AI 이노베이션스퀘어

- 파이썬 프로그래밍 -

Day 5
Python

전준헌

객체(object)

OOP

- 객체지향 프로그래밍(object-oriented Programming) :
  - 객체를 이용하여 프로그래밍 하는것

OOP

- 객체란?
  - 객체는 하나의 물건이라고 생각하면 된다.
  - 객체는 속성(attribute)과 동작(action)을 가지고 있다.

OOP

- 객체지향 프로그래밍
  - 클래스(class) 를 먼저 만들고 (설계도 작성)
  - 객체(object)를 만든다. (설계도로 부터 객체를 생성)

OOP

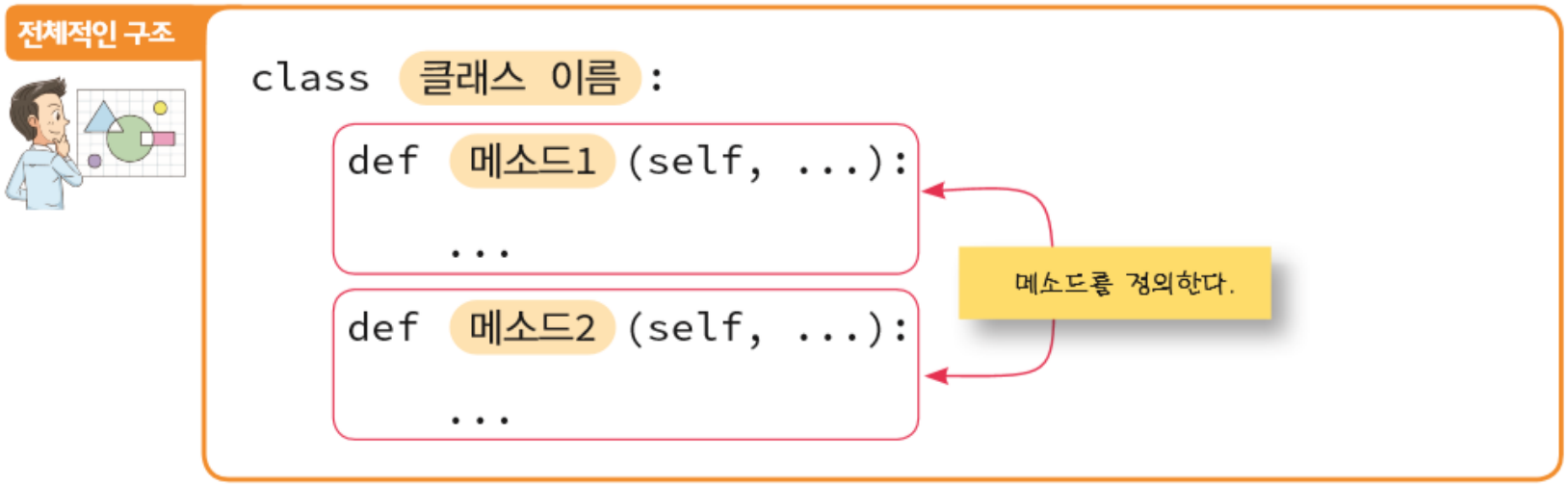
- 클래스(Class) 와 객체(Object)
  - 객체에 대한 설계도를 클래스(class)라고 한다.
  - 클래스로부터 만들어지는 각각의 객체를 그 클래스의 인스턴스(instance)라고 한다.

OOP

- 우리는 하나의 클래스로 여러 개의 객체를 생성할 수 있다

OOP

- 클래스 작성하기



OOP

- 클래스를 만들고, 객체를 생성해보자

In [4]:
```
class Person:
    pass
```

In [2]:
```
p = Person()
print(p)
```

```
<__main__.Person object at 0x0000014F665B07F0>
```

OOP

- 클래스 내 함수(메소드)에서 사용되는 self
  - say_hi 메소드는 어떤값(매개변수)도 받지 않음
  - say_hi 메소드는 함수정의에 self 를 매개변수 가지고 있음

```
In [7]: class Person:
            def say_hi(self):
                print('hello~')

In [8]: p = Person()

In [9]: p.say_hi()
hello~
```

OOP

```
class Car:
    def __init__(self, speed, color, model):
        self.speed = speed
        self.color = color
        self.model = model

    def drive(self):
        self.speed = 60

myCar = Car(0, "blue", "E-class")

print("자동차 객체를 생성하였습니다.")
print("자동차의 속도는", myCar.speed)
print("자동차의 색상은", myCar.color)
print("자동차의 모델은", myCar.model)
print("자동차를 주행합니다.")
myCar.drive()
print("자동차의 속도는", myCar.speed)
```

```
자동차 객체를 생성하였습니다.
자동차의 속도는 0
자동차의 색상은 blue
자동차의 모델은 E-class
자동차를 주행합니다.
자동차의 속도는 60
```

OOP

- Counter 클래스를 작성하여 보자.
  Counter 클래스는 기계식 계수기를 나타내며 경기장이나 콘서트에 입장하는 관객 수를 세기 위하여 사용할 수 있다.

OOP

- Count 클래스

```
class Counter:
    def reset(self):
        self.count = 0
    def increment(self):
        self.count += 1
    def get(self):
        return self.count
```

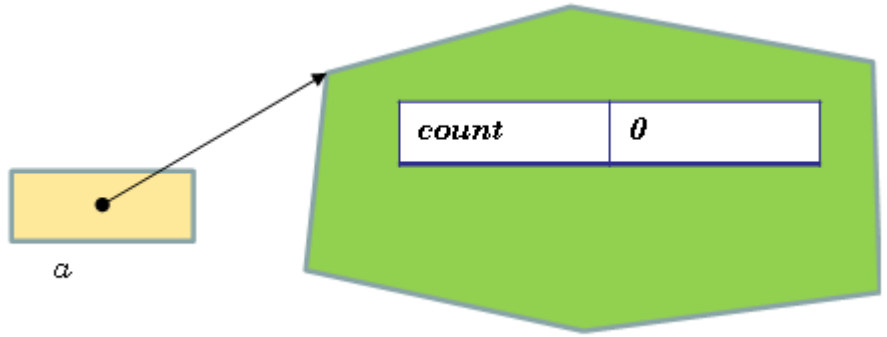
OOP

- 객체 생성

```
a = Counter()

a.reset()
a.increment()
print("카운터 a의 값은", a.get())
```

```
카운터 a의 값은 1
```

