6. 클래스와모듈

1. 클래스

기본 생성자 :

- __init__(self) : => 생성자를 구현하지 않으면 자동으로 제공되는 생성자
- pass

self : 자기참조변수:자신객체의 멤버를 표현시 사용되는 예약어

```
class Car :   #클래스
   color = ""  # 멤버변수
   speed = 0  #  멤버변수
   def upSpeed(self, value):  # 멤버메서드
      self.speed += value
   def downSpeed(self, value):  # 멤버메서드
      self.speed -= value
```

1. 클래스

```
myCar1 = Car()  # 객체화. 생성자 호출됨. 기본생성자 호출
myCar1.color = "빨강"
myCar1.speed = 0
myCar2 = Car()
myCar2.color = "노랑"
myCar2.speed = 0
myCar3 = Car()
myCar3.color = "파랑"
myCar3.speed = 0
myCar1.upSpeed(30)
print("자동차1의 색상은 %s 이며, 현재 속도는 %dkm 입니다." %  (myCar1.color, myCar1.speed))
```

2. 생성자

__init__ 생성자임

```
class Car :
    color = ""  # 초기화
    speed = 0
    #생성자.
    def __init__(self, v1, v2):  # self :매개변수 목록의 첫번째여야 함
        self.color = v1
        self.speed = v2
    def upSpeed(self, value):
        self.speed += value
    def downSpeed(self, value):
        self.speed -= value

myCar1 = Car("빨강", 10)  # v1<=빨강, v2<=10  값으로 매핑함
myCar2 = Car("노랑", 20)  # v1<=노랑, v2<=20 값으로 매칭함
print("자동차1의 색상은 %s 이며, 현재 속도는 %dkm 입니다." %  (myCar1.color, myCar1.speed))
print("자동차2의 색상은 %s 이며, 현재 속도는 %dkm 입니다." %  (myCar2.color, myCar2.speed))
```

3. 클래스 멤버,인스턴스멤버

```
class Car :
    color = ""
    speed = 0
    num = 0
    count = 0
    def __init__(self,v=""):
        self.color = v
        self.speed = 0  #인스턴스 변수
        Car.count += 1  #클래스 변수 : 클래스명.변수명
        self.num = Car.count #인스턴스 변수

    def printMessage(self):
        print("색상:%s, 속도:%dkm, 번호:%d, 생산번호:%s" %
          (self.color,self.speed,self.num,Car.count),end="")
```

3. 클래스 멤버,인스턴스멤버

```
mycar1,mycar2 = None, None #null, 참조 객체 없음.
mycar1 = Car()  #객체화. num=1
mycar1.speed = 30
mycar1.printMessage()
print()
mycar2 = Car()  #객체화 num=2
mycar2.speed = 50
mycar2.printMessage()
print()
#Car.count += 10
print("생산번호:%d" % (mycar1.count))
print("생산번호:%d" % (mycar2.count))
mycar3 = Car("빨강")  #객체화. 오류 발생. 생성자 없음.
mycar3.printMessage()
```

4. 상속, 오버라이딩

파이썬에서 다중 상속 가능

```
class Car :
    speed = 0
    door = 3
    def upSpeed(self,value):
        self.speed += value
        print("현재 속도(부모클래스): %d" % self.speed)

class Sedan(Car):  # Car클래스의 하위 클래스
    pass          # Car 클래스의 멤버와 같다.
```

4. 상속, 오버라이딩

```
class Truck(Car):
    def upSpeed(self,value):  #150이상의 speed인 경우는 최대 150. 오버라이딩
        self.speed += value
        if self.speed > 150 :
            self.speed = 150
        print("현재 속도(자손클래스): %d" % self.speed)

class Container :
    room = 1

class MovingCar(Container, Car) :  #다중 상속: 부모가 여럿인 상속
    pass
```

4. 상속, 오버라이딩

```
sedan1 = Sedan()  #객체화
truck1 = Truck()  #객체화
print("트럭 :",end="")
truck1.upSpeed(200) #Truck의 upSpeed 메서드가 오버라이딩됨
print("승용차 :",end="")
sedan1.upSpeed(200)

mcar = MovingCar() #Container,Car 클래스를 상속받음.
mcar.upSpeed(60)
print("이동차량의 방갯수:",mcar.room,", 문의 갯수:",mcar.door)
```

5.클래스에서 사용되는 특별한 메서드

생성자 : __init__(self,...)

