파이썬 - HW7

임베디드스쿨1기
Lv1과정
2020. 09. 21
박성환

1-1. Review(Operator Overloading)

1. Operator Overloading 정의

- 인스턴스 객체끼리 서로 연산을 할 수 있게끔 기존에 있는 연산자의 기능을 바꾸어 중복으로 정의하는 것

2. 예제

아래 예제를 보면, 인스턴스 객체 n에 '+' 연산자를 사용하여 100을 더하려는 코드가 보이는데 이는 지원되지 않는 연산 타입이므로 NumBox와 int간의 연산을 수행하기 힘들다는 것. +연산자를 사용하여 성공적으로 클래스 NumBox 내에 있는 변수 Num의 값을 증가시키기 위해 Overloading 기법 사용됨.

적용 전

```
>>> class NumBox:
    def __init__(self, num):
        self.Num = num

>>> n = NumBox(40)
>>> n + 100
Traceback (most recent call last):
  File "<pyshell#5>", line 1, in <module>
    n + 100
TypeError: unsupported operand type(s) for +: 'NumBox' and 'int'
```

```
n.__add__(100)
```

적용 후

```
>>> class NumBox:
    def __init__(self, num):
        self.Num = num
    def __add__(self, num):
        self.Num += num
    def __sub__(self, num):
        self.Num -= num

>>> n = NumBox(40)
>>> n + 100
>>> n.Num
140
>>> n - 110
>>> n.Num
30
```

1-1. Review(Operator Overloading)

3. 정의되어 있는 주요 Method
인스턴스 객체끼리 서로 연산을 할 수 있게끔 기존에 있는 연산자의 기능을 바꾸어 중복으로 정의하는 것

참고 : __repr__(self): Print문과 동일 (repr : representation)

연산자/함수 (Operator/Function)	메소드(Method)	설명(Description)
+	__add__(self, other)	덧셈
*	__mul__(self, other)	곱셈
-	__sub__(self, other)	뺄셈
/	__truediv__(self, other)	나눗셈
%	__mod__(self, other)	나머지
<	__lt__(self, other)	작다(미만)
<=	__le__(self, other)	작거나 같다(이하)
==	__eq__(self, other)	같다
!=	__ne__(self, other)	같지 않다
>	__gt__(self, other)	크다(초과)
>=	__ge__(self, other)	크거나 같다(이상)
[index]	__getitem__(self, index)	인덱스 연산자
in	__contains__(self, value)	멤버 확인
len	__len__(self)	요소 길이
str	__str__(self)	문자열 표현

1-1.Review(Overloading)

1. obj1, obj2 객체는 일종의 포인터 변수라 생각

```
"""
__repr__메소드가 오버로딩 되어있지 않은 경우 print문 사용시 인스턴스 객체의 주소를 return함
obj1 + obj2은 주소의 덧셈으로 이상한 주소값을 return함
"""


class OpOverload(object):
    def __init__(self, number):
        self.Number = number

    def __add__(self, other):
        print("__add__ is called")
        return OpOverload(self.Number + other.getNumber())

    def __sub__(self, other):
        print("__sub__ is called")
        return OpOverload(self.Number - other.getNumber())

    def getNumber(self):
        return self.Number

obj1 = OpOverload(10)
obj2 = OpOverload(30)

print(obj1)
print(obj2)
print(obj1 + obj2)
print(obj1 - obj2)
```

```
<__main__.OpOverload object at 0x000001CE6A4A9A88>
<__main__.OpOverload object at 0x000001CE6A4A9288>
__add__ is called
<__main__.OpOverload object at 0x000001CE6A4ABOC8>
__sub__ is called
<__main__.OpOverload object at 0x000001CE6A483288>
```

1-1.Review(Overloading)