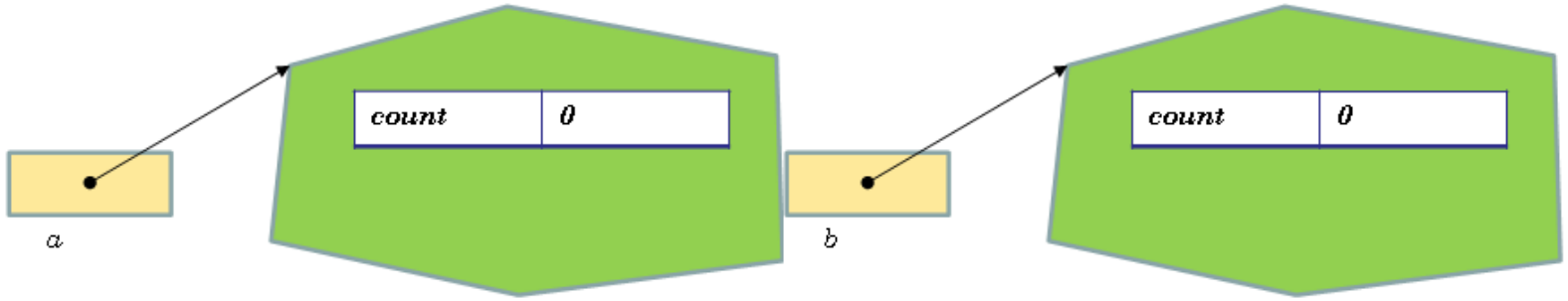


OOP

- 객체 2개 생성하기

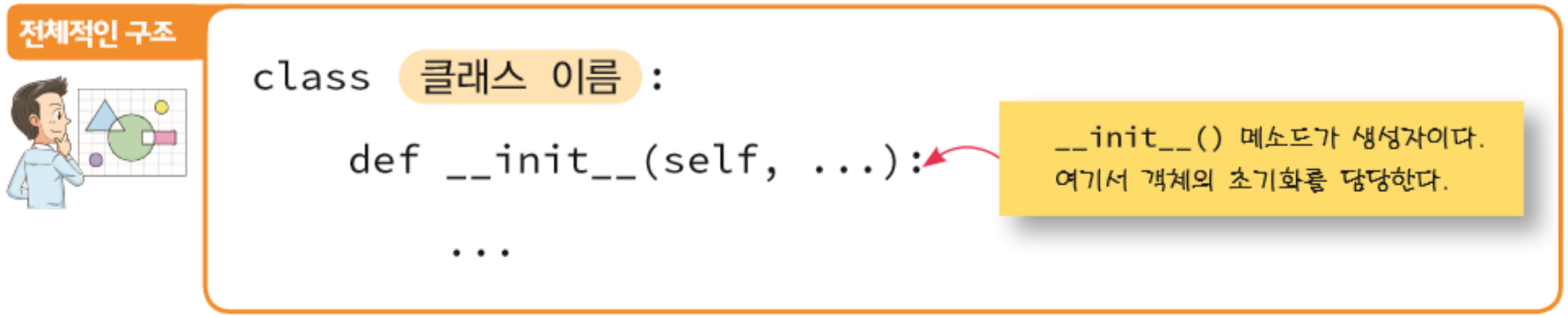
```
a = Counter()
b = Counter()

a.reset()
b.reset()
```



OOP

- __init__(self)의 예




```
class Counter:
    def __init__(self) :
        self.count = 0
    def reset(self) :
        self.count = 0
    def increment(self):
        self.count += 1
    def get(self):
        return self.count
```

OOP

- init 메소드는 클래스가 인스턴스화 될 때 호출
- 객체가 생성시 초기화 명령이 필요할 때 유용하게 사용
- init 의 앞과 뒤에 있는 밑줄은 두 번씩 입력

In [10]:
```
class Person:
    def __init__(self, name):
        self.name = name
    def say_hi(self):
        print('hello~', self.name)
```

In [12]:
```
p = Person('길동')
p.say_hi()
```

```
hello~ 길동
```

기본 생성자 예제

- 매개 변수가 self 만 있는 생성자

```
 1  ## 클래스 선언 부분 ##
 2  class Car :
 3
 4
 5
 6
 7
 8
 9
10      def upSpeed(self, value) :
11          self.speed += value
12
13      def downSpeed(self, value) :
14          self.speed -= value
15
16  ## 메인 코드 부분 ##
17  myCar1 = Car()
18  myCar2 = Car()
19
20  print("자동차1의 색상은 %s이며, 현재 속도는 %dkm입니다." % (myCar1.color, myCar1.speed))
21  print("자동차2의 색상은 %s이며, 현재 속도는 %dkm입니다." % (myCar2.color, myCar2.speed))
```

출력 결과

```
자동차1의 색상은 빨강이며, 현재 속도는 0km입니다.
자동차2의 색상은 빨강이며, 현재 속도는 0km입니다.
```

기본 생성자 예제 솔루션

- 매개 변수가 self 만 있는 생성자

```
## 클래스 선언 부분 ##
class Car :



    def __init__(self) :
        self.color = "빨강"
        self.speed = 0

    def upSpeed(self, value) :
        self.speed += value

    def downSpeed(self, value) :
        self.speed -= value

## 메인 코드 부분 ##
myCar1 = Car()
myCar2 = Car()

print("자동차1의 색상은 %s이며, 현재 속도는 %dkm입니다." % (myCar1.color, myCar1.speed))
print("자동차2의 색상은 %s이며, 현재 속도는 %dkm입니다." % (myCar2.color, myCar2.speed))
```

출력 결과

```
자동차1의 색상은 빨강이며, 현재 속도는 0km입니다.
자동차2의 색상은 빨강이며, 현재 속도는 0km입니다.
```

매개 변수가 있는 생성자 예제

```
## 클래스 선언 부분 ##
class Car :



                -03.py의 upSpeed(), downSpeed() 함수와 동일

## 메인 코드 부분 ##
myCar1 = Car("빨강", 30)
myCar2 = Car("파랑", 60)

# Code12-03.py의 20~21행과 동일
```

출력 결과

```
자동차1의 색상은 빨강이며, 현재 속도는 30km입니다.
자동차2의 색상은 파랑이며, 현재 속도는 60km입니다.
```

매개 변수가 있는 생성자 예제

```
1  ## 클래스 선언 부분 ##
2  class Car :
3
4
5
6      def __init__(self, value1, value2) :
7          self.color = value1
8          self.speed = value2
9
10             2-03.py의 upSpeed(), downSpeed() 함수와 동일
11
12 ## 메인 코드 부분 ##
13 myCar1 = Car("빨강", 30)
14 myCar2 = Car("파랑", 60)
15
16 # Code12-03.py의 20~21행과 동일
```

출력 결과

```
자동차1의 색상은 빨강이며, 현재 속도는 30km입니다.
자동차2의 색상은 파랑이며, 현재 속도는 60km입니다.
```

OOP

- 클래스 변수
  - 공유됨
  - 모든 인스턴스들이 접근할 수 있음
  - 한 개만 존재
  - 객체가 값을 변경하면 다른 인스턴스에 적용됨

OOP

- 객체 변수
  - 개별 객체 / 인스턴스에 속함 (독립)
  - 다른 인스턴스들이 접근할 수 없음
  - self 에 연결(self.name)

OOP

- 클래스 함수
  - 클래스가 가진 함수
  - '클래스가 가진 기능' 명시적으로 나타냄
  - 데코레이터 (decorator) : @classmethod

클래스 함수 만들기

```
class 클래스 이름:
    @classmethod
    def 클래스 함수(cls, 매개변수):
        pass
```

