1. 모듈

- 모듈 : 외부 라이브러리 ex) requests(HTTP요청), numpy(수치분석), pandas(데이터분석)

[모듈 전체 불러오기]

import

- import math - .pi / .sin(x) / .cos(x) 등
- import functools
- import numpy
- import pandas
- import requests

[모듈에서 특정 함수만 불러오기]

from 모듈명 import 함수명(변수, 클래스 등도 가능)

이 경우, 특정함수 사용 위해서 모듈명.함수 필요X. 함수 명만 써도 기능 math.pi 할 필요 없이 pi만 써도 기능함
math.cos(x) 할 필요 없이 cos(x) 가능

[모듈의 별명짓기]

import math as m

math.pi 가 아닌

m.pi 가능

2. class 생성, 생성자

class 라는 타입 > 속성(멤버변수), 행동(생성자 및 메소드) > 객체(인스턴스)

```
# class 정의 및 속성(멤버변수) 초기화(생성자 사용)
class Human:
    def __init__(self,name,game):

        self.a = name
        self.b = game

# 인스턴스 설정
p1 = Human('이슬','pretty')

# 인스턴스 수행
print(p1.a, p1.b)
```

class 생성 시에는, 항상 그 class의 속성(멤버변수)를 만들기 위해 생성자(init메소드)를 사용!

생성자('**init**' 메소드) 는 멤버변수 생성의 기능!

```
class person():
    def __init__(self,멤버변수1 인자,멤버변수2 인자,멤버변수3 인자):
        self.멤버변수1 = 멤버변수1 인자
        self.멤버변수2 = 멤버변수2 인자
        self.멤버변수3 = 멤버변수3 인자

#이렇게 하면, 멤버변수 주머니가 생기게 되는 것.
#이렇게 해야, 인스턴스 생성 ->  p1 = person(인자1, 인자2, 인자3)    ex)  p1 = perso
n('kathy',23,'서울')
#이렇게 해야, 멤버변수 값 출력 -> p1.인자1 / p1.인자2 등이 가능    ex)  p1.name  ->
 kathy  출력
```

In [13]:

```
# '이름 : kathy, 나이 : 19살' 을 class를 활용해 출력해보라
# class 생성 후 생성자(최초의 method) 를 통해 --> 멤버변수 생성
# person이라는 클래스 생성 -> 속성과 메소드 생성 -> 메소드(생성자)로 속성(이름, 나이 / 멤버변수) 설정 -> 인스턴스 생성 -> 출력에 활용


#클레스 생성, 클래스의 속성과 메소드 설정   ->   속성 : 이름, 나이 / 메소드 : 생성자
class person:
    def __init__(self,namegiven,agegiven):
        self.name = namegiven
        self.age = agegiven

#속성의 구체적 제시
namegiven = 'kathy'
agegiven = 19

#클래스에 특정 속성을 가진 인스턴스 생성
p1 = person(namegiven,agegiven)

#출력에 활용
a = '내 이름은 {}. 나이는 {}살이야'.format(p1.name,p1.age)
a
```

Out[13]:

```
'내 이름은 kathy. 나이는 19살이야'
```

In [19]:

```
# '이름 : kathy, 나이 : 19살' 을 class를 활용해 출력해보라
# class 생성 + 메소드 추가(introduce)를 통해서 한번에 출력되도록 만들어보라

class person:
    def __init__(self,givenname,givenage):
        self.name = givenname
        self.age = givenage

    def sleep(self):
        print('안녕! 내 이름은 {}. 나이는 {}살이야.'.format(self.name,self.age))

givenname = 'kathy'
givenage = 19

p1 = person(givenname,givenage)

p1.sleep()
```

```
안녕! 내 이름은 kathy. 나이는 19살이야.
```


참고1. self의 의미 = 자기자신 지칭... 멤버변수와 메소드 생성시 어떤 의미인가?

- 파이썬 method는 항상 첫번째 인자로 self를 전달

```
# class 정의 및 속성(멤버변수) 초기화(생성자 사용)
class Human:
    def __init__(self,name,game):

        self.a = name
        self.b = game

# 인스턴스 설정
p1 = Human('이슬','pretty')
->  p1 = Humna(p1,'이슬','pretty') 인 것. but, init생성자에선 이를 self로 굳이 안 넣어도 알아서 self를 넣어주는 것

# 인스턴스 수행
print(p1.a)
```

- self는 현재 해당 method가 호출되는 객체 자신을 가리킴
- 역시, 이름이 self일 필요는 없으나, 위치는 항상 맨 처음의 parameter이며, 관례적으로 self를 사용