2. 이전페이지와 비교(__repr__ 추가)

```
"""
Q1. other.getNumber()와 other.Number의 범용성 측면에서의 차이?

"""

"""
__add__ 실행 -> __repr 실행
print(obj1 + obj2) 에서 사실 print(obj1)하면 주소가 출력됨 즉 obj1 은 주소값
obj1 + obj2 는 주소값끼리의 덧셈으로 C언어에서 포인터 변수끼리의 덧셈은 알수 없는 위치를 가리키기 때문에
사용 불가능 하지만 이를 파이썬에서는 기본정의된 메서드를 호출함으로써 이를 가능하게 해줌

"""


class OpOverload(object):
    def __init__(self, number):
        self.Number = number

    def __repr__(self): #representation
        return str(self.Number)

    def __add__(self, other):
        print("__add__ is called")
        return OpOverload(self.Number + other.getNumber())
        #return OpOverload(self.Number + other.Number)

    def __sub__(self, other):
        print("__sub__ is called")
        return OpOverload(self.Number - other.getNumber())
        #return OpOverload(self.Number - other.Number)

    def getNumber(self):
        return self.Number

obj1 = OpOverload(10)
obj2 = OpOverload(30)

print(obj1 + obj2)
print(obj1 - obj2)
```

```
__add__ is called
40
__sub__ is called
-20
```

1-1.Review(Overloading)

3.뒷페이지 이어서

```
"""
Q2. __str__ 과 __repr__의 차이?
Q3. Protected 잘 사용되나? 어디에 사용되나?
Q4. return Vector([w[i] + self.getVector()[i] for i in range(len(self))])
    return Vector([w[i]+ self[i] for i in range(len(self))])
    return Vector([w.getVector()[i] + self[i]])
    범용성에 대해 잘 이해가 가질 않음

"""
class VectorError(Exception): #예외발생시키는 최상위 객체를 상속받음
    def __init__(self, err):
        self.value = err

    def __str__(self): #__str__
        return repr(self.value)

class Vector(object): #object는 void형과 같은 개념
    def __init__(self, v): #생성자
        self.__name__ = "Vector" #__name__
        if type(v) is list:
            self.__v = v
            self.__Rn = len(self) #오버로딩 된 len 참조
        else:
            raise VectorError("Invalid Vector")

    def __add__(self,w):
        if type(w) is list:
            if len(w) == len(self):
                return Vector([w[i] + self.getVector()[i] for i in range(len(self))])
                #return Vector([w[i]+ self[i] for i in range(len(self))])
                #return Vector([w.getVector()[i] + self[i]])
            else :
                raise VectorError("Both are not in the same Rn Space. [v, l]")
        elif hasattr(w, '__name__') and w.__name__ == "Vector":
            if len(w) == len(self):
                return Vector([w.getVector()[i] + self.getVector()[i] for i in range(len(self))])
            else :
                raise VectorError("Both are not in the same Rn Space. [v, v]")
        else:
            raise VectorError("Invalid Vector")

    def __len__(self): #len 메서드 오버라이딩
        return len(self.__v) #len(객체) => len(객체.__v) 하도록 오버라이딩
```

수업시간에 질문하기

1-1.Review(Overloading)

3.앞페이지와 이어

```
44
45      def __repr__(self):
46          return str(self.__v)
47
48      def getVector(self):
49          return self.__v
50
51  if __name__ == "__main__": #다른 import된것에서의 호출이 아닌 현재 직접 실행된 모듈만 출력하도록 하기 위함
52      v1 = Vector([1,2,3])
53      w1 = Vector([4,5,6])
54      print(v1 + w1)
55
56      v2 = Vector([1,-2])
57      w2 = Vector([-1,4])
58      print(v2 + w2)
59
```

```
[5, 7, 9]
[0, 2]
```

참고. __name__(내장 변수)

1. 정의
현재 모듈의 이름을 담고 있는 내장 변수
직접 실행된 모듈의 경우 __main__ 이라는 값을 가지며
직접 실행되지 않은 import 된 경우 모듈이름(파일명)을 가지게 된다.