클래스 함수 호출하기

```
클래스 이름.함수 이름(매개변수)
```

OOP

```
class Robot:
    '''로봇 클래스'''
    population = 0                          #클래스 변수

    def __init__(self, name):
        '''먼저 시작되는 메소드'''
        self.name = name                    #객체변수
        print('시작합니다.', self.name)
        Robot.population += 1

    def die(self):
        '''로봇 파괴'''
        print(self.name, '파괴되었습니다.')

        Robot.population -= 1
        if Robot.population == 0:
            print(self.name, '마지막 로봇입니다.')
        else:
            print(Robot.population, '남았습니다.')
```

OOP

```
def say_hi(self):
    '''인사'''
    print(self.name, '반갑습니다.')

@classmethod
def how_many(cls):
    print(cls.population, '가지고 있습니다.')
```

OOP

```
In [44]: droid1 = Robot('길동1')
```

```
시작합니다. 길동1
```

```
In [45]: droid1.say_hi()
```

```
길동1 반갑습니다.
```

```
In [46]: Robot.how_many()
```

```
1 가지고 있습니다.
```

```
In [47]: droid2 = Robot('길동2')
```

```
시작합니다. 길동2
```

```
In [47]: droid2 = Robot('길동2')
```

```
시작합니다. 길동2
```

```
In [48]: droid2.say_hi()
```

```
길동2 반갑습니다.
```

```
In [49]: Robot.how_many()
```

```
2 가지고 있습니다.
```

```
In [50]: droid2.die()
```

```
길동2 파괴되었습니다.
1 남았습니다.
```

```
In [51]: Robot.how_many()
```

```
1 가지고 있습니다.
```

OOP

- 정보 은닉
  - 구현의 세부 사항을 클래스 안에 감추는 것

OOP

- 파이썬에서 클래스 private 멤버로 정의 하려면
  변수 이름 앞에 __을 붙이면 된다.
- 이 변수는 클래스 내부에서만 접근 될 수 있다.

```
class Student:
        def __init__(self, name=None, age=0):
                self.__name = name
                self.__age = age

obj=Student()
print(obj.__age)
```

```
...
AttributeError: 'Student' object has no attribute '__age'
```

OOP

- 접근자와 설정자
  - 하나는 인스턴스 변수값을 반환하는 **접근자(getters)**이고 또 하나는 인스턴스 변수값을 설정하는 **설정자(setters)**이다.
  - 프라이빗 변수 값 추출하거나 변경할 목적으로 간접적으로 속성에 접근하도록 하는 함수

OOP

```
class Student:
        def __init__(self, name=None, age=0):
                self.__name = name
                self.__age = age

        def getAge(self):               # 접근자
                return self.__age

        def getName(self):
                return self.__name

        def setAge(self, age):          # 설정자
                self.__age=age

        def setName(self, name):
                self.__name=name
```

OOP

In [60]:
```
s = Student()
```

In [61]:
```
print(s.__age)
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
<ipython-input-61-37cec98c33ac> in <module>
----> 1 print(s.__age)

AttributeError: 'Student' object has no attribute '__age'
```

In [71]:
```
s.setAge(20)
```

In [76]:
```
print(s.getAge())
```

```
20
```

OOP 실습 예제

- 원을 클래스도 표시해보자.
  원은 반지름(radius)을 가지고 있다. 원의 넓이와 둘레를 계산하는 메소드도 정의해보자. 설정자와 접근자 메소드도 작성한다.

```
원의 반지름= 10
원의 넓이= 314.0
원의 둘레= 62.80000000000004
```

```
In [78]: c1=Circle(10)
         print("원의 반지름=", c1.getRadius())
         print("원의 넓이=", c1.calcArea())
         print("원의 둘레=", c1.calcCircum())
```

OOP 실습 예제 (솔루션)

```
class Circle:
  def __init__(self, radius=1.0):
    self.__radius = radius

  def setRadius(self, r):
    self.__radius = r

  def getRadius(self):
    return self.__radius

  def calcArea(self):
    area = 3.14*self.__radius*self.__radius
    return area
  def calcCircum(self):
    circumference = 2.0*3.14*self.__radius
    return circumference
```

OOP 실습 예제 (솔루션)

In [78]:

```
c1=Circle(10)
print("원의 반지름=", c1.getRadius())
print("원의 넓이=", c1.calcArea())
print("원의 둘레=", c1.calcCircum())
```

```
원의 반지름= 10
원의 넓이= 314.0
원의 둘레= 62.800000000000004
```

OOP 실습 예제

- 우리는 은행 계좌에 돈을 저금할 수 있고 인출할 수도 있다.
  은행 계좌를 클래스로 모델링하여 보자. 은행 계좌는 현재 잔액(balance)만을 인스턴스 변수로 가진다. __init__와 인출 메소드 withdraw()와 저축 메소드 deposit() 만을 가정하자.

```
통장에서  100 가 입금되었음
통장에  10 가 출금되었음
```

```
In [81]: a = BankAccount()

In [82]: a.deposit(100)
         통장에서  100 가 출금되었음
Out[82]: 100

In [83]: a.withdraw(10)
         통장에  10 가 입금되었음
Out[83]: 90
```

OOP 실습 예제(솔루션)

```
class BankAccount:
    def __init__(self):
        self.__balance = 0

    def withdraw(self, amount):
        self.__balance -= amount
        print("통장에 ", amount, "가 출금되었음 " )
        return self.__balance

    def deposit(self, amount):
        self.__balance += amount
        print( " 통장에서  " , amount,  " 가 입금되었음")
        return self.__balance
```

OOP

- 객체를 함수로 전달할 때
  - 우리가 작성한 객체가 전달되면 함수가 객체를 변경할 수 있다.

```
# 사각형을 클래스로 정의한다.
class Rectangle:
        def __init__(self, side=0):
                self.side = side

        def getArea(self):
                return self.side*self.side

# 사각형 객체와 반복횟수를 받아서 변을 증가시키면서 면적을 출력한다.
def printAreas(r, n):
        while n >= 1:
                print(r.side, "\t", r.getArea())
                r.side = r.side + 1
                n = n - 1
```

OOP

- 객체를 함수로 전달할 때

```
# printAreas()을 호출하여서 객체의 내용이 변경되는지를 확인한다.
myRect = Rectangle();
count = 5
printAreas(myRect, count)
print("사각형의 변=", myRect.side)
print("반복횟수=", count)
```

In [85]:
```
# printAreas()을 호출하여서 객체의 내용이 변경되는지를 확인한다.
myRect = Rectangle();
count = 5
printAreas(myRect, count)
print("사각형의 변=", myRect.side)
print("반복횟수=", count)
```

```
0       0
1       1
2       4
3       9
4       16
사각형의 변= 5
반복횟수= 5
```

