파이썬의 클래스와 모듈

클래스(Class)란?

- Class란, 함수나 변수들을 모아놓은 집합체
  - ex) 영화 제목 list, 점수 list, scoring 함수(), ranking 함수() 들을 모아놓음.
  - class에서 함수를 (instance) method, 변수를 class variable라고 함.
- 또한 class는 설계에 따라 코딩만 해놓은 상태를 말함.
  - 이런 class가 memory에 올라가 있는 상태를 instance(객체)라고 함.
  - 하나의 class는 이런 instance들을 무수히 많이 만들 수 있음.
  - ex) 자주 쓰이는 붕어빵 틀(class), 붕어빵(instance)

Class

```
class Simple:
    pass
```

Instance

```
a = Simple()
```

Class Example

class_ex.py

```
# -*- coding: cp949 -*-

class Service:

    def sum(self, a, b):
        result = a + b
        print("%s + %s = %s 입니다." % (a, b, result))
```

해당 python file을 수행하기 위해 python file을 import 시킨다.

```
>>> import class_ex
```

class의 type은 class obj

```
>>> type(Service)
<type 'classobj'>
```

class를 pey에 선언한 뒤 class의 type은 instance

```
>>> pey = Service()
>>> type(pey)
<type 'instance'>
```

Class Example – cont'd

```
# -*- coding: cp949 -*-

class Service:

    def sum(self, a, b):
        result = a + b
        print("%s + %s = %s 입니다." % (a, b,
result))
```

pey객체의 sum을 수행하면 다음과 같다.

```
>>> pey.sum(1, 1)
1 + 1 = 2 입니다.
```

예제에서 sum()의 인자 수는 3개였음. 그러나 여기선 2개만 넘겨주는데..?
→ self 때문임 (self를 사용해야 객체의 함수로 사용할 수 있음)

- self란?
  - class에서 정의된 method는 class가 객체화 되면서 instance method라 함.
  - instance method는 첫 번째 인수로 넘어오는 class의 instance에 대해 작동하는 함수임.
  - 즉, method를 호출하는 instance에 대해 명시해주는 것.
  - self가 이런 역할을 해줌.
  - 따라서 instance method를 사용하기 위해선 첫 번째 인자는 무조건 self이어야 함.

Class 사용법

- ‘class’는 class를 만들 때 쓰이는 예약어
- ‘class’ 뒤에는 바로 class의 이름을 써주어야 함
- Class 이름 뒤에 상속할 class가 있으면 상속할 class의 이름을 씀
- Class 내부에는 instance variable과 method들을 정의함

```
class 클래스이름[(상속 클래스명)]:
    <클래스 변수 1>
    <클래스 변수 2>
    ...
    def 클래스함수1(self[, 인수1, 인수2,,,]):
        <수행할 문장 1>
        <수행할 문장 2>
        ...
    def 클래스함수2(self[, 인수1, 인수2,,,]):
        <수행할 문장1>
        <수행할 문장2>
        ...
    ...
```

예제: 사칙연산 Class

아래 주석 부분에 method를 정의해주세요.

```
# -*- coding: cp949 -*-
class Calc:
    def sum(self, a, b):
        result = a + b
        print("%s + %s = %s 입니다." % (a, b, result))
    #subtraction
    #multiplication
    #division
```

Result

```
>>> import class_ex
>>> calc = Calc()
>>> calc.sum(1,2)
1 + 2 = 3 입니다.
>>> calc.sub(5,1)
5 - 1 = 4 입니다.
>>> calc.multi(2,3)
2 * 3 = 6 입니다.
>>> calc.divi(6,2)
6 / 2 = 3 입니다.
```

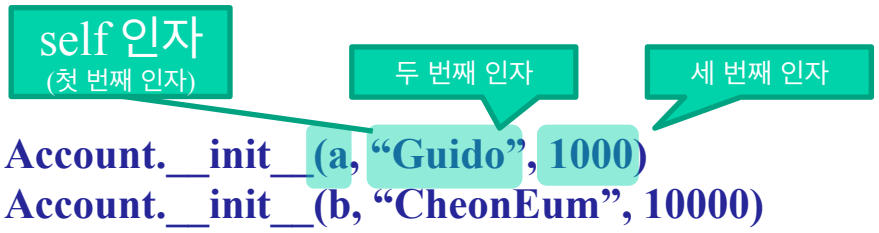
Class 내 연산자 함수

함수	설명	예제
__init__	생성자(Constructor), 인스턴스가 만들어 질 때 호출	
__del__	소멸자(Destructor) 인스턴스가 사라질 때 호출	
__add__	연산자 "+"	x + y
__or__	연산자 "|"	x | y
__repr__	print	print x
__call__	함수호출 X()했을 때 호출	
__getattr__	자격부여	x.method
__getitem__	인덱싱	x[i]
__setitem__	인덱스 치환	x[key] = value
__getslice__	슬라이싱	x[i:j]
__cmp__	비교	x > y

