩어리로 묶음 단위(객체)**)로 저장
- **클래스를 구체화한 객체를 인스턴스(instance)라고 부름** (객체와 인스턴스는 같은 뜻으로 사용)
- 객체는 값(value), 유형(type), id(identity) 이라는 세 특성이 있다.

객체 생성 : 10 vs. a = 10 vs. a = int(10)

- 숫자, 문자 객체 : 1. 리터럴 표현식 통해 생성. 2. 클래스 생성자를 통해서 생성하는 방법
- 변수a 값은 int라는 Class로 만들어진 형태로 저장
- int 클래스 생성자를 통해서 int 타입객체를 생성할 수 있음

In [ ]:
```
# 10 vs. a = 10  vs. a = int(10)
# 값이 10인 정수 객체를 생성, a에 할당

10 ; print(10, type(10))
a1 = 10; print("a1: ", type(a1))
a2 = int(10); print("a2 :", type(a2)) # 객체의 형을 명시적 객체를 생성
id(10), id(a1), id(a2)
```

- 객체 처리 (객체(변수값).메서드) : (. 표기로 메소드 호출)

- 객체.메소드 # 예: "welcom".upper()
- 객체(objects)의 함수(functions)를 메소드(methods)라고 부른다. ex) lower(), upper()

In [ ]:
```
# 5 vs 5.0 vs "5" : int, float, str
"5".upper() # 5.upper(), 5.0.upper() # 객체 종류에 따라 사용가능한 메소드 다르다
```

In [ ]:
```
# 객체(변수).메서드
s = "welcom"
s.upper()
```

In [ ]:
```
"welcom".upper()    # 객체(데이터/변수).메서드
```

In [ ]:
```
li = [1, 3, 5]
li.append(20)
li
```

In [ ]:
```
dir(li)
```

In [ ]:
```
animals = ['lion', 'tiger', 'cat', 'dog']
animals.sort()
animals
```

In [ ]:
```
print(type(animals)) # animals 객체의 자료형을 반환함
print(id(animals)) # animals 객체의 고유한 id를 반환함
```

In [ ]:
```
animals.append('rabbit')
animals
```

In [ ]:
```
s = animals.pop()
s.upper()
```

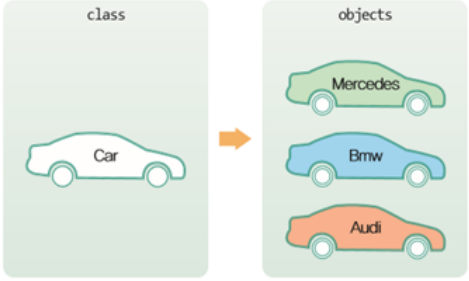
In [ ]:
```
s = 'tiger'
print(type(s)) # s 객체의 자료형을 반환함
```

In [ ]:
```
n = 200 # n = int(200)
print(type(n))
```

user-defined classes (클래스)

클래스와 객체의 관계 : 클래스는 설계도 , 객체는 물건 : 예) benz = Car()

- 객체 타입은 클래스에 의해 결정



- 객체 (object) 만들기

- Any data with state (attributes or value) and defined behavior (methods).
- 객체(object)는 어떠한 속성값과 행동을 가지고 있는 데이터.
- **객체(object)는 서로 연관된 데이터와 그 데이터를 조작하기 위한 함수를 하나의 집합에 모아놓은 것**
- **`객체 = 변수(property, 상태, 특징) + 함수(method, 행동, 동작)`**

In [ ]:
```
# 클래스 정의

class 붕어빵틀:
    def __init__(self, 앙꼬):
        self.앙꼬 = 앙꼬

# 붕어빵틀 만들어서, 다양한 종류 붕어빵 만들 수 있다

붕어빵1 = 붕어빵틀("초코맛")
붕어빵2 = 붕어빵틀("딸기맛")

print(붕어빵1.앙꼬)
print(붕어빵2.앙꼬)
```

In [ ]:
```
# 함수 만들기 [비교]

def fx():
    pass

def gx(x):
    return x
# ----------------

# 클래스 만들기 (사용자 정의 데이터타입 )

class Person:
    pass
```