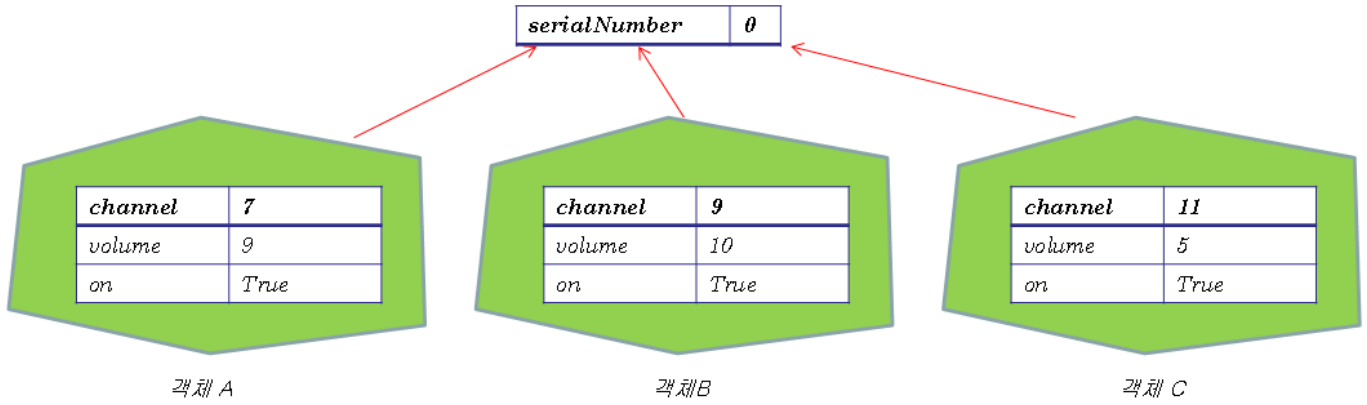
OOP

- 정적 변수
  - 이들 변수는 모든 객체를 통틀어서 하나만 생성되고 모든 객체가 이것을 공유하게 된다. 이러한 변수를 정적 멤버 또는 클래스 멤버(class member)라고 한다.

OOP

- 정적 변수

```
class Television:
        serialNumber = 0          # 이것이 정적 변수이다.
        def __init__(self):
                Television.serialNumber += 1
        self.number = Television.serialNumber
                ...
```



OOP

- 특수 메소드
  - 파이썬에는 연산자(+, −, *, /)에 관련된 **특수 메소드 (special method)**가 있다.

```
class Circle:
        ...
        def __eq__(self, other):
                return self.radius == other.radius

c1 = Circle(10)
c2 = Circle(10)
if c1 == c2:
        print("원의 반지름은 동일합니다. ")
```

OOP

연산자	메소드	설명
x + y	__add__(self, y)	덧셈
x - y	__sub__(self, y)	뺄셈
x * y	__mul__(self, y)	곱셈
x / y	__truediv__(self, y)	실수나눗셈
x // y	__floordiv__(self, y)	정수나눗셈
x % y	__mod__(self, y)	나머지
divmod(x, y)	__divmod__(self, y)	실수나눗셈과 나머지
x ** y	__pow__(self, y)	지수
x << y	__lshift__(self, y)	왼쪽 비트 이동
x >> y	__rshift__(self, y)	오른쪽 비트 이동
x <= y	__le__(self, y)	less than or equal(작거나 같다)
x < y	__lt__(self, y)	less than(작다)
x >= y	__ge__(self, y)	greater than or equal(크거나 같다)
x > y	__gt__(self, y)	greater than(크다)
x == y	__eq__(self, y)	같다
x != y	__neq__(self, y)	같지않다

OOP 예제

```
class Vector2D :
    def __init__(self, x, y):
        self.x = x
        self.y = y

    def __add__(self, other):
        return Vector2D(self.x + other.x, self.y + other.y)

    def __sub__(self, other):
        return Vector2D(self.x - other.x, self.y - other.y)

    def __eq__(self, other):
        return self.x == other.x and self.y == other.y

    def __str__(self):
        return '(%d, %d)' % (self.x, self.y)

u = Vector2D(0,1)
v = Vector2D(1,0)
w = Vector2D(1,1)
a = u + v
print( a)
```

OOP

- 파이썬에서의 변수의 종류
  - 지역 변수 – 함수 안에서 선언되는 변수
  - 전역 변수 – 함수 외부에서 선언되는 변수
  - 인스턴스 변수 – 클래스 안에 선언된 변수, 앞에 self.가 붙는다.

OOP

- 핵심 정리
  - 클래스는 속성과 동작으로 이루어진다. 속성은 인스턴스 변수로 표현되고 동작은 메소드로 표현된다.
  - 객체를 생성하려면 생성자 메소드를 호출한다. 생성자 메소드는 __init__() 이름의 메소드이다.
  - 인스턴스 변수를 정의하려면 생성자 메소드 안에서 self.변수이름 과 같이 생성한다.

상속

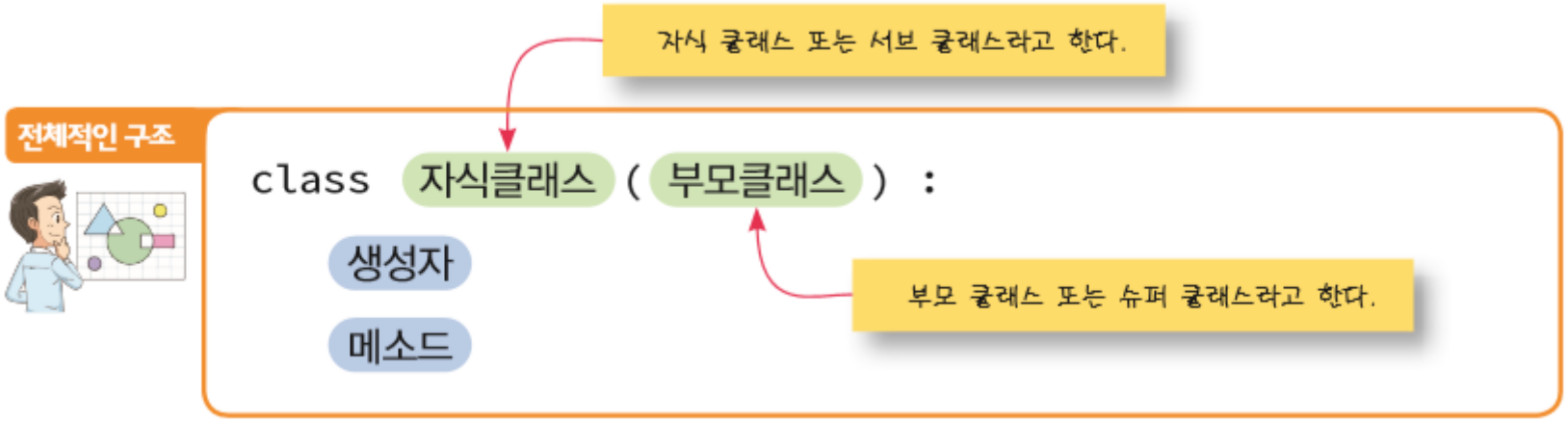
- 상속이란?
  - 상속은 클래스를 정의할 때 부모 클래스를 지정하는 것이다. 자식 클래스는 부모 클래스의 메소드와 변수들을 사용할 수 있다.

상속

- 상속(inheritance)은 기존에 존재하는 클래스로부터 코드와 데이터를 이어받고 자신이 필요한 기능을 추가하는 기법이다.

