파이썬 - HW7

임베디드스쿨1기

Lv1과정

2020. 09. 22

박하늘

[HW1] C 언어 복소수 연산 체계

HW 1. C언어로 파이썬에서 연산자 오버로딩을 통해 쉽게 계산할 수 있었던 복소수 연산체계를 만들어보자!
이때 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.

```c
#include <stdio.h>

typedef struct {
        double num;
        double jnum;
}Complex;

int main(void)
{
        Complex c1, c2;

        printf("복소수 다항식1: ");
        scanf("%lf %lf", &c1.num, &c1.jnum);

        printf("복소수 다항식2: ");
        scanf("%lf %lf", &c2.num, &c2.jnum);

        printf("실수부 합: %lf, 허수부 합:%lf\n", c1.num+c2.num, c1.jnum+c2.jnum);

        return 0;
}
```

```
(base) haneulpark@haneulpark-910S3K-9310SK-910S3P-911S3K:~/문서/HW/Python/7회차$
 vim 0921_complex.c
(base) haneulpark@haneulpark-910S3K-9310SK-910S3P-911S3K:~/문서/HW/Python/7회차$
 gcc 0921_complex.c
(base) haneulpark@haneulpark-910S3K-9310SK-910S3P-911S3K:~/문서/HW/Python/7회차$
 ./a.out
복소수 다항식1: 5 7
복소수 다항식2: 3 6
실수부 합: 8.000000, 허수부 합:13.000000
```

복소수 연산 합만 구현함

[Review] Python Class 2

1) @property

: 외부에서 클래스 내부 변수를 참조하기 위한 함수

- 함수 호출을 변수 호출처럼 사용 가능

(함수호출 person.full_name() / 변수호출 person.full_name)

- @property를 사용하면 getter, setter를 간단하게 구현 가능

[getter, 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

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

```
20
```

get method = @property

set method = @method_name.setter

[@property, @(property 매서드).setter]

```
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 #함수 호출을 변수 호출처럼 사용 가능
print(james.age)    # 인스턴스.속성 형식으로 값을 가져옴
```

```
20
```

[Review] Python Class 2

2) 접근 제어자 (Access Modifier)

: 접근 제어자에는 public, private, protected, default가 있다.

① public → 접두사에 밑줄이 없음
- 클래스내, 클래스 외의 어디서도 액세스 가능
- 제 3자가 접근할 수 있다.

② protected → 접두사에 한개의 밑줄(_)을 적용
- 같은 클래스 및 자식 클래스에서 액세스 가능
- 제 3자는 접근 불가

③ private → 접두사에 두개의 밑줄(__)을 적용
- 같은 클래스 안에서만 엑세스 가능
- 외부 접근할 수 없다.
- 단 예외 있음. (객체 + '_' + 함수명 + '_ _' + 변수 형태로 private사용 가능함)

```
class DummpyPrint:
    def __init__(self):
        self.var1 = 3
        self._var2 = 'Python'
        self.__var3 = 'Class'

dp = DummpyPrint()

print(dp.var1)
print(dp._var2)
# print(dp__var3)
print(dp._DummpyPrint__var3)
```

```
3
Python
Class
```

[Review] Python Operator Overloading

3) 연산자 오버로딩 예제 코드 분석(Python & C)

- __repr__(): 문자 표현(to string)
- __repr__()이 없으면 to string이 아닌 주소값이 반환된다.
- 연산자 오버로딩 : obj1 + obj2 는 obj1__add__(obj2) 를 뜻한다.
- self.Number은 obj1이고 other.getNumber()은 obj2를 지칭한다.
- 범용성을 높이기 위해 obj2를 other.getNumber()로 받는다.

```python
class OpOverload(object):
    def __init__(self, number):
        self.Number = number
    def __repr__(self):
        return str(self.Number)# to_string
    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 + obj2) #obj1__add__(obj2) #print가 호출될때 __repr__()가 실행되어 to string 형태가 됨
print(obj1 - obj2) #obj1__sub__(obj2)
```

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

```c
int *p1 = (int *)malloc(sizeof(int) * 3);
int *p2 = (int *)malloc(sizeof(int) * 4);

printf("p1 = 0x%x\n", p1);
printf("p2 = 0x%x\n", p2);

printf("p1 + p2 = 0x%x\n", p1 + p2); //p1 -> number + p2 -> number
```

메서드(Method)	연산자(Operator)	사용 예
__add__(self, other)	+ (이항)	A + B, A += B
__pos__(self)	+ (단항)	+A
__sub__(self, other)	- (이항)	A - B, A -= B
__neg__(self)	- (단항)	-A
__mul__(self, other)	*	A * B, A *= B
__truediv__(self, other)	/	A / B, A /= B
__floordiv__(self, other)	//	A // B, A //= B
__mod__(self, other)	%	A % B, A %= B
__pow__(self, other)	pow(), **	pow(A, B), A ** B
__lshift__(self, other)	<<	A << B, A <<= B
__rshift__(self, other)	>>	A >> B, A >>= B
__and__(self, other)	&	A & B, A &= B
__xor__(self, other)	^	A ^ B, A ^= B
__or__(self, other)	|	A | B, A |= B
__invert__(self)	~	~A
__abs__(self)	abs()	abs(A)

[Preview] Python Class 2

1) 부모 자식 상속 개념

```
class Animal:
    def __init__(self, name, height, weight):
        self.Name = name
        self.Height = height
        self.Weight = weight

    def info(self):
        print("Name: ", str(self.Name))
        print("Height ", str(self.Height))
        print("Weight ", str(self.Weight))

class Carnivore(Animal): #자식이 부모클래스를 상속받음, 즉 부모의 모든 속성이 포함됨
    def __init__(self, name, height, weight, feed, sound):
        Animal.__init__(self, name, height, weight)
        self.Feed = feed
        self.Sound = sound

    def sounds(self):
        print(str(self.Name) + ": " + str(self.Sound))

    def info(self):
        Animal.info(self)
        print("Food: ", str(self.Feed))
        print("Sound: ", str(self.Sound))

wolf = Carnivore("Timber Wolf", 140, 75, "Meat", "Howl")

wolf.info() #Animal의 info + Carnivore의 info 정보
print('-------------------------')
wolf.sounds() #print(str(self.Name) + ": " + str(self.Sound))
```

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

[Preview] Python Class 2

2) Super()

: 자식 클래스에서 부모클래스의 내용을 사용하고 싶을 경우 사용

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

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

class Child(Parent):# 자식클래스(부모클래스) 아빠매소드를 상속받겠다
    def __init__(self, number):
        super(Child, self).__init__(number) #Parent매소드 __init__()상속받겠다.

    def printMsg(self):
        print("I'm a Sub Class: [%s]" % str(self.Number))
        super(Child, self).printMsg() #Parent 매소드 printMsg() 상속받겠다.

c = Child(3)
c.printMsg()
```

```
I'm a Sub Class: [3]
I'm a Super Class
```