Class – __init__

- class의 instance는 class 객체를 함수로서 호출할 때 생성된다.
- 그럼, 새로운 instance가 생성되고 class의 __init()__ method에 이 instance가
  전달된다.
- __init()__은 새롭게 생성된 instance **self**와 class 객체를 함수로서 호출할 때
  제공한 **인수를** 받는다.
  - __init()__은 넘겨 받은 인자로 instance를 초기화 해주는 기능이다.

```
# 계좌를 몇 개 생성할 때

a = Account("Guido", 1000)
b = Account("CheonEum", 10000)
```



Class – __del__

- __del__ 함수는 실행 중인 instance를 소멸시킬 수 있음

```
>>> class HousePark:
...     lastname = "박"
...     def __init__(self, name):
...             self.fullname = self.lastname + name
...     def travel(self, where):
...             print("%s, %s여행을 가다." % (self.fullname, where))
...     def __del__(self):
...             print("%s 죽네" % self.fullname)
```

```
>>> pey = HousePark("응용")
>>> del pey
박응용 죽네
```

Class – Inheritance (상속)

- Inheritance(상속)은 기존 class의 작동 방식을 특수화하거나 변경하기 위해 새로운 class를 만드는 매커니즘임

Ex)
- 부모로부터 Audi r8 쿠페를 상속받으면, Audi r8 쿠페는 이제 내꺼.
- 이 차를 그대로 타도 되고, 색상을 변경하거나 타이어를 갈아낄 수도 있다.

- B class가 A class를 **상속** 받음
- 이런 상황에서 B class는 **자식**, A class는 **부모**라고 한다.
- 이렇게 되면 B class는 A class의 속성을 상속받게 되어 A class의 기능을 마음껏 사용할 수 있게 된다.
- 또한 A class의 속성을 다시 재정의해서 사용할 수도 있다 (소유권이 넘어오기 때문임)

Class – 상속: 예제

```
# -*- coding: cp949 -*-
class HouseKim:
    lastname = "김"
    fullname = ""
    def __init__(self, name):
        self.fullname = self.lastname + name
    def travel(self, where):
        print "%s, %s 여행을 가다." % (self.fullname, where)
```

```
>>> import inheri_ex
>>> kim = HouseKim("대리")
>>> kim.travel("해외")
김대리, 해외 여행을 가다.
>>>
>>> class HousePark(HouseKim):
        lastname = "박"


>>> park = HousePark("천음")
>>> park.travel("독도")
박천음, 독도 여행을 가다.
```

HouseKim을 상속받아
travel method를 사용하였다.

Class – 상속: 예제 – cont'd

```
# -*- coding: cp949 -*-
class HouseKim:
    lastname = "김"
    fullname = ""
    def __init__(self, name):
        self.fullname = self.lastname + name
    def travel(self, where):
        print "%s, %s 여행을 가다." % (self.fullname, where)
```

```
>>> import inheri_ex
>>> kim = HouseKim("대리")
>>> kim.travel("해외")
김대리, 해외 여행을 가다.
```

HouseKim의 travel method를 overriding 함.

```
>>> class HousePark(HouseKim):
        lastname = "박"
        def travel(self, where, day):
            print "%s, %s 여행 %d일 가네." % (self.fullname, where, day)
```

```
>>> park = HousePark("천음")
>>> park.travel("독도", 3)
박천음, 독도 여행 3일 가네.
```

Class – 연산자 오버로딩

- 연산자 오버로딩이란 연산자(+, –, *, /, , ,)등을 instance 끼리 사용할 수 있게 하는 기법을 말한다

```
>>> pey = HousePark("응용")
>>> julliet = HouseKim("줄리엣")
>>> pey + julliet
박응용, 김줄리엣 결혼했네
```

- 즉, instance끼리 연산자 기호를 사용하는 방법을 말함

Class – 연산자 오버로딩 – cont'd

```
# -*- coding: cp949 -*-
class HousePark:
    lastname = "박"

    def __init__(self, name):
        self.fullname = self.lastname + name
    def travel(self, where):
        print "%s, %s 여행을 가다." % (self.fullname, where)
    def love(self, other):
        print "%s, %s 사랑에 빠졌네" % (self.fullname, other.fullname)
    def __add__(self, other):
        print "%s, %s 결혼했네" % (self.fullname, other.fullname)
    def __del__(self):
        print "%s 죽네" % self.fullname


class HouseKim(HousePark):
    lastname = "김"
    def travel(self, where, day):
        print "%s, %s여행 %d일 가네." % (self.fullname, where, day)

pey = HousePark("응용")
julliet = HouseKim("줄리엣")
pey.love(julliet)
pey + julliet
```

```
C:\Python27\course>oper_over_ex.py
박응용, 김줄리엣 사랑에 빠졌네
박응용, 김줄리엣 결혼했네
김줄리엣 죽네
박응용 죽네
```

모듈(Module)

모듈이란?