상속

- 상속 구현하기


자식 클래스 또는 서브 클래스라고 한다.
전체적인 구조
class 자식클래스 (부모클래스) :
생성자
메소드
부모 클래스 또는 슈퍼 클래스라고 한다.

상속 예제

```
# 일반적인 운송수단을 나타내는 클래스이다.
class Vehicle:
        def __init__(self, make, model, color,  price):
                self.__make = make                      # 메이커
                self.model = model              # 모델
                self.color = color              # 자동차의 색상
                self.price = price              # 자동차의 가격

        def setMake(self, make):    # 설정자 메소드
                self.__make = make

        def getMake(self):                      # 접근자 메소드
                return self.__make

        # 차량에 대한 정보를 문자열로 요약하여서 반환한다.
        def getDesc(self):
                return "차량 =("+str(self.__make)+","+\
                        str(self.model)+","+\
                        str(self.color)+","+\
                        str(self.price)+")"
```

상속 예제

```
class Truck(Vehicle) :                                          # ①
        def __init__(self, make, model, color, price, payload):
                super().__init__(make, model, color, price)     # ②
                self.payload=payload                            # ③

        def setPayload(self, payload):          # 설정자 메소드
                self.payload=payload

        def getPayload(self):                   # 접근자 메소드
                return self.payload
```

상속 예제

```
In [58]: myTruck = Truck("Tisla", "Model S", "white", 10000, 2000)

In [59]: print(myTruck.getDesc())
         차량 =(Tisla,Model S,white,10000)

In [60]: myTruck.setMake("Tesla")

In [61]: myTruck.setPayload(2000)

In [62]: print(myTruck.getDesc())
         차량 =(Tesla,Model S,white,10000)
```

상속과 다형성

- 부모 클래스의 생성자를 명시적으로 호출

```
class ChildClass(ParentClass) :
        def __init__(self):
                super().__init__()
                ...
```

- super() 함수의 반환 값을 상위클래스의 객체로 간주
- super()함수는 부모 클래스의 객체 역할을 하는 내장 함수

상속과 다형성

- 부모 클래스의 생성자를 명시적으로 호출

```
class ChildClass(ParentClass) :
        def __init__(self):
                super().__init__()
                ...
```

```
class ChildClass(ParentClass) :
        def __init__(self):
                ParentClass.__init__(self)
                ...
```

상속 실습

- 교수, 학생 명부 작성
  - 교수 / 학생 : 이름, 나이, 주소
  - 교수 : 월급, 과목
  - 학생 : 성적, 학년

상속 실습

- 방법 1 : 2개의 클래스 작성
  - 교수 클래스 : 이름, 나이, 주소, 월급, 과목
  - 학생 클래스 : 이름, 나이, 주소, 성적, 학년

- 방법 2 : 공통 클래스와 상속
  - 공통 클래스 : 이름, 나이, 주소
  - 교수 클래스 : 월급, 과목
  - 학생 클래스 : 성적, 학년

상속 실습

```
class SchoolMember:
    def __init__(self, name, age):
        self.name = name
        self.age = age
        print('공통 멤버: {}'.format(self.name))
    def tell(self):
        print('이름: "{}" 나이: {}'.format(self.name, self.age), end =' ')
```

상속 실습

```
class Teacher(SchoolMember):
    def __init__(self, name, age, salary):
        SchoolMember.__init__(self, name, age)
        self.salary = salary
        print('교수 시작: {}'.format(self.name))
    def tell(self):
        SchoolMember.tell(self)
        print('월급: {:d}'.format(self.salary))
```

상속 실습

```
class Student(SchoolMember):
    def __init__(self, name, age, grade):
        SchoolMember.__init__(self, name, age)
        self.grade = grade
        print('학생 시작: {}'.format(self.name))
    def tell(self):
        SchoolMember.tell(self)
        print('학점 : {:.2f}'.format(self.grade))
```

상속 실습

In [71]:
```
t = []
t.append(Teacher('길동1', 30, 300))
t.append(Teacher('길동2', 35, 350))
t.append(Teacher('길동3', 40, 400))
s = []
s.append(Student('길산1', 21, 3.5))
s.append(Student('길사2', 22, 4.0))
s.append(Student('길사3', 23, 4.5))
print()
```

```
공통 멤버: 길동1
교수 시작: 길동1
공통 멤버: 길동2
교수 시작: 길동2
공통 멤버: 길동3
교수 시작: 길동3
공통 멤버: 길산1
학생 시작: 길산1
공통 멤버: 길사2
학생 시작: 길사2
공통 멤버: 길사3
학생 시작: 길사3
```

상속 실습

In [72]:
```
members = t + s
for member in members:
    member.tell()
```

```
이름: "길동1" 나이: 30 월급: 300
이름: "길동2" 나이: 35 월급: 350
이름: "길동3" 나이: 40 월급: 400
이름: "길산1" 나이: 21 학점 : 3.50
이름: "길사2" 나이: 22 학점 : 4.00
이름: "길사3" 나이: 23 학점 : 4.50
```

상속 실습

- 일반적인 자동차를 나타내는 클래스인 Car 클래스를 상속받아서 수퍼카를 나타내는 클래스인 SportsCar를 작성하는 것이 쉽다. 다음 그림을 참조하여 Car 클래스와 SportsCar 클래스를 작성해보자.

상속과 다형성

```
class Car :
   def __init__(self, speed):
      self.__speed = speed
   def setSpeed(self, speed):
      self.__speed = speed
   def getDesc(self):
      return "차량 =("+str(self.__speed) + ")"

class SportsCar(Car) :
   def __init__(self, speed, turbo):
      super().__init__(speed)
      self.turbo=turbo

   def setTurbo(self, turbo):
      self.turbo=turbo

obj = SportsCar(100, True)
print(obj.getDesc())
obj.setTurbo(False)
```

상속과 다형성 실습

- 일반적인 사람을 나타내는 Person 클래스를 정의한다. Person 클래스를 상속받아서 학생을 나타내는 클래스 Student와 선생님을 나타내는 클래스 Teacher를 정의한다.

```
In [117]: s = Student('길동', '12345678', '학생' )

In [118]: s.enrollCourse('자료구조')

In [119]: print(s)
```

```
타입=학생
이름=길동
주민번호=12345678
수강과목=자료구조
```

```
In [123]: t  = Teacher('길산', '0987654321', '교수')
          t.assignTeaching("Python")
          print(t)
```

```
타입=교수
이름=길산
주민번호=0987654321
강의과목=Python
```

__str__() : 객체의 문자열 표현으로 돌려준다.

객체를 print할때 str 메소드를 호출한다.

