파이썬 프로그래밍

클래스와 객체

한국기술교육대학교
온라인평생교육원

▣ 파이썬 클래스와 이름공간

1. 파이썬 클래스 정의

- 파이썬 클래스는 새로운 이름 공간을 지원하는 또 다른 단위
- 클래스 정의 구문

```
class 클래스 이름: #헤더(Header)
        pass #몸체(Body)
```

- 인스턴스: 클래스로 부터 만들어낸 객체
- 모듈 vs. 클래스 vs. 인스턴스
  - 모듈: 파일 단위로 이름 공간을 구성
  - 클래스: 클래스 영역 내에 이름 공간을 구성
  - 인스턴스: 인스턴스 영역 내에 이름 공간을 구성

- 대표적 이름공간 → 모듈
- 클래스도 전체적인 프로그램을 모듈화 (독립적인 구성요소로 활용)
- 클래스 → 자료형 제공
- 헤더(Heather) → 앞줄(Class 클래스이름)에 대한 해설
- pass는 정의한 클래스에 아무 내용이 없을 시 사용
- 클래스 내용에 멤버, 메소드 정의
- 위에 콜론(:)이 있으니 들여쓰기 해야 함
- 클래스가 공장, 붕어빵 틀이면, 인스턴스는 공장물품, 붕어빵
- 모듈, 클래스, 인스턴스 → 이름공간을 제공해주는 대표적인 단위
- 모듈 = .py로 끝나는 확장자 파일

▣ 파이썬 클래스와 이름공간

1. 파이썬 클래스 정의

```
class S1:
    a = 1

print S1.a
print

S1.b = 2 # 클래스 이름 공간에 새로운 이름의 생성
print S1.b
print

print dir(S1) # S1에 포함된 이름들을 리스트로 반환
del S1.b # 이름 공간 S1에서 b삭제
print dir(S1)
```

```
1

2

['__doc__', '__module__', 'a', 'b']
['__doc__', '__module__', 'a']
```

- a 변수는 S1 클래스의 이름공간에 존재하는 변수
- 클래스 정의 바깥에서 새로운 내용 정의
- b 변수는 S1 클래스의 이름공간에 존재하는 변수
- dir(식별자) → 식별자가 가지고 오는 이름공간 안 함수들 알려줌
- '__doc__', '__module__' → 클래스를 만들면 항상 생기는 것
- '__doc__' : 문서 문자열
- '__module__' : 클래스가 정의되고 있는 모듈의 이름
- del S1.b → S1 이름공간에 있는 b를 삭제

▣ 파이썬 클래스와 이름공간

1. 파이썬 클래스 정의

- 파이썬에서는 동적으로 인스턴스 외부에서 인스턴스 멤버를 추가할 수 있음
  - 파이썬에서는 클래스와 독립적으로 각 인스턴스를 하나의 이름 공간으로 취급함

```
x = S1() # x는 S1의 클래스 인스턴스
print x.a

x.a = 10 # 클래스 인스턴스 x의 이름 공간에 이름 생성
print x.a

print S1.a # 클래스 이름 공간과 클래스 인스턴스의 이름공간은 다르다
```

```
1
10
1
```

- 인스턴스는 독립적인 이름 공간
- 클래스 뒤 함수 호출식 사용 → 클래스가 내부적 생성자 호출
- S1( ) → S1의 생성자 호출되면서 클래스의 인스턴스 생성
- S1은 붕어빵 틀, x는 붕어빵
- x.a에서 x가 a를 가지고 있는 것이 아님
- x 안에 있는 클래스의 a를 출력하는 것
- x가 독립적인 이름공간에 존재하기 때문에 그 이름공간에 a 삽입
- S1 클래스가 가지고 있는 a와 인스턴스 a는 다름

▣ 파이썬 클래스와 이름공간

1. 파이썬 클래스 정의

```
y = S1() # S1 클래스의 또 다른 인스턴스 생성

y.a = 300 # 클래스 인스턴스 y의 이름 공간에 이름 생성

print y.a
print x.a # x 인스턴스 공간의 이름 a 확인
print S1.a # 클래스 이름 공간의 a 확인
```

```
300
10
1
```

- y.a → y 이름공간에 a 삽입
- 클래스 S1의 이름공간, 인스턴스 소문자 x,y는 별개의 이름공간 가짐

```
class Simple:
    pass

s1 = Simple()
s2 = Simple()
```

- simple은 바디쪽 내용이 없음 (pass)