<코드 - 모듈>

```
#module.py

def hello():
    print("Hello!")

print(__name__)
```

<코드 - 메인>

```
#main.py

import module

print(__name__)
module.hello()
```

<결과 - 메인>

```
module
__main__
Hello!
```

① import module : module.py 를 import 하면 해당 스크립트 파일이 한 번 실행됨(즉, module.py안의 코드가 실행됨)(import하므로 **module** 출력)
② print(__name__) : main.py의 print문 실행 (직접 실행되므로 **__main__** 출력)
③ module.hello() : print("Hello!") 출력

참고. __name__(내장 변수)

2. 사용예

if __name__ == "__main__"이라는 조건문을 넣어주고 그 아래에 직접 실행 시켰을 때만 실행되기를 원하는 코드를 넣어준다.

모듈내에서 테스트나 로그 출력 부분들이 있는데 실제 impor를 하면 해당 출력들은 사용하지 않으면서 module내에 정의된 기능들만 사용할 수 있기 때문에 유용

참고. hasattr(object, name)

Object 내에 name에 해당하는 attribute(속성)이 있으면 **true**, 없으면 **fail**을 리턴

```
class foobar():
    data = [1, 2, 3, 4]
    def __init__(self, val):
            self.val = val

>>> x = foobar
>>> y = foobar(['a', 'b'])
>>> z = foobar([1, 2])
>>> hasattr(x, 'data')
True
>>> hasattr(y, 'data')
True
>>> hasattr(x, 'val')
False
>>> hasattr(y, 'val')
True
>>> delattr(x, 'data')
>>> hasattr(x, 'data')
False
```

hasattr : has attribute로 생각하자

hasattr(x, 'data') : x 객체에 'data' 라는 attribute가 존재여부에 따라 **'true/false'**

ex)

hasattr(w, '__name__') : w객체에 __name__이라는 속성이 존재하면 true

1-2.Review(Class2)

예제(1) – 메서드를 통하여 속성의 값을 가져오거나 저장하는 경우

```
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

james = Person()
james.set_age(20)
print(james.get_age())
```

실행 결과

```
20
```

이렇게 하면 메서드를 직접 사용할 수 있다
즉, 은닉이 되지 않는다.?
따라서 메서드를 은닉하기 위해 **다음 페이지처럼** 한다.

getter : 값을 가져오는 메서드를 칭함
setter : 값을 저장하는 메서드를 칭함

1-2.Review(Class2)

예제(2) – 메서드 은닉하기

```
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

james = Person()
james.age = 20      # 인스턴스.속성 형식으로 접근하여 값 저장
print(james.age)    # 인스턴스.속성 형식으로 값을 가져옴
```

실행 결과

```
20
```

getter => @property
setter => @메서드이름.setter

set_age/get_age 따로 안맡르고
같은 age 메서드로 만들 수 있음

메서드를 속성처럼 사용 가능
(= 함수이름을 변수명처럼 사용 가능)

@property가 @메서드이름.setter보다 앞에 있어야 함

1-2.Review(Class2)

예제(3) – 메서드 은닉하기

```
1  from math import pi
2
3  class Circle(object):
4      def __init__(self,r):
5          self.__r = r
6
7      @property #함수를 변수처럼 쓰게 하는것이 주목적
8      def area(self): #원의 넓이
9          return pi * self.__r**2
10
11     @property
12     def circumference(self): #원둘레 길이
13         return 2 * pi * self.__r
14
15     @property  #getter
16     def radius(self): #반지름 길이
17         return self.__r
18
19     @radius.setter #setter
20     def radius(self,r):
21         self.__r = r
22
23
24
25 c = Circle(3.0)
26 print(c.radius)
27 print(c.area)
28 print(c.circumference)
29
30 c.radius = 7.0
31 print(c.radius)
32 print(c.area)
33 print(c.circumference)
```

```
3.0
28.274333882308138
18.84955592153876
7.0
153.93804002589985
43.982297150257104
```

@property 사용하여 메서드를 속성처럼 사용

radiu라는 동일 이름의 setter/getter 만듬

self.r 뿐만 아니라 self.__r도 변경 가능한데
이러면 은닉이며 public, private가 무의미한 것 아닌가?