In [ ]:
```
a = gx(3)# 함수 결과값 변수에 할당
print(a, type(a))

print(gx(5), type(gx(5)))

# ------ 클래스로 객체 생성 ---------------
cc = Person() # 객체 생성
print(cc, type(cc))
```

- Built-in Class vs. User-defined Class

In [ ]:
```
# Built-in Class

i = int() # x = int(10) <-- x = 10
print(type(int), type(i))

# ------------------------

# user-defined Class
class A:
    pass
```

```
a = A()
print(type(A), type(a))
```

[비교] 일반, 함수, 클래스 (예) 면적구하기

- 가로 길이와 세로 길이라는 두 개의 데이터를 넣을 변수
- 두 길이를 곱해서 면적을 구하는 함수

In [ ]:
```
# 1. 일반 (면적구하기)
h = 10
v = 20

area = h * v
print(area)
```

In [ ]:
```
# 2. 함수 사용
def area(h, v):
    return h * v

a = area(10, 20)
print(a)
```

In [ ]:
```
# 3. 클래스 사용 (객체지향 코드)

# Class 정의
class Rect:
    def __init__(self, h, v):
        self.h = h
        self.v = v

    def area(self):
        return self.h * self.v

# 객체생성
r = Rect(10,20)
a = r.area()

print(a)
print(r.h, r.v)
```

[클래스 선언, 객체 생성 및 사용]

- 클래스는 데이타를 표현하는 속성(attribute)과 행위를 표현하는 메서드(method)를 포함하는 논리적인 컨테이너이다.
- 클래스 = 속성(Attribute): 자료구조)] + 메소드(method)]
- 클래스 멤버 : 메서드(method), 프로퍼티(property), 클래스 변수(class variable), 인스턴스 변수(instance variable) 등 다양한 종류 멤버

- 객체를 만들려면 먼저 클래스를 선언해야 함
- 클래스는 객체를 만들기 위한 기본 틀 (설계도)
- 클래스 생성자를 통해 객체 생성하기
- 객체는 클래스에서 생성하므로 객체를 클래스의 **인스턴스(Instance)**라고 함

단계	작업	형식	예
1단계	클래스 선언	class 클래스명 : # 필드 선언 # 메서드 선언	class Car : color = " " def upSpeed(self, value) : ...
		↓	
2단계	인스턴스 생성	인스턴스 = 클래스명()	myCar1 = Car()
		↓	
3단계	필드나 메서드 사용	인스턴스.필드명 = 값 인스턴스.메서드()	myCar1.color = "빨강" myCar1.upSpeed(30)

1. 클래스 정의/선언 (class 클래스명) : class Car:

2. 객체 (인스턴스) 생성 (객체명 = 클래스명()) : myCar = Car()

3-1. 필드 사용 (객체명.필드명 = 속성값) : myCar.color = "빨강"

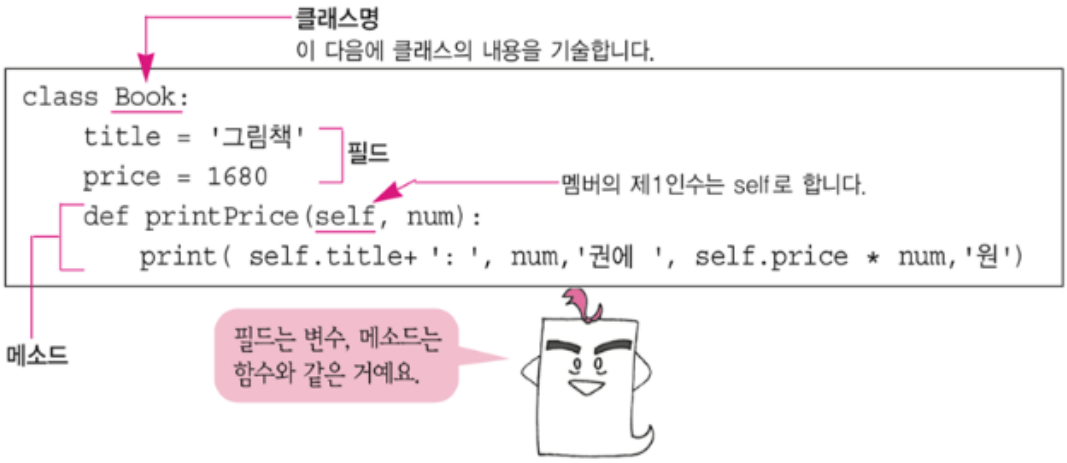
3-2. 메서드 사용 (객체명.메서드()) : myCar.upSpeed(30)