▣ 파이썬 클래스와 이름공간

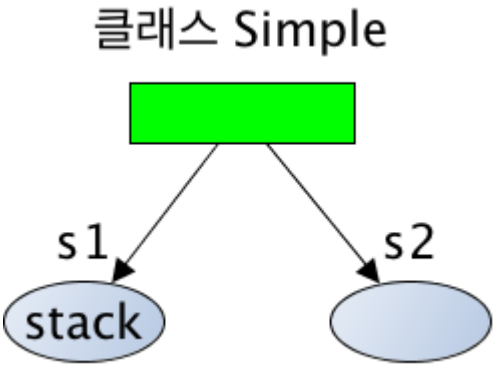
1. 파이썬 클래스 정의

```
s1.stack = [] # 동적으로 클래스 인스턴스 이름 공간 안에 새로운 변수(이름) stack 생성
s1.stack.append(1) # 값 추가
s1.stack.append(2)
s1.stack.append(3)

print s1.stack
print s1.stack.pop()
print s1.stack.pop()
print
print s1.stack # 최종 s1.stack값
print s2.stack # s2에는 stack을 정의한 적이 없다.
```

```
[1, 2, 3]
3
2

[1]


---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
<ipython-input-5-bf2593ac4e2c> in <module>()
      9 print
     10 print s1.stack # 최종 s1.stack값
---> 11 print s2.stack # s2에는 stack을 정의한 적이 없다.

AttributeError: Simple instance has no attribute 'stack'
```

▣ 파이썬 클래스와 이름공간

1. 파이썬 클래스 정의




```
del s1.stack  # s1에서 stack삭제
```

파이썬 프로그래밍

클래스와 객체

한국기술교육대학교
온라인평생교육원

▣ 메소드의 정의와 호출

1. 일반 메소드의 정의와 호출

• 클래스 내부에 메소드 선언 - def 키워드 사용
- 일반 함수와 다른 점은 첫번째 인수로 self 사용 (self라는 이름은 관례적)
  ° self: 인스턴스 객체 자신의 레퍼런스를 지니고 있음
  ° 각 인스턴스들은 self를 이용하여 자신의 이름 공간에 접근

```
class MyClass:
    def set(self, v):
        self.value = v
    def get(self):
        return self.value
```

- 클래스 안 메소듸 정의 시 def 키워드 활용
- 클래스 안에 존재하는 함수들 → 클래스의 메소드
- 클래스의 메소드는 첫 번째 인자에 self 넣음 → 인스턴스에 불려짐
- 첫 번째 인자에 self 들어간 것 → 인스턴스 메소드

▣ 메소드의 정의와 호출

1. 일반 메소드의 정의와 호출

- 인스턴스 객체를 통하여 메소드를 호출할 때 self 인자는 없다고 생각

```
c = MyClass() # 인스턴스 생성
c.set('egg')  # 메소드 set 호출
print c.get() # 메소드 get 호출
print c.value # 인스턴스 변수에 직접 접근
```

```
egg
egg
```

- c.set( ) → 인스턴스 c의 set 메소드 호출 → set은 인스턴스 메소드
- set이 인스턴스 메소드인 이유 → 첫 번째 인자가 self
- c 객체의 레퍼런스는 self로 copy됨 → c가 self가 됨
- set 정의 시 인자가 2개지만, 호출할 때는 1개만 줌
- c 인스턴스가 value 변수를 가짐
- c 인스턴스의 이름 공간에 value 생성 → value를 c 이름공간에서 호출

▣ 메소드의 정의와 호출

1. 일반 메소드의 정의와 호출

- 위 코드는 실제로 아래 코드와 동일함

```
c = MyClass() # 인스턴스 생성
MyClass.set(c, 'egg')
print MyClass.get(c)
print c.value
```

```
egg
egg
```

- self 자리에는 인스턴스가 들어가야 함
- 클래스 이름을 통해서 메소드 접근 → 직접 인스턴스 이름 삽입 필요

▣ 메소드의 정의와 호출

1. 일반 메소드의 정의와 호출

```
class Simple:
    pass


c = MyClass()
s1 = Simple()
MyClass.set(s1, 'egg')  # 다른 클래스의 인스턴스를 넣어주면 에러 발생
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-14-0660940f4507> in <module>()
      4 c = MyClass()
      5 s1 = Simple()
----> 6 MyClass.set(s1, 'egg')  # 다른 클래스의 인스턴스를 넣어주면 에러 발생

TypeError: unbound method set() must be called with MyClass instance as first
argument (got Simple instance instead)
```

- simple을 통해서도 인스턴스 생성이 가능
- self 자리에는 반드시 myclass 인스턴스만 올 수 있음
- c.set() → 이 순간의 set은 bound 메소드(특정 개체에 묶여 있음)
- MyClass.set() → 클래스 이름 통한 것은 unbound 메소드
- 특정 개체에 묶이지 않은 것
- class.메소드 호출 → unbound 메소드

```
• 메소드 호출 종류
 - Unbound method call: 클래스 객체를 이용한 메소드 호출
    * 예: MyClass.set(c, 'egg')
 - Bound method call: 인스턴스 객체를 통한 메소드 호출
    (self 인자는 호출받은 객체가 자동으로 할당)
    * 예: c.set('egg')
```