1-2.Review(Class2)

예제(4) – 접근제어

```
1  class DummyPrint:
2      def __init__(self):
3          self.var1 = 3
4          self._var2 = 'Python'
5          self.__var3 = 'Class'
6
7  dp = DummyPrint()
8
9  print(dp.var1)
10 print(dp._var2)
11 #print(dp.__var3) #private 속성에 외부에서 접근 불가능(아래처럼 써야함)
12 print(dp._DummyPrint__var3) #이런식으로 표현하는 것이 규칙이다라고 우선 이해
```

```
3
Python
Class
```

print(dp.__var3) : 접근 불가능 **(X)**
print(dp._DummyPrint__var3) : 접근 가능 **(O)**
(외부에서 이런식으로 접근 가능하면 private의미가 있을까?)

2.Preview(Class2)

예제(5) – 상속예제

```
1  """
2  상속예제
3  """
4  class Animal:
5      def __init__(self, name, height, weight):
6          self.Name = name
7          self.Height = height
8          self.Weight = weight
9
10     def info(self):
11         #print("Name: ", str(self.Name))
12         print("Name: {0}".format(self.Name))
13         print("Height: ", str(self.Height))
14         print("Weight: ", str(self.Weight))
15
16 class Carnivore(Animal):
17     def __init__(self, name, height, weight, feed, sound):
18         Animal.__init__(self, name, height, weight)
19         self.Feed = feed
20         self.Sound = sound
21
22     def sounds(self):
23         print(str(self.Name) + ": " + str(self.Sound))
24
25     def info(self):
26         Animal.info(self)
27         print("Food: ", str(self.Feed))
28         print("Sound: ", str(self.Sound))
29
30 wolf = Carnivore("Timber Wolf", 140, 75, "Meat", "Howl")
31
32 wolf.info()
33 wolf.sounds()
34
```

18줄 - Animal.__init__ 부분 참고

```
Name: Timber Wolf
Height:  140
Weight:  75
Food:  Meat
Sound:  Howl
Timber Wolf: Howl
```

2.Preview(Class2)

예제(6) – 부모 클래스 super()로 표현

```
"""
super() 사용
"""
class Parent(object):
    def __init__(self, number):
        self.Number = number

    def printMsg(self):
        print("I'm a Super Class")

class Child(Parent):
    def __init__(self, number):
        # super(Child, self).__init__(number) #부모 메서드 호출시 'super()' 사용
        super().__init__(number)
    def printMsg(self):
        print("I'm a Sub class: [%s]" %str(self.Number))
        #super(Child, self).printMsg()
        super().printMsg()

c = Child(5)
c.printMsg()

```

```
I'm a Sub class: [5]
I'm a Super Class
```

3.HW1

Q1 C언어로 파이썬에서 연산자 오버로딩을 통해 쉽게 계산할 수 있었던 복소수 연산체계를 만들어보자!

- 이 때 드 무아브르 법칙이 굉장히 유용하게 사용될 수 있다.
- 복소수의 극좌표 형식 등등
  (AC 회로 - Phasor Domain): 위상
- AC 회로를 해석할 때 복소수 기반으로 해석
- DC-DC(컨버터), DC-AC(인버터) 설계에 활용된다.