소멸자 : __del__(self)

문자열화 : __repr__(self)

\+ : __add__(self,other)

\- : __sub__(self,other)

< : __lt__(self,other)

> : __gt__(self,other)

== : __eq__(self,other)

소멸자:__del__(self)

5.클래스에서 사용되는 특별한 메서드

```
class Line :
    length = 0
    def __init__(self,length):
        self.length = length
    def __repr__(self) :
        return "선의 길이:" + str(self.length)
    def __add__(self,other) : # + 연산자 사용시 호출되는 메서드
        print("+ 연산자 호출")
        return self.length + other.length
    def __lt__(self,other):    # myline1 < myline2 구문실행시 호출
        print("< 연산자 호출")
        return self.length < other.length
    def __gt__(self,other):    # myline1 > myline2 구문실행시 호출
        print("> 연산자 호출")
        return self.length > other.length
    def __eq__(self,other):    # myline1 == myline2 구문실행시 호출
        print("== 연산자 호출")
        return self.length == other.length
    def __del__(self):
        print(self.length,"길이 선이 제거 되었습니다.")
```

5.클래스에서 사용되는 특별한 메서드

```
myline1 = Line(200) # __init__ 호출. 생성자 :객체생성시 호출되는 메서드
myline2 = Line(100)
print(myline1)     #__repr__(self) :
print(myline2)
print("두선의 길이의 합:",myline1+myline2) #__add__ 호출
print("두선의 길이의 합:",myline1.__add__(myline2)) #__add__ 호출
if myline1 < myline2 :    # __lt__ 호출
   print("myline2 선이 더 깁니다.")
elif myline1 == myline2 :       # __eq__ 호출
   print("myline1과 mylise2선의 길이는 같습니다.")
elif myline1 > myline2 :
   print("myline1 선이 더 깁니다.")
if myline1.__lt__(myline2) :    # __lt__ 호출
   print("myline2 선이 더 깁니다.")
elif myline1.__eq__(myline2) :       # __eq__ 호출
   print("myline1과 mylise2선의 길이는 같습니다.")
elif myline1.__gt__(myline2) :
   print("myline1 선이 더 깁니다.")
print("프로그램 종료") #생성된 모든 객체들이 제거됨. 소멸자 호출됨.
```

6. 추상메서드

부모클래스의 멤버 메서드를 자손 클래스에서 구현함

```
class SuperClass :
    # raise NotImplementedError : 오버라이딩안하면 예외 발생.
    def method(self):   #추상메서드
        raise NotImplementedError  #  반드시 오버라이딩 해야 함
# SubClass1 클래스 생성하기. SuperClass의 하위클래스. 부모와 같은 멤버를 가진다.
class SubClass1(SuperClass) :
#    pass   #오류발생
    def method(self) :
        print("SubClass1에서 method 함수를 오버라이딩 함.")

sub1 = SubClass1()
sub1.method()
```

7. 모듈구현하기

mod.py

```
def add(a, b):
    return a + b

def sub(a, b):
    return a-b
```

- mod2.py

```
def add(a, b):
    return a + b

def sub(a, b):
    return a-b
'''
__name__ : mod2.py 파일을 실행할 경우 "__main__" 값이 저장됨.
'''
if __name__ =="__main__":  #mod2.py 파일이 직접실행시
    print(add(3, 4))
    print(sub(4, 2))
```

8. 모듈 사용하기

import : 모듈 가져오기

```
import mod #모듈 가져오기.
import mod2

print("mod 모듈",mod.add(3, 4))
print("mod 모듈",mod.sub(4, 2))

print("mod2 모듈",mod2.add(3, 4))
print("mod2 모듈",mod2.sub(4, 2))
```

8. 모듈 사용하기

from ... import 사용하기

```
#mod 모듈에서 add(함수,객체) 만 import함. sub 함수 사용 불
from mod import add

print(add(3, 4))
#print(sub(4, 2))
```