상속과 다형성

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

```
class Student(Person):
    def __init__(self, name, number, studentType ):
        Person.__init__(self, name, number)
        self.studentType = studentType

    def enrollCourse(self, course):
        self.classes = course

    def __str__(self):
        return  "\n타입="+self.studentType+ "\n이름="+self.name+ "\n주민번호="+self.number+\
            "\n수강과목="+ str(self.classes)
```

상속과 다형성

```
class Teacher(Person):
    def __init__(self, name, number, teacherType):
        super().__init__(name, number)
        self.teacherType = teacherType

    def assignTeaching(self, course):
        self.courses=course

    def __str__(self):
        return  "\n타입="+self.teacherType+ "\n이름="+self.name+ "\n주민번호="+self.number+\
            "\n강의과목="+str(self.courses)
```

```
In [117]: s = Student('길동', '12345678', '학생' )

In [118]: s.enrollCourse('자료구조')

In [119]: print(s)
```

```
타입=학생
이름=길동
주민번호=12345678
수강과목=자료구조
```

```
In [123]: t  = Teacher('길산', '0987654321', '교수')
          t.assignTeaching("Python")
          print(t)
```

```
타입=교수
이름=길산
주민번호=0987654321
강의과목=Python
```

상속과 다형성

- 메소드 오버라이딩
  - “자식 클래스의 메소드가 부모 클래스의 메소드를 오버라이드(재정의)한다”고 말한다.

상속과 다형성 예제

```
class Animal:
    def __init__(self, name=""):
        self.name=name
    def eat(self):
        print("동물이 먹고 있습니다. ")

class Dog(Animal):
    def __init__(self):
            super().__init__()
    def eat(self):
        print("강아지가 먹고 있습니다. ")

d = Dog();
d.eat()
```

강아지가 먹고 있습니다.

상속과 다형성 실습

- 직원과 매니저
  - 회사에 직원(Employee)과 매니저(Manager)가 있다. 직원은 월급만 있지만 매니저는 월급외에 보너스가 있다고 하자. Employee 클래스를 상속받아서 Manager 클래스를 작성한다. Employee 클래스의 getSalary()는 Manager 클래스에서 재정의된다.

```
In [130]: jeon = Manager('길동', 200, 100)

In [131]: print(jeon)
이름: 길동; 월급: 200; 보너스: 100
```

상속과 다형성

```
class Employee:
    def __init__(self, name, salary):
        self.name = name
        self.salary = salary

    def  getSalary(self):
        return self.salary
```

```
class Manager(Employee):
    def __init__(self, name, salary, bonus):
        super().__init__(name, salary)
        self.bonus =bonus

    def  getSalary(self):
        salary = super().getSalary()
        return salary + self.bonus

    def __str__(self):
        return "이름: "+ self.name+ "; 월급: "+ str(self.salary)+\
               "; 보너스: "+str(self.bonus)
```

상속과 다형성

- 다형성
  - **다형성(polymorphism)**은 **“많은(poly)+모양(morph)”**이라는 의미로서 주로 프로그래밍 언어에서 하나의 식별자로 다양한 타입(클래스)을 처리하는 것을 의미한다.

상속과 다형성

- 상속과 다형성

```
class Animal:
    def __init__(self, name):
        self.name = name
    def speak(self):
        return '알 수 없음'

class Dog(Animal):
    def speak(self):
        return '멍멍!'

class Cat(Animal):
    def speak(self):
        return '야옹!'

animalList = [Dog('dog1'),
              Dog('dog2'),
              Cat('cat1')]

for a in animalList:
    print (a.name + ': ' + a.speak())
```

상속과 다형성

- Lab: Vehicle와 Car, Truck
  - 일반적인 운송수단을 나타내는 Vehicle 클래스를 상속받아서 Car 클래스와 Truck 클래스를 작성해보자.

```
In [165]: cars = [Truck('truck1'), Truck('truck2'),  Car('car1')]

          for car in cars:
              print( car.name + ': ' + car.drive())
```

```
truck1: 트럭을 운전합니다.
truck2: 트럭을 운전합니다.
car1: 승용차를 운전합니다.
```

상속과 다형성

```
class Vehicle:
    def __init__(self, name):
        self.name = name

    def drive(self):
        return '운전을 합니다.'

    def stop(self):
        return '정지합니다.'
```

상속과 다형성

```
class Car(Vehicle):
    def drive(self):
        return '승용자를 운전합니다. '

    def stop(self):
        return '승용자를 정지합니다. '


class Truck(Vehicle):
    def drive(self):
        return '트럭을 운전합니다. '

    def stop(self):
        return '트럭을 정지합니다. '

cars = [Truck('truck1'), Truck('truck2'),  Car('car1')]

for car in cars:
    print( car.name + ': ' + car.drive())
```

상속

- 다중 상속 : 자식 하나가 여러 부모로 부터 상속
  - 클래스 이름을 콤마(,)로 구분하여 적어준다.

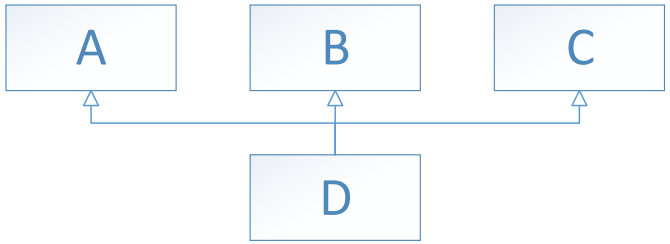
```
class A:
    pass

class B:
    pass

class C:
    pass

class D(A, B, C):
    pass
```

클래스 D는 클래스 A, B, C를 부모로부터 상속받습니다.



상속과 다형성

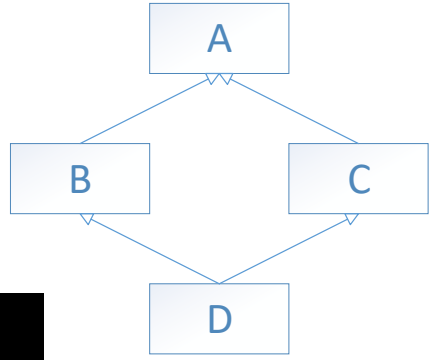
- 다이아몬드 상속 : 다중 상속이 만들어 내는 곤란한 상황
  - D는 B와 C 중 누구의 method()를 물려받게 되는 걸까?

```
class A:
    def method(self):
        print("A")

class B(A):
    def method(self):
        print("B")

class C(A):
    def method(self):
        print("C")

class D(B, C):
    pass
```




```
>>> class A:
    def method(self):
        print("A")

>>> class B(A):
    def method(self):
        print("B")

>>> class C(A):
    def method(self):
        print("C")

>>> class D(B, C):
    pass

>>> obj = D()
>>> obj.method()
B
```

D는 B의 method()를 물려받았습니다.

- __call__(self) 메소드
  - 객체를 함수 호출 방식으로 사용하게 만드는 메소드

```
In [124]: class Callable:
              def __call__(self):
                  print('I am called.')

In [125]: obj = Callable()

In [126]: obj
Out[126]: <__main__.Callable at 0x21c61c74b38>

In [127]: obj()
          I am called.
```

인스턴스 뒤에 괄호 (와)를 붙여 “호출”하면, 내부적으로는 __call__ 메소드가 호출됩니다.