참고2. self의 의미

In [21]:

```
#### 객체로 자기자신을 지정하고 싶을 때 self를 사용함

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

    def sleep(self):
        print(self.name, '은 잠을 잡니다.')

a = Person('Aaron',20)

#### self는 자기자신. a = Person(a,'Aaron',20)과 같은 것임 사실은. 제일 처음에 자기자신을 부르는...

b = Person('bob',30)

print(a)
print(b)

#### 아래에 출력되는 값은 a와 b라는 객체(인스턴스)가 상주하고 있는 주소를 의미함  ex) 0x103db7ad0

a.sleep()

#### self는 자기자신. a.sleep() 은 a.sleep(a)가 되는 것.
#### print(self.name, ~~ )되었지만, self자리엔 역시나 a가 들어감. 즉, print(a.name, ~~ ) 가 되는 것.

b.sleep()
```

```
self:  <__main__.Person object at 0x10766f690>
self:  <__main__.Person object at 0x10766ff90>
<__main__.Person object at 0x10766f690>
<__main__.Person object at 0x10766ff90>
Aaron 은 잠을 잡니다.
bob 은 잠을 잡니다.
```

3. 클래스 - > 메소드 생성

method 정의

- 멤버변수라고도 하며, 해당 클래스의 object에서만 호출가능
- 메쏘드는 객체 레벨에서 호출되며, 해당 객체의 속성에 대한 연산 수행

In [22]:

```
#1. 숫자를 하나 증가
#2. 숫자를 0으로 초기화

#예시) increment -> +1 / minus -> -1/ reset -> 0으로 / print_current_value -> '현재값은' ~ 출력

class Counter:
    def __init__(self):
        self.num = 0

    def increment(self):     #-> c1.increment()로 받을 것이고, 이는 사실 c1.increment(c1)을 의미
        self.num += 1

    def reset(self):
        self.num = 0

    def print_current_value(self):
        print('현재값은:',self.num)


c1 = Counter()
# 실은 c1 = Counter(c1)으로 적용되는 것. self로 인해서
c1.print_current_value()

c1.increment()
c1.increment()
c1.increment()

c1.print_current_value()

c1.reset()

c1.print_current_value()
```

```
현재값은: 0
현재값은: 3
현재값은: 0
```

In [26]:

```
#예시) mulply -> *2 / minus -> -1/ reset -> 0으로 / print_current_value -> '현재값은'
~ 출력

#아래의 메소드를 생성 후, 수행하라.

# b1.mulply()
# b1.minus()
# b1.print_current_value()
# b1.reset()

class cal:
    def __init__(self,number):
        self.num = number

    def mulply(self):
        self.num = self.num*2

    def minus(self):
        self.num -= 1

    def print_current_value(self):
        print(self.num)

    def reset(self):
        self.num = 0

b1 = cal(10)

b1.mulply()
b1.mulply()
b1.mulply()
b1.minus()
b1.print_current_value()
b1.minus()
b1.print_current_value()
b1.reset()
b1.print_current_value()
```

```
79
78
0
```

-.- 정리

instance method / class method (utility 적 기능 method) 차이 확인

- p1.name - > 인스턴스.멤버변수 (이름이 출력됨)
- p1.sleep() - > 인스턴스.메소드
- b1.mulply() - > 인스턴스.메소드
- math.cos() - > 클래스.메소드 (멤버변수가 없고, 그냥 기능만 하는 메소드... 즉, 위에는 인스턴스.메소드 인데, 이건 클래스.메소드)

class method는 내부적으로 유지할 성질은 없고, 그냥 주어진 값을 연산처리하면 되는 utility method

class method 구현 방법 -> @staticmethod 데코레이터 + self 제거

```
class math:
    def add(x,y):
        print x+y

math.add(1,2)
```

In [31]:

```
class math:
    @staticmethod
    def add(x,y):
        return x+y

# class method
math.add(1,2)
```

Out[31]:

```
3
```

In [37]:

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

    def increment(self):
        self.age += 1

a12 = person('kim',12)

# instance method
a12.increment()
a12.age
```

Out[37]:

```
13
```

4. Class 상속

Class Inheritance (상속)

함수와 다른 의미의 코드를 재사용할 수 있게 해주는 도구

- 기존에 정의해둔 클래스의 기능을 그대로 물려받을 수 있다.
- 기존 클래스에 기능 일부를 추가하거나, 변경하여 새로운 클래스를 정의한다.
- 코드를 재사용할 수 있게된다.
- **상속 받고자 하는 대상인 기존 클래스는 (Parent, Super, Base class 라고 부른다.)**
- **상속 받는 새로운 클래스는(Child, Sub, Derived class 라고 부른다.)**
- 의미적으로 is-a관계를 갖는다

[is - a 관계?]

```
class person:   - > parent
class employee:    - > child
class student:     - > child

'''
employee is a person
student is a person
child is a parent
'''

class person():
    def __init__(self,name,age):
        self.name = name
        self.age = age
    def work(self,minute):
        print('{}는 {}분동안 일합니다'.format(self.name,minute)

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

여기서 override하려면?

여기서 override후 상위 클래스 기능(메소드) 그대로 사용하려면?