- 메서드 호출

- 메서드 호출은 두 단계 과정으로 이뤄진다.
- 메서드가 있는지 확인(lookup): . 연산자
- 메서드 호출: ( )연산자

클래스 구성

- 클래스는 `데이타를 표현하는 필드/속성(attribute)과 행위를 표현하는 메서드(method)` 를 포함하는 논리적인 컨테이너이다.
- class 의 내부 블록에 **필드 (변수), 메소드(함수)**를 정의 (**class = 필드(변수) + 메소드(함수)**)
- 클래스는 메서드(method), 프로퍼티(property), 클래스 변수(class variable), 인스턴스 변수(instance variable), 초기자(initializer), 소멸자(destructor) 등 다양한 종류의 멤버들로 구분할 수 있다.



- 메소드를 정의할 때 **첫번째 인자는 반드시 self**.
- 멤버 변수는 **self.필드** 로 접근한다.

클래스정의/선언 방법, 객체사용

객체 생성 (생성자 사용)

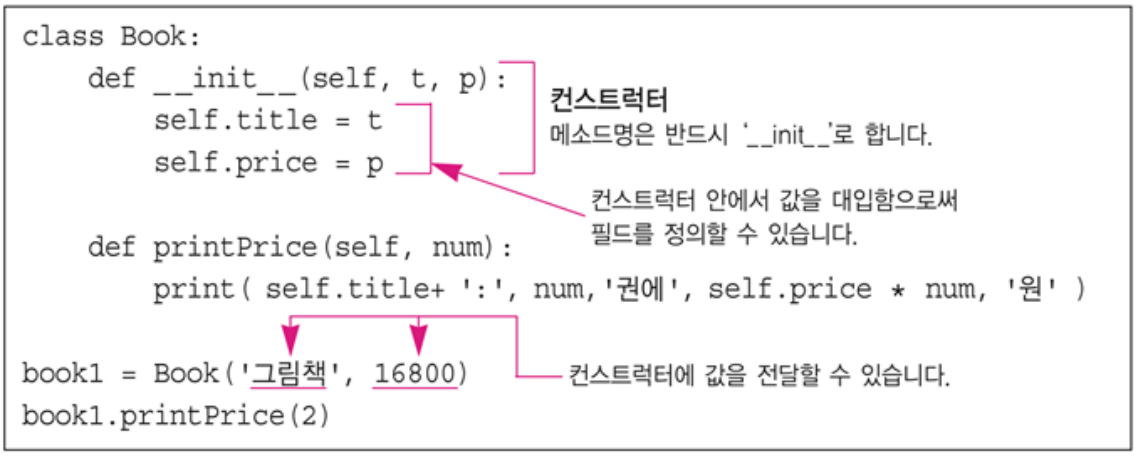
- 생성자사용 파이썬에서 클래스를 정의하는 문법은 다음과 같다.

```
class 클래스이름(object):
    def __init__(self, 속성값1, 속성값2, 속성값3):
        self.속성이름1 = 속성값1
        self.속성이름2 = 속성값2
        self.속성이름3 = 속성값3
```

- **객체가 생성할때 속성값을 지정하면 속성초기화 가능**
- 이때 속성값 인수는 필요하지 않다면 없어도 된다.
- 여기에서 `class` 블럭 안에 정의된 `__init__` 란 함수는 생성자(constructor)라고 하며 클래스 정의에서 가장 중요한 함수.

- 객체를 생성할 때는 `클래스이름` 을 함수처럼 호출해야 하는데, 이때 실제로는 `__init__` 로 정의된 생성자 함수가 호출된다.

- 생성자 함수 내부에서는 생성자를 호출할 때 넣은 입력 변수, 즉 인자의 값을 속성값으로 저장한다.