8. 모듈 사용하기

mod 모듈에서 add,sub 함수를 가져오기.

```
from mod import add, sub
print(add(3, 4))
print(sub(4, 2))
```

9.모듈 목록 출력하기

```
import math
import sys

print(sys.builtin_module_names)  # 설정 모듈 목록 리턴
print(dir(__builtins__))
print(dir(math))
```

10.정규식

정규화 모듈 사용 안함

```
data = '''
   park 800905-1234567
   kim  700905-1234567
   choi 850101-a123456
'''
result = [] # park 800905-*******
#line : park 800905-1234567
for line in data.split("\n") :
    word_result = [] # park 800905-*******
    # word : 800905-1234567
    # isdigit : 숫자인경우 true
    for word in line.split(" ") :
        if len(word) == 14 and word[:6].isdigit() and word[7:].isdigit():
            word = word[:6] + "-" + "*******"
        word_result.append(word)
# join : list의 요소를 연결
    result.append(" ".join(word_result))
print("\n".join(result))
```

10.정규식

정규화 모듈 사용

```
import re  #정규식을위한 모듈

data = '''
   park 80090-1234567
   kim  700905-1234567
   choi 850101-a123456
'''
pat = re.compile("(\d{6,7})[-]\d{7}")
print(pat.sub("\g<1>-*******", data))
```

10.정규식

정규식에서 사용되는 기호

1. () : 그룹화
2. [] : 문자
3. {n} : n개 갯수
   ca{2}t : a 문자가 2개
   "ct" : false
   "cat" : false
   "caat" : true
   "caaaat" : false
   {n,m} : n개이상 m개 이하 갯수
   ca{2,5}t : a 문자가 2개
   "ct" : false
   "cat" : false
   "caat" : true
   "caaaat" : true
   "caaaaaaat" : false
4. \\d: 숫자
5. \g<n> : n번째 그룹
6. ? : 0 또는 1
   ca?t : a문자가 없거나 1개인경우
   "ct" : true
   "cat" : true
   "caaaat" : false
7. * : 0개이상
   ca*t : a문자가 0개이상
   "ct" : true
   "cat" : true
   "caaaat" : true
8. + : 1개이상
   ca+t : a문자가 1개이상
   "ct" : false
   "cat" : true
   "caaaat" : true

10.정규식

```
import re

str = "The quick brown fox jumps over the lazy dog Te Thhe Thhhe."
str_list = str.split() #공백으로 단어를 분리.
print(str_list)
# T 문자로 시작하고, 0개이상의 h문자, e문자로 끝나는 패턴
pattern = re.compile("Th*e")
count = 0
for word in str_list :
    if pattern.search(word) :  #패턴에 맞는 문자열?
        count += 1
print("결과 1 : {1:s}:{0:d} ".format(count, "갯수"))
```

10.정규식

```
# re.I : 대문자 구분 없이.
pattern = re.compile("Th*e",re.I)
count = 0
for word in str_list :
    if pattern.search(word) :  #패턴에 맞는 문자열?
        count += 1
print("결과 2 : {1:s}:{0:d} ".format(count, "갯수"))

#결과2에 맞는 문자열 출력하기
print("결과 3 : ", end="")
for word in str_list :
    if pattern.search(word) :  #패턴에 맞는 문자열?
        print(word,end=",")
print()
```

10.정규식

```
#결과2에 맞는 문자열 출력하기
# (?P<match_word>Th*e) :대소문자 구분없이  Th*e 패턴을 match_word라는 패턴그룹으로 이름 설정
pattern = re.compile("(?P<match_word>Th*e)",re.I)
print("결과 4 :",end=" ")
for word in str_list :
    if pattern.search(word) :
        print("{0}".format(pattern.search(word).group
                    ("match_word")),end=",")
print()
#pattern.sub : 값의 치환함.
#Th*e 패턴의 맞는 문장을 "a"로 치환하기
pattern = re.compile("Th*e")
print("결과 5 : {0}".format(pattern.sub("a",str)))
print("결과 5 :",pattern.sub("a",str))
```