- isinstance() 함수
  - 객체가 어떤 클래스로부터 만들어졌는지 확인

```
isinstance(인스턴스, 클래스)
```

```
# 클래스를 선언합니다.
class Student:
    def __init__(self):
        pass

# 학생을 선언합니다.
student = Student()

# 인스턴스 확인하기
print("isinstance(student, Student):", isinstance(student, Student))
```

```
isinstance(students[0], Student): True
```

- isinstance() 함수의 다양한 활용
  - 예시 – 리스트 내부에 여러 종류의 인스턴스 들어있을 때, 인스턴스들을 구분하며 속성과 기능 사용

```
01  # 학생 클래스를 선언합니다.
02  class Student:
03      def study(self):
04          print("공부를 합니다.")
05
06  # 선생님 클래스를 선언합니다.
07  class Teacher:
08      def teach(self):
09          print("학생을 가르칩니다.")
10
11  # 교실 내부의 객체 리스트를 생성합니다.
12  classroom = [Student(), Student(), Teacher(), Student(), Student()]
13
14  # 반복을 적용해서 적절한 함수를 호출하게 합니다.
15  for person in classroom:
16      if isinstance(person, Student):
17          person.study()
18      elif isinstance(person, Teacher):
19          person.teach()
```

실행결과

```
공부를 합니다.
공부를 합니다.
학생을 가르칩니다.
공부를 합니다.
공부를 합니다.
```

Property

- Property 사용하기
  - 값을 가져오는 getter
  - 값을 저장하는 setter

```
class Person:
    def __init__(self):
        self.__age = 0

    def get_age(self):              #getter
        return self.__age

    def set_age(self, value):       #setter
        self.__age = value
```

```
In [8]: jeon = Person()

In [9]: jeon.set_age(35)

In [10]: print(jeon.get_age())
35
```

Property

- Property 사용하기
  - @property : 값을 가져오는 메소드에 붙인다.
  - @메소드이름.setter : 값을 저장하는 메소드에 붙인다.

```
class Person:
    def __init__(self):
        self.__age = 0

    @property
    def age(self):              #getter
        return self.__age

    @age.setter
    def age(self, value):       #setter
        self.__age = value
```

```
In [4]: jeon = Person()

In [5]: jeon.age = 35

In [6]: print(jeon.age)
35
```

클래스 관계

- 클래스 관계
  - is – a 관계 : 상속 관계
    - ✓ 명확하게 같은 종류, 동등한 관계일 때
    - ✓ '학생은 사람이다.'라고 했을 때 말이 되면 동등한 관계
    - ✓ Student is a Person
  - has – a 관계 : 포함 관계
    - ✓ 사람 목록을 관리하는 클래스 만든다면, 리스트 속성에 Person 객체를 넣어서 관리
    - ✓ 같은 종류에 동등한 관계일 때는 상속, 그 이외에는 속성에 인스턴스를 포함

```
In [12]: class Person:
             pass

         class Student(Person):
             pass
```

모듈과 패키지 가져오기

- 모듈과 패키지 사용하기
  - 모듈(module)은 함수, 변수, 클래스를 담고 있는 파일
  - 패키지(package) 여러 모듈을 묶은 것

- import 로 모듈 가져오기
  - 여러 개 모듈을 가져올 때는 콤마(,)로 구분
    - ✓ import 모듈
    - ✓ import 모듈1, 모듈2
    - ✓ 모듈.변수
    - ✓ 모듈.함수()

```
In [16]: import math
In [17]: math.pi
Out[17]: 3.141592653589793
```

```
In [18]: math.sqrt(4.0)
Out[18]: 2.0
```

모듈과 패키지 가져오기

- import as 로 모듈 이름 지정하기
  - 앞선 예제에서 math를 입력하고 싶지 않을 때
  - import 모듈 as 이름

```
In [20]: import math as m

In [21]: m.pi
Out[21]: 3.141592653589793
```

```
In [22]: m.sqrt(2)
Out[22]: 1.4142135623730951
```

모듈과 패키지 가져오기

- from import 로 모듈 일부만 가져오기
  - from 모듈 import 변수 (or 함수, or 클래스)

```
In [23]: from math import pi

In [24]: pi
Out[24]: 3.141592653589793
```

```
In [25]: from math import sqrt

In [26]: sqrt(2)
Out[26]: 1.4142135623730951
```

  - math 모듈에서 가져올 변수와 함수가 여러 개 일 경우
    - ✓ import 뒤에 가져올 변수, 함수, 클래스를 콤마로 구분
    - ✓ from 모듈 import 변수, 함수, 클래스

```
In [27]: from math import pi, sqrt

In [28]: pi
Out[28]: 3.141592653589793

In [29]: sqrt(4)
Out[29]: 2.0
```

모듈과 패키지 가져오기

- from import 로 모듈 일부만 가져오기
  - 모든 변수, 함수, 클래스를 가져올 경우
  - from 모듈 import *

```
In [30]: from math import *

In [31]: pi
Out[31]: 3.141592653589793
```

- 모듈 일부 가져오면서 이름 지정하기
  - from 모듈 import 변수 as 이름

```
In [32]: from math import sqrt as s

In [33]: s(4)
Out[33]: 2.0
```

모듈과 패키지 가져오기

- 모듈 일부 가져오면서 이름 지정하기
  - 여러 개 가져와 이름 지정할 경우
  - from 모듈 import 변수 as 이름1, 함수 as 이름2, 클래스 as 이름3

```
In [36]: from math import pi as p, sqrt as s

In [37]: p
Out[37]: 3.141592653589793

In [38]: s(2)
Out[38]: 1.4142135623730951
```

모듈과 패키지 가져오기

- import 로 패키지 가져오기
  - 패키지는 특정 기능과 관련된 여러 모듈을 묶은 것.
  - 패키지 안에 들어있는 모듈도 import 를 사용하여 가져옴
    - ✓ import 패키지.모듈
    - ✓ import 패키지.모듈1, 패키지.모듈2
    - ✓ 패키지.모듈.변수
    - ✓ 패키지.모

```
In [39]: import urllib.request

In [44]: response = urllib.request.urlopen('http://www.google.co.kr')
         response.status

Out[44]: 200
```

모듈과 패키지 가져오기

- import as로 패키지 모듈 이름 지정하기
  - import 패키지.모듈 as 이름

```
In [45]: import urllib.request as r

In [46]: response = r.urlopen('http://www.google.co.kr')
         response.status

Out[46]: 200
```

- from import
  - from 패키지.모듈 import 변수

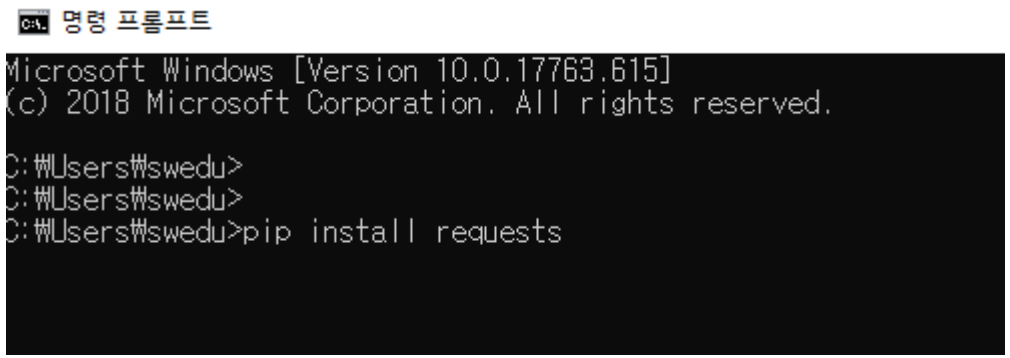
- 패키지 모듈에서 모든 변수, 함수, 클래스 가져오기
  - from 패키지.모듈 import *