In [1]:

```
#상속하려면 ()로 parent를 넣고, 생성자만 넣어주면 동일하게 속성, 기능 상속됨
#           class student(person):
#               def __init__( ~~ ):

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

    def eat(self, food):
        print('{}은 {}를 먹습니다.'.format(self.name, food))

    def sleep(self, minute):
        print('{}은 {}분동안 잡니다.'.format(self.name, minute))

    def work(self, minute):
        print('{}은 {}분동안 일합니다.'.format(self.name, minute))

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

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

bob = Employee('Bob', 25)
bob.eat('BBQ')
bob.sleep(30)
bob.work(60)
```

```
Bob은 BBQ를 먹습니다.
Bob은 30분동안 잡니다.
Bob은 60분동안 일합니다.
```

method override

- 상속하더라도, **부모 클래스의 method를 재정의(override)**
- 하위 클래스(자식 클래스) 의 인스턴스로 호출시, 재정의된 메소드가 호출됨

In [2]:

```
#학생 class에서 work의 메소드를 재정의한다.
#그냥 그 class에서 method를 새로 정의해주면 됨
#같은 method명이라도, instance.method 하면 재정의된 method로 나타날 것! (대신, instance가 학생class 소속이어야 그렇게 되겠지?)

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

    def eat(self, food):
        print('{}은 {}를 먹습니다.'.format(self.name, food))

    def sleep(self, minute):
        print('{}은 {}분동안 잡니다.'.format(self.name, minute))

    def work(self, minute):
        print('{}은 {}분동안 일합니다.'.format(self.name, minute))

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

    def work(self, minute):
        print('{}은 {}분동안 공부합니다.'.format(self.name, minute))

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

    def work(self, minute):
        print('{}은 {}분동안 업무를 합니다.'.format(self.name, minute))

bob = Employee('Bob', 25)
bob.eat('BBQ')
bob.sleep(30)
bob.work(60)
```

```
Bob은 BBQ를 먹습니다.
Bob은 30분동안 잡니다.
Bob은 60분동안 업무를 합니다.
```

super

재정의 하면서 부모 클래스의 상속도 그대로 활용하고 싶을 때

```
    def work(self,minute):
        super().work(minute)        # -> 이것처럼 하면, 부모 클래스의 work 메소드도 그대로 수행됨
        print('{}은 {}분 동안 업무를 합니다.'.format(self.name, minute)
```

- 하위클래스(자식 클래스)에서 부모클래스의 method를 호출할 때 사용

In [3]:

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

    def work(self, minute):
        print('{}은 {}분동안 준비를 합니다.'.format(self.name, minute))

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

    def work(self, minute):
        super().work(minute)
        print('{}은 {}분동안 공부합니다.'.format(self.name, minute))

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

    def work(self, minute):
        super().work(minute)                        # super().메소드(속성)  -> super클래스의 메소드가 수행됨.
        print('{}은 {}분동안 업무를 합니다.'.format(self.name, minute))

bob = Employee('Bob', 25)
bob.work(60)
```

```
Bob은 60분동안 준비를 합니다.
Bob은 60분동안 업무를 합니다.
```

5. special method

- **로 시작** 로 끝나는 특수 함수
- 해당 메쏘드들을 구현하면, 커스텀 객체(p1, p2 같은..)에 여러가지 파이썬 내장 함수나 연산자를 적용 가능 ex) p3 = p1+p2
- 오버라이딩 가능한 함수 목록은 아래 링크에서 참조
  - https://docs.python.org/3/reference/datamodel.html (https://docs.python.org/3/reference/datamodel.html)

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

def __str__(self):
    return '({},{})'.format(self.x,self.y)

'이거 하고 __add__ 등 써나가야 커스텀 객체(p1)를 그대로 연산자, 내장함수에 사용 가능'
```

In [5]:

```
# Point
# 2차원 좌표평면 각 점(x, y)
# 연산
# 두점 의 덧셈, 뺄셈 (1, 2) + (3, 4) = (4, 6)
# 한점과 숫자의 곱셈 (1,  2) * 3 = (3, 6)
# 그 점의 길이 (0,0) 부터의 거리
# x, y 값 가져오기
# 출력하기

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

    #point의 인스턴스들은 1+2 처럼 다룰 수 있게 만들어보겠다 선언!
    def __str__(self):
        return '({},{})'.format(self.x,self.y)

    def __add__(self,pt):
        new_x = self.x+pt.x
        new_y = self.y+pt.y
        return point(new_x,new_y)

    def __sub__(self,pt):
        new_x = self.x - pt.x
        new_y = self.y - pt.y
        return point(new_y,new_y)

    def __mul__(self,factor):
        return point(self.x*factor,self.y*factor)

    def __getitem__(self,index):
        if index == 0:
            return self.x
        elif index == 1:
            return self.y
        else:
            return -1

    def __len__(self):
        return self.x**2 + self.y**2


p1 = point(1,2)
p2 = point(2,3)
p3 = p1+p2
p4 = p1-p2
print(p3)
print(p4)
p5 = p1 * 2
print(p5)
p1[0]
p1[1]
print(len(p1))
```



```
(3,5)
(-1,-1)
(2,4)
5
```

연습문제)

- 복소수 클래스를 정의 해봅시다.
- 덧셈, 뺄셈, 곱셈 연산자 지원
- 길이 (복소수의 크기) 지원
- 복소수 출력 '1 + 4j'와 같이 표현
- 비교 연산 ==, != 지원
- =, <= , <, > 연산 지원
- 절대값 지원

In [ ]: