파이썬 – HW8

임베디드스쿨1기

Lv1과정

2020. 09. 23

강경수

▣ Python 연산자 오버로딩

2020.09.15

1. 연산자 오버로딩이란?
- 미리정의된 연산자들을 사용자가 재 정의하여 사용 하는 것.

2. 기본 MagicMtehod 종류

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)

1. Python class

3. 사용 예시

```
class Numbox(object):
    def __init__(self,num):
        self.Num = num
    def __add__(self,num):
        self.Num += num
    def __sub__(self,num):
        self.Num -= num
    def __repr__(self):
        return str(self.Num)


n = Numbox(40)
n + 100
print(n)
n - 100
print(n)

140
40
```

1. Python class

4. 상호연산을 위해 범용성을 높인 코드.

```
class Numbox(object):
    def __init__(self,num):
        self.Num = num
    def __add__(self,other):
        return Numbox(self.Num + other.getNumber())
    def __sub__(self,other):
        return Numbox(self.Num - other.getNumber())
    def __repr__(self):
        return str(self.Num)
    def getNumber(self):
        return self.Num


n = Numbox(40)
m = Numbox(50)
print(m)
print(n)
print(m+n)
```

```
50
40
90
```

1. Python class

5. @ property란 무엇인가

```
class Person :
    def __init__(self):
        self.__name = 'Kang'
    def get_name(self):
        return self.__name
    def set_name(self, name):
        self.__name = name
```

@ property를 사용하여 get, set에 대해서 직관적으로 사용이 가능하다.

```
class Person :
    def __init__(self):
        self.__name = 'Kang'
    @property
    def name(self):
        return self.__name
    @name.setter
    def name(self, name):
        self.__name = name

person = Person()
print(person.name)    # Kang
person.name = 'kim'
print(person.name)    # kim
```

1. Python class

6. private,protected,public

-private : private 로 선언된 경우 해당 클래스에서만 접근 가능

-protected : protected로 선언된 경우 해당 클래스 또는 해당 클래스를 상속받은 클래스에서만

-public : public으로 선언된 경우 어떤 클래스에서든 접근 가능

```
class access:
    def __init__(self):
        self.public = 'PUBLIC'
        self.__private = "PRIVATE"
        self._protected = "PROTECTED"

    def print_test(self):
        print(self.public)
        print(self.__private)
        print(self._protected)


a = access()
a.print_test()
print(a.public)
print(a._protected)
print(a.__private)
```

```
PUBLIC
PRIVATE
PROTECTED
PUBLIC
PROTECTED
```

1. Python class

7. class 부모 상속의 개념

```
class Person:
    def __init__(self, name, age, gender):
        self.Name = name
        self.Age = age
        self.Gender = gender

    def aboutMe(self):
        print("저의 이름은 " + self.Name + "이구요, 제 나이는 " + self.Age + "살 입니다.")

    def superclass(self):
        print("i am super class")


class Employee(Person):
    def __init__(self, name, age, gender, salary, hiredate):
        Person.__init__(self, name, age, gender)
        self.Salary = salary
        self.Hiredate = hiredate

    def doWork(self):
        print("열심히 일을 합니다.")

    def aboutMe(self):
        Person.aboutMe(self)
        print("제 급여는 " + self.Salary + "원 이구요, 제 입사일은 " + self.Hiredate + " 입니다.")
        super(Employee,self).superclass()

objectEmployee = Employee("경수", "26", "남", "5000", "2019년 7월 1일")
objectEmployee.doWork()
objectEmployee.aboutMe()
```

- class(부모이름): 방식으로 사용한다.
- 부모의 모든 정보를 사용할 수 있다.
- super로 부모의 모든것들에 접근할 수 있다.