모모듈과 패키지 가져오기듈

- 패키지 모듈의 일부를 가져와서 이름 지정
  - from 패키지.모듈 import 변수 as 이름
  - from 패키지.모듈 import 변수 as 이름1, 함수 as 이름2, 클래스 as 이름3

패키지 설치하기

- 파이썬 패키지 인덱스에서 패키지 설치하기
  - PyPI (Python Package Index) 통해 다양한 패키지를 설치할 수 있다.
- pip 설치 하기
  - pip는 파이썬 패키지 인덱스의 패키지 관리 명령어
  - 윈도우용 파이썬에는 기본 내장
  - 사용방법
    - ✓ pip install 패키지
    - ✓ 윈도우키 + R , cmd입력



패키지 설치하기

- pip 설치 하기
  - -m 옵션을 지정해서 pip 실행
  - -m 옵션은 모듈을 실행하는 옵션이며 pip 도 모듈

```
C:₩Users₩swedu>python -m pip install requests
```

모듈과 패키지 만들기

- 모듈과 패키지 만들기
  - 2의 거듭제곱 모듈 만들기
  - Ai 폴더(c:\Ai) 안에 square2.py 파일로 저장

square2.py - C:/Ai/square2.py (3.7.3)

File Edit Format Run Options Window Help

```
base = 2

def square(n):
    return base ** n
```

  - Ai 폴더(c:\Ai) 안에 main.py 파일로 다음을 저장

main.py - C:/Ai/main.py (3.7.3)

File Edit Format Run Options Window Help

```
import square2

print(square2.base)
print(square2.square(3))
```

```
2
8
>>>
```

모듈과 패키지 만들기

main.py - C:/Ai/main.py (3.7.3)

File Edit Format Run Options Window Help

```
from square2 import base, square

print(base)
print(square(3))
```

- 모듈에 클래스 작성하기
  - Ai 폴더(c:\Ai) 안에 person.py 파일로 저장

person.py - C:₩Ai₩person.py (3.7.3)

File Edit Format Run Options Window Help

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

    def greet(self):
        print('안녕하세요.', self.name)
```

모듈과 패키지 만들기

- main 파일 수정

main.py - C:/Ai/main.py (3.7.3)

File Edit Format Run Options Window Help

```
import person

jeon = person.Person('길동', 35)
jeon.greet()
```

```
====================
안녕하세요. 길동
>>>
```

모듈 시작점

- 모듈 시작점 알아보기
  - 다음 코드를 작성하여 Ai 폴더에 저장(파일명 hello.py)

hello.py - C:/Ai/hello.py (3.7.3)

File Edit Format Run Options Window Help

```
print('hello 모듈 시작')
print('hello.py __name__:' __name__)

print('hello 모듈 끝')
```

  - 다음 코드를 작성하여 Ai 폴더에 저장(파일명 main.py)

main.py - C:/Ai/main.py (3.7.3)

File Edit Format Run Options Window Help

```
import hello

print('main.py __name__ : ', __name__)
```

모듈 시작점

- hello 모듈을 import 하면 hello 모듈의 내용이 실행
  - __name__에 모듈이 이름이 출력

```
============================== RE
hello 모듈 시작
hello.py __name__: hello
hello 모듈 끝
main.py __name__ :  __main__
>>>
```

모듈 시작점

- 파이썬 인터프리터가 최초로 실행한 스크립트 파일의 __name__ 에는 __main__ 이 들어간다.(프로그램의 시작점 의미)

```
C:₩Ai>python main.py
hello 모듈 시작
hello.py __name__: hello
hello 모듈 끝
main.py __name__ :  __main__

C:₩Ai>python phello.py
python: can't open file 'phello.py': [Errno 2] N

C:₩Ai>python hello.py
hello 모듈 시작
hello.py __name__: __main__
hello 모듈 끝

C:₩Ai>
```

모듈 시작점

- 모듈과 시작점
  - 최초 시작 스크립트 파일과 모듈의 차이가 없음
  - 스크립트 파일이 시작점도 될 수 있고, 모듈도 될 수 있다.
  - __name__ 변수를 통해 현재 스크립트 파일이 시작점인지 모듈인지 판단
  - if __name__ == '__main__' :

    __name__ 변수 값이 '__main__' 인지 확인하는 코드는 현재 파일이 프로그램의 시작점이 맞는지 판단

    파일이 메인 프로그램으로 사용될 때와 모듈로 사용될 때를 구분하기 위한 용도로 사용

모듈 시작점

calc.py - C:/Ai/calc.py (3.7.3)

File Edit Format Run Options Window Help

```
def add_(x, y):
    return x + y

def mul_(x, y):
    return x * y

if __name__ == '__main__':
    print(add_(1, 2))
    print(mul_(1, 2))
```

```
3
2
>>>
```

모듈 시작점

- calc.py를 모듈로 사용하면

```
>>> import calc
>>>
>>>
```

  - 아무것도 출력이 나오지 않는다.

    __name__ 변수 값이 '__main__' 아니기 때문

  - 파일을 모듈로 사용할 경우 calc.add 또는 calc.mul 함수만 사용하는 것이 목적이 되므로 1, 2의 합과 곱을 출력하는 코드는 필요 없음

```
>>> calc.add_(3, 4)
7
>>> calc.mul_(3, 4)
12
>>>
```

패키지 만들기

- 패키지 만들기
  - 모듈은 파일이 한 개지만, 패키지는 폴더로 구성
  - C:\Ai 폴더 밑에
  - main.py 파일
  - calpkg 폴더 (폴더 안에 __init__.py, operation.py, geometry.py) 로 구성

패키지 만들기

- 패키지 만들기
  - C:\Ai 폴더 안에 calcpkg 폴더 만든다.
  - calcpkg 폴더 안에 __init__.py 파일 생성
  - calcpkg 폴더 안에 operation.py 파일 생성

operation.py - C:/Ai/calcpkg/operation.py (3.7.3)

File Edit Format Run Options Window Help

```
def add_(x, y):
    return x + y

def mul_(x, y):
    return x * y
```

geometry.py - C:/Ai/calcpkg/geometry.py (3.7.3)

File Edit Format Run Options Window Help

```
def t_area(x, y):
    return x * y / 2

def r_area(x, y):
    return x * y
```

  - calcpkg 폴더 안에 geometry.py 파일 생성

패키지 만들기

- C:\Ai 폴더 안에 main.py 생성

main.py - C:/Ai/main.py (3.7.3)

File Edit Format Run Options Window Help

```
import calcpkg.operation
import calcpkg.geometry

print(calcpkg.operation.add_(1, 2))
print(calcpkg.operation.mul_(1, 2))

print(calcpkg.geometry.t_area(3, 4))
print(calcpkg.geometry.r_area(3, 4))
```

```
3
2
6.0
12
>>>
```