드 무아브르 공식 활용 및 적용?

```c
/*
 * 정현파 교류전압이나 전류를 페이저로 변환하고 그 페이저를 복소수에서 배운
 * 극좌표 형식과 같은 것으로 취급하면 예상보다 훨씬 간단하게 정현파 교류회로에서의
 * 복잡한 수식을 계산할 수 있음
 *
 * 1. 복소수 표현은 극좌표계로 받는다(반지름, 각도) => 구조체로 표현하기
 * 2. 사칙연산에 해당하는 명령어와 함수를 맵핑(테이블 형태로)
 * 3. 연산자체가 복잡하지 않으니 과도하게 포인터 사용은 굳이 하지 않는다.
 * 4. 최종적으로 반지름과 각도로 표현되도록 함
 */

/*****************************************************************************************************
 *  Include
 *****************************************************************************************************/
#include <stdio.h>
#include <math.h>

/*****************************************************************************************************
 *  Define
 *****************************************************************************************************/
#define PI 3.1415926535

/*****************************************************************************************************
 *  Type Def
 *****************************************************************************************************/
typedef struct polarPosition
{
    double radius;
    double degree;
} polarPosition;

typedef struct orthogonalPosition
{
    double real;
    double image;
} orthogonalPosition;

typedef polarPosition (*CalcFuncPtr)(polarPosition, polarPosition);

typedef struct OperatingCmd
{
    char* cmd;
    CalcFuncPtr calc;
} OperatingCmd;

/*****************************************************************************************************
 *  Func Prototype
 *****************************************************************************************************/
polarPosition Cal_add(polarPosition A, polarPosition B);
polarPosition Cal_sub(polarPosition A, polarPosition B);
polarPosition Cal_mux(polarPosition A, polarPosition B);
polarPosition Cal_div(polarPosition A, polarPosition B);
orthogonalPosition polarTorthogonal(polarPosition A);
polarPosition orthogonalTopolar(orthogonalPosition A);
polarPosition Calculator (polarPosition A, polarPosition B, char* opmode);
```


것도 없다.

3.HW1

```c
/*****************************************************************************************************
*  Variable
*****************************************************************************************************/
OperatingCmd Op_Cmd[] =
{
    {"ADD"  ,   Cal_add},
    {"SUB"  ,   Cal_sub},
    {"MUX"  ,   Cal_mux},
    {"DIV"  ,   Cal_div},
    {NULL   ,   NULL}
};

/*****************************************************************************************************
*  Main
*****************************************************************************************************/
int main(void)
{
    polarPosition V1 = {10, -36.9};
    polarPosition V2 = {10, 53.1};
    polarPosition Result1, Result2, Result3, Result4;

    Result1 = Calculator(V1, V2, "ADD");
    Result2 = Calculator(V1, V2, "SUB");
    Result3 = Calculator(V1, V2, "MUX");
    Result4 = Calculator(V1, V2, "DIV");

    printf("ADD : radius = %.2f, degree = %.2f\n", Result1.radius, Result1.degree);
    printf("SUB : radius = %.2f, degree = %.2f\n", Result2.radius, Result2.degree);
    printf("MUX : radius = %.2f, degree = %.2f\n", Result3.radius, Result3.degree);
    printf("DIV : radius = %.2f, degree = %.2f\n", Result4.radius, Result4.degree);

    return 0;
}
```

```c
/*****************************************************************************************************
*  Func (Calc - add, sub, mux, div)
*****************************************************************************************************/
polarPosition Calculator (polarPosition A, polarPosition B, char* opmode)
{
    OperatingCmd *cmdptr;

    for(cmdptr = Op_Cmd; cmdptr->cmd; cmdptr++)
    {
        if(*(cmdptr->cmd) == *opmode)
        {
            return cmdptr->calc(A, B);
        }
    }

    if(cmdptr->cmd == NULL)
    {
    }
}

polarPosition Cal_add(polarPosition A, polarPosition B)
{
    orthogonalPosition orTemp1;
    orthogonalPosition orTemp2;
    orthogonalPosition orTempResult;

    orTemp1 = polarTorthogonal(A);
    //printf("%f+j%f\n", orTemp1.real, orTemp1.image);

    orTemp2 = polarTorthogonal(B);
    //printf("%f+j%f\n", orTemp2.real, orTemp2.image);

    orTempResult.real = orTemp1.real + orTemp2.real;
    orTempResult.image = orTemp1.image + orTemp2.image;
    //printf("%f+j%f\n", orTempResult.real, orTempResult.image);

    return orthogonalTopolar(orTempResult);
}

polarPosition Cal_sub(polarPosition A, polarPosition B)
{
    orthogonalPosition orTemp1;
    orthogonalPosition orTemp2;
    orthogonalPosition orTempResult;

    orTemp1 = polarTorthogonal(A);
    orTemp2 = polarTorthogonal(B);

    orTempResult.real = orTemp1.real - orTemp2.real;
    orTempResult.image = orTemp1.image - orTemp2.image;
    //printf("%f+j%f\n", orTempResult.real, orTempResult.image);

    return orthogonalTopolar(orTempResult);
}
```