- 함수나 변수들, 또는 클래스들을 모아놓은 파일
- 다른 python 프로그램에서 import하여 쓸 수 있게 만들어진 파일

Ex)

```
import sys
import math
from optparse import OptionParser
```

Module – 모듈 만들고 불러보기

Module 생성

```
#mod1.py

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

Module 실행

```
>>> import mod1
>>> print mod1.sum(3, 4)
7
```

OR

```
>>> from mod1 import sum
>>> sum(3, 4)
7
```

import와 from x import의 차이

Module – __main__

if __name__ == "__main__": 의 의미

- 직접 이 파일을 실행시켰을 때는 __name__ == "__main__" 이 참이 되어 if문 다음 문장들이 수행 됨
- 대화형 인터프리터나 다른 파일에서 이 모듈을 불러서 쓸 때는 __name__ == "__main__"이 거짓이 되어 if문 아래문장들이 수행되지 않도록 한다는 뜻

Module – __main__ - cont'd

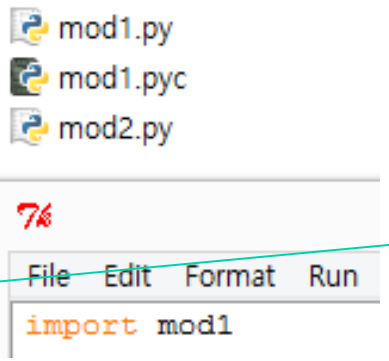
```
# -*- coding: cp949 -*-
#mod1.py

def sum(a, b):
    return a + b

def safe_sum(a, b):
    if type(a) != type(b):
        print "더할 수 있는 것이 아닙니다."
        return
    else:
        result = sum(a, b)
    return result

if __name__ == "__main__":
    print safe_sum('a', 1)
    print safe_sum(1, 4)
    print sum(10, 10.4)
```

__main__에 대한 조건

__main__으로 실행될 때

```
C:\Python27\course>mod1.py
더할 수 있는 것이 아닙니다.
None
5
20.4
```

__main__이 아닐 때



실행

```
>>> ========== RESTART ==========
>>>
>>>
```

Module – class나 변수 등을 포함한 모듈

```
# mod2.py
PI = 3.141592

class Math:
    def solv(self, r):
        return PI * (r ** 2)


def sum(a, b):
    return a+b

if __name__ == "__main__":
    print(PI)
    a = Math()
    print(a.solv(2))
    print(sum(PI , 4.4))
```

__main__으로 실행될 때

```
C:\Python27\course\ex>mod2.py
3.141592
12.566368
7.541592
```

import하여 실행

```
>>> import mod2
>>>
>>> print(mod2.PI)
3.141592
>>> a = mod2.Math()
>>> print a.solv(2)
12.566368
>>> print mod2.sum(mod2.PI, 4.4)
7.541592
```

Module – module을 불러오는 또 다른 방법

우리는 지금껏 만든 모듈을 써먹기 위해서 도스창을 열고 모듈이 있는 디렉토리로 이동한 다음에나 쓸 수 있었음.
하지만 항상 이렇게 해야 하는 불편함을 해소할 수 있는 방법이 있다. → Directory 이용!!

- sys.path로 python library들이 설치 돼 있는 directory 확인

```
>>> import sys
>>> sys.path
['C:\\Python27\\course\\ex', 'C:\\Python27\\Lib\\idlelib', 'C:\\Python27\\lib\\s
ite-packages\\setuptools-0.6c11-py2.7.egg', 'C:\\Python27\\lib\\site-packages\\g
oogle_api_python_client-1.2-py2.7.egg', 'C:\\Python27\\lib\\site-packages\\httpl
ib2-0.8-py2.7.egg', 'C:\\Windows\\SYSTEM32\\python27.zip', 'C:\\Python27\\DLLs',
 'C:\\Python27\\lib', 'C:\\Python27\\lib\\plat-win', 'C:\\Python27\\lib\\lib-tk'
, 'C:\\Python27', 'C:\\Python27\\lib\\site-packages']
```

위의 directory 내에서는 directory 추가 없이 python을 실행할 수 있다.