In [ ]:
```
# 클래스 정의 : With Constructor(생성자)
class Book:
    def __init__(self, t, p):    # 생성자 : __init__
        self.title = t
        self.price = p

    def printPrice(self, num):
        print(self.title +':', num, '권에', self.price * num, '원')

# ----- 객체 Test ---------
test = Book("테스트책", 20000)
test.printPrice(3)
```

In [ ]:
```
# 클래스 정의 : With Constructor(생성자)
class Book2:
    title = "hihi"
    price = 500
    def printPrice(self, num):
        print(self.title +':', num, '권에', self.price * num, '원')

# ----- 객체 Test ---------
test = Book2()
test.printPrice(3)
```

In [ ]:
```
## 객체 생성, 사용

# 객체생성 + 초기화
book1 = Book('그림책', 16800)
book2 = Book('소설책', 12000)
book3 = Book('전공책', 22000)

# 객체사용 (객체.메소드)
book1.printPrice(2)
book2.printPrice(3)
book3.printPrice(1)
```

In [ ]:
```
dir(list) # 비교
```

In [ ]:
```
dir(Book)
```

- 사칙연산 (with 생성자(Constructor))

In [ ]:
```
class FourCal:
    def __init__(self, x, y):
        self.x = x
```

```
        self.y = y
    def add(self):
        result = self.x + self.y
        return result
    def mul(self):
        result = self.x * self.y
        return result
    def sub(self):
        result = self.x - self.y
        return result
    def div(self):
        if self.y != 0:
            result = self.x / self.y
        else :
            result = "ERROR"
        return result
```

In [ ]:
```
a = FourCal(4, 2) # 객체생성 + 초기화
b = FourCal(5, 0)

print(a.x)
print(a.y)

print(b.x)
print(b.y)
```

In [ ]:
```
a.add(), a.mul(), a.sub(), a.div()
```

In [ ]:
```
b.add(), b.mul(), b.sub(), b.div()
```

[실습 : 클래스]

- 학생들의 성적 관리를 위해 클래스 만듬
- 데이터 속성 : 학번, 이름, 성적
- 메소드:
  - info() 학번, 이름, 성적 보여줌
  - getScore() 점수 전달
- 실행:
  - 3명 학생 데이터 입력
  - 입력데이터 표시
  - 3명의 학생 평균 값과 최고 성적 출력하는 코드 작성

In [ ]:
```
class Student:
        """ 학생들의 성적을 관리하기 위한 Student 클래스입니다.
        학생들의 학번, 이름, 성적을 데이터로 저장합니다. """

        def __init__(self, no, name, score):
                self.no = no
                self.name = name
                self.score = score

        def info(self):
                print('- 학번:{}, 이름:{}, 성적:{}'.format(self.no, self.name,

        def getScore(self):
                return self.score
```

In [ ]:
```
# main

# 인스턴스화, 데이터 입력
s1 = Student(201812345, '홍길동', 90)
s2 = Student(201811111, '이철수', 85)
s3 = Student(201800100, '김영희', 93)

# 출력
# 입력 데이터 표시
s1.info( )
s2.info( )
s3.info( )

# 평균, 최고값 표시

a1 = s1.getScore( )
a2 = s2.getScore( )
a3 = s3.getScore( )

print()
print('# 평균 점수 : {:.2f}'.format((a1 + a2 + a3)/3))
print('# 최고 점수 : {}'.format(max(a1, a2, a3)))
```

- 클래스 상속

- 다른 클래스의 멤버를 상속받을 수 있다. --> 이를 클래스의 상속이라함
- 물려주는 클래스(부모클래스, Parent Class, Super class) --> 상속받는/물려받는 클래스(자식 클래스, Child class, sub class)
- **서브 클래스는 상속받은 항목에 메소드, 필드를 추가한다**

In [ ]:
```
class A:
    print("클래스 A")

class B(A):
    pass

b = B() # B클래스
```

In [ ]:
```
# 클래스 상속
class Country: # Super Class
    name = '국가명'
    population = '인구'
    capital = '수도'

    def show(self):
        print('국가 클래스의 메소드입니다.')

class Korea(Country): # Sub Class 상속
    def __init__(self, name, cap):
        self.name = name
        self.capital = cap

    def show_name(self):
        print('국가 이름은 : ', self.name)
```

In [ ]:
```
a = Korea('대한민국','서울') # 인스턴스

a.show()
```

```
a.show_name()

print(a.capital)
print(a.name)
```

---------- End ----------