3.HW1

```
polarPosition Cal_mux(polarPosition A, polarPosition B)
{
    orthogonalPosition orTemp1;
    orthogonalPosition orTemp2;
    orthogonalPosition orTempResult;

    orTemp1 = polarTorthogonal(A);
    orTemp2 = polarTorthogonal(B);

    orTempResult.real = (orTemp1.real * orTemp2.real) - (orTemp1.image * orTemp2.image);
    orTempResult.image = (orTemp1.image * orTemp2.real) + (orTemp2.image * orTemp1.real);
    //printf("%f+j%f\n", orTempResult.real, orTempResult.image);

    return orthogonalTopolar(orTempResult);
}

polarPosition Cal_div(polarPosition A, polarPosition B)
{
    orthogonalPosition orTemp1;
    orthogonalPosition orTemp2;
    orthogonalPosition orTempResult;

    orTemp1 = polarTorthogonal(A);
    orTemp2 = polarTorthogonal(B);

    orTempResult.real = ((orTemp1.real * orTemp2.real) + (orTemp2.image * orTemp1.image)) /
                        (orTemp2.real * orTemp2.real + orTemp2.image * orTemp2.image);
    orTempResult.image = ((orTemp1.image * orTemp2.real) - (orTemp2.image * orTemp1.real)) /
                         (orTemp2.real * orTemp2.real + orTemp2.image * orTemp2.image);
    //printf("%f+j%f\n", orTempResult.real, orTempResult.image);

    return orthogonalTopolar(orTempResult);
}

/*****************************************************************************************************
*   Driver
*****************************************************************************************************/
orthogonalPosition polarTorthogonal(polarPosition A)
{
    orthogonalPosition var;
    A.degree = PI * A.degree / 180;

    var.real = A.radius * cos(A.degree);
    var.image = A.radius * sin(A.degree);
    return var;
}

polarPosition orthogonalTopolar(orthogonalPosition A)
{
    /*
    * 디그리 -> 라디안 : PI / 180 곱해줌
    * 라디안 -> 디그리 : 180 / PI 곱해줌
    */
    polarPosition var;
    var.radius = sqrt(pow(A.real,2) + pow(A.image,2));
    var.degree = atan2(A.image, A.real) * 180/PI; //atan2가 -pi ~ pi 까지이기 때문에 음의 값 표현 가능
    return var;
}
```

3.HW2

Q2 void *를 활용해서 범용성(어떤 상황에서든 동작할 수 있는) 강건한 코드를 만들어보자!
주제가 여러가지가 될 수 있으므로 각자 한 번 생각해서 구현을 해보도록 한다.
예) N by N 형태의 공간이 존재한다.
여기에 배치될 물건의 크기가 2 by 2, 2 by 3, 3 by 3, 3 by 4, 5 by 2, 5 by 3 이 존재한다.
이들을 모두 일관되게 배치할 수 있는 효율적인 방법을 찾으시오.
뿐만 아니라 가장 공간의 낭비가 적게 만들려면 어떻게 해야하는지도 고민해보자!

감사합니다.