PYTHON_PROGRAMMING
클래스(CLASS) / 예외처리

클래스 정의

❖ 정의

클래스(class)는 속성과메소드를 공유하는 유사한 성질의 객체들을 하나로 그룹화한 것이다. 객체는 클래스의 구성원으로, 클래스 인스턴스(class instance) 또는 객체 인스턴스(object instance)라고도 한다.

클래스 내부에는 해당 클래스의 객체를 위한 데이터 구조와 메소드 구현의 세부 사항을 기술한다. 예를 들어 그림과 같이 속성 네 개와 메소드 두 개를 포함하는 사원 클래스를 정의할 수 있다. 사원 클래스에는 객체가 세 개 속해 있는데, 모두 속성이 같고 같은 메시지에 응답한다.

예제

```
result1= 0
def add1(n):
    global result1
    result1 += n
def sub1(n):
    global result1
    result1 -= n
result2=0
def add2(n):
    global result2
    result2 += n
def sub2(n):
    global result2
    result2 -= n
add1(8)
add1(19)
sub2(15)
print("1번 계산기 계산결과 : %d"% result1)
print("2번 계산기 계산결과 : %d"% result2)
```

예제

```
class Calculator:
        result = 0
        def add(self,n):
                self.result+=n
        def sub(self,n):
                self.result-=n
cal1 = Calculator() #1qjs 계산기 생성
cal2 = Calculator() #2qjs 계산기 생성
cal3 = Calculator() #3qjs 계산기 생성
cal1.add(19)
cal1.sub(5)
cal3.add(56)
cal3.sub(27)
print("1번 계산기 계산결과 %d"% cal1.result)
print("2번 계산기 계산결과 %d"% cal2.result)
print("3번 계산기 계산결과 %d"% cal3.result)
```

예제

```
class Calculator:
        result = 0
        def add(self,n):
                self.result+=n
        def sub(self,n):
                self.result-=n
cal1 = Calculator() #1번 계산기 생성
cal2 = Calculator() #2번 계산기 생성
cal3 = Calculator() #3번 계산기 생성
cal1.add(19)
cal1.sub(5)
cal3.add(56)
cal3.sub(27)
print("1번 계산기 계산결과 %d"% cal1.result)
print("2번 계산기 계산결과 %d"% cal2.result)
print("3번 계산기 계산결과 %d"% cal3.result)
print(type(cal1)) ; print(type(cal2)) ; print(type(cal3))
```

예제

```
class Person:
    name = 'Hong Kil Dong'
    gender = 'male'
    address = 'seoul'

    def set_name(self, name):
        self.name = name
    def print(self):
        print('my name is {0}'.format(self.name))

p1 = Person()
p2 = Person()
p1.name = "hong kildong"
print("p1's name is ", p1.name)
print("p2's name is ", p2.name)
```

예제

```
class Calculator:
    def setting(self,color,n1,n2):
        self.color = color
        self.n1=n1
        self.n2=n2
    def add(self):
        result=self.n1 +self.n2
        return result
red_calc= Calculator()
blue_calc= Calculator()
red_calc.setting('빨강색',15,5)
blue_calc.setting('파랑색,20,15)
print("파랑계산기 색깔(self.color):",blue_calc.color)
print("빨강계산기 색깔(self.color):",red_calc.color)
print("파랑계산기 덧셈결과:", blue_calc.add())
print("삘강계산기 덧셈결과:", red_calc.add())
```

생성자(Constructor)

- 생성자는 객체가 생성될 때 클래스의 멤버변수를 초기화 하거나 객체 생성과 동시에 해야할 작업을 기술하는 데 사용

- 파이썬에서는 클래스 내부의 메서드 이름을 __init__ 로 작성하면 이 메서드를 자동으로 생성자로 취급하게 됩니다.

- 생성자는 객체 생성 명령에 의해 자동으로 호출됩니다.

예제

```
class Account:
    def __init__(self,bank):
        self.balance=0
        self.bank=bank
    def get_balance(self):
            return self.balance
    def deposit(self,money):
            self.balance +=money
    def withdraw(self,money):
            self.balnace -= money

if __name__ == '__main__':
    woori = Account('우리은행')
    woori.deposit(15000)
    print('%s 잔액 : %d원:'%(woori.bank,woori.get_balance()))
```

상속

- 부모 클래스 (Parent clas, super class)
- 변수와 메서드를 물려주는 클래스

- 자식클래스(child class, sub class)
- 변수와 메서드를 물려받는 클래스를 자식 클래스라고 합니다.
- 파이썬의 상속표현은 클래스 이름 뒤에 데이터를 물려받을 부모클래스의 이름을 적어줍니다.

- 클래스를 상속하여 중복되는 클래스 코드를 최소화 함

예제

```
class  Player:
        def  __init__(self,user_id):
                self.user_id=user_id
                self.level=1
                self.hp=50
                self.atk=3
                self.exp=0
        def info(self):
                print('₩n=======캐릭터 정보========')
                print('아이디:',self.user_id)
                print('레벨:',self.level)
                print('공격력:',self.atk)
                print('체력:',self.hp)
                print(' ========================== ')

if __name__ == '__main__':

    class  Healer(Player):
            def heal(self):
                pass
    class  Tanker(Player):
            def tank(self):
                pass
    P1=Healer('아처')
    P2=Tanker('워리어')
    P1.info()
    P2.info()
```

예외처리[try ~except]

❖ 프로그램은 사용자 끊임없는 상호작용을 합니다.
그러나 사용자는 예측불가의 행동으로 잘못된 사용으로 인해
에러를 발생시킬 수 있습니다.
프로그램에서 예외가 발생하면 프로그램이 비정상 종료되기 때문에
예외처리 문법을 통해 프로그램이 정상 작동 할 수 있도록 구성하기
위함입니다.

Try : 예외발생 가능성이 있는 코드 작성
Except : 예외발생시 실행할 코드 작성

Try 내부에서 예외가 발생한다면 즉시 try의 실행을 중단하고
Except의 코드를 실행합니다.

예제

```
n1= 10
n2=0
try:
        result=n1/n2
        print("%d / %d = %d" %(n1,n2,result))
except:
        print("0으로 나눌 수 없습니다.")
print("프로그램 정상 종료!!")
```

예제

```
while true
        try  :
            n1= int(input(“정수1: “))
            n2= int(input(“정수2: “))
            break
        except:
            print(“숫자로만 입력하세요~”)
        print(“%d / %d = %d” %(n1,n2,result))
try:
        result=n1/n2
        print(“%d / %d = %d” %(n1,n2,result))
except:
        print(“0으로 나눌 수 없습니다.”)
print(“프로그램 정상 종료!!”)
```

예제

```
s = input('정수: ')
try :
    point = int(s)
    print(150 /point)
except ValueError:
    print("숫자로만 입력하세요~")
except ZeroDivisionError:
    print("0으로 나눌 수 없습니다.")
except:
    print("알수 없는 오류 발생. 점검 후 조치하겠습니다..")
print("프로그램 정상 종료!")
```

예제

```
pet = [‘거북이, ‘타란튤라’, ‘전갈’]
for I in range(4):
        try:
            print(pets[i],”키울래용!”
        except:
            print(“애완동물의 정보가 없습니다.”)
        finally:#예외처리(에러)와 상관없이 꼭 실행되야 하는경우
            print(“애완동물 넘 조아~~”)
print(“프로그램 정상 종료!”)
```