▣ 메소드의 정의와 호출

2. 클래스 내부에서의 메소드 호출

```
class MyClass:
    def set(self, v):
        self.value = v
    def incr(self):
        self.set(self.value + 1) # 내부 메소드 호출
    def get(self):
        return self.value

c = MyClass()
c.set(1)
print c.get()
print

c.incr()
print c.get()
```

```
1

2
```

- incr 메소드 내 다른 메소드를 호출하는 예제
- self 가 가지고 있는 value 값 + 1을 v에 넣음
- c 인스턴스가 self에 들어가고 self 자리엔 c가 존재
- incr() → increment의 약자

▣ 메소드의 정의와 호출

2. 클래스 내부에서의 메소드 호출

- 만약 위 코드에서 self.set(self.value + 1)를 set(self.value + 1)으로 바꾸면 set 함수를 정적 영역에서 찾는다.

```
def set(i):
    print "set function outside function - ", i

class MyClass:
    def set(self, v):
        self.value = v
    def incr(self):
        set(self.value + 1)     # 정적 영역에 존재하는 set 메소드 호출
    def get(self):
        return self.value

c = MyClass()
c.set(1)
print c.get()

print

c.incr()
print c.get()
```

```
1

set function outside function -  2
1
```

- set 앞에 self가 없으면 set 함수를 class 바깥에서 찾음
- 메소드 내 다른 메소드, 인스턴스 호출 시 반드시 self 사용

▣ 메소드의 정의와 호출

3. 정적 메소드

- 정적 메소드: 인스턴스 객체와 무관하게 클래스 이름 공간에 존재하는 메소드로서 클래스 이름을 이용하여 직접 호출할 수 있는 메소드
  - [주의] 해당 클래스의 인스턴스를 통해서도 호출 가능
- 장식자(Decorator) @staticmethod 활용

```
class D:
    @staticmethod
    def spam(x, y):            # self가 없다.
        print 'static method', x, y

D.spam(1,2) # 인스턴스 객체 없이 클래스에서 직접 호출

print
d = D()
d.spam(1,2) # 인스턴스 객체를 통해서도 호출 가능
```

```
static method 1 2

static method 1 2
```

- 클래스 이름공간에 존재하는 메소드 → 인스턴스 메소드도 마찬가지
- 클래스 내 메소드는 인스턴스를 통해 호출 & self 사용
- → 인스턴스 메소드
- 정적 메소드는 첫 번째 인자에 self 사용하지 않음
- 인스턴스를 통해서도 호출이 가능함
- 하지만, 인스턴스 이름공간에 작업 X → 클래스 이름공간 변수 조작
- spam 메소드 → 위에 장식자가 있으므로 static 메소드
- 인스턴스를 통해 호출 X → 첫 번째 인자에 self 필요 없음
- 인스턴스에 static 메소드 호출 가능

▣ 메소드의 정의와 호출

4. 클래스 메소드

- 클래스 메소드: 인스턴스 객체와 무관하게 클래스 이름 공간에 존재하는 메소드로서 클래스 이름을 이용하여 호출하며 첫 인수로 클래스 객체를 자동으로 받는 메소드
  - [주의] 해당 클래스의 인스턴스를 통해서도 호출 가능
- 장식자(Decorator) @classmethod 활용

```
class C:
    @classmethod
    def spam(cls, y):
        print cls, '->', y

print C

print
C.spam(5) # 첫번째 인수로 C가 잠재적으로 전달된다.

print
c = C()
c.spam(5) # 인스턴스 객체를 통해서도 호출 가능.
```

```
__main__.C

__main__.C -> 5

__main__.C -> 5
```

- 공통점 = 인스턴스 객체와 무관하게 클래스 이름 공간 존재
- 공통점 2 = 클래스 이름을 통해서 호출
- 클래스 메소드만 첫 번째 인자에 클래스 객체 자동으로 사용
- spam(cls, y) → C. spam(5) : y는 5, cls는 C 클래스 자동 삽입

▣ 메소드의 정의와 호출

4. 클래스 메소드

- 상속받은 서브 클래스를 통해 호출하면, 첫 인수에는 서브 클래스 객체가 자동으로 할당됨

```
class D(C):
    pass

print D.spam(3)

d = D()
print d.spam(3)
```

```
__main__.D -> 3
None
__main__.D -> 3
None
```

- D(C): → 클래스 D는 클래스 C를 상속 받음
- 상속 받으면 클래스 C가 가지고 있는 메소드 모두 가질 수 있음
- d라는 인스턴스를 통해서도 spam 호출 가능

파이썬 프로그래밍

클래스와 객체

한국기술교육대학교
온라인평생교육원

▣ 클래스 멤버와 인스턴스 멤버

1. 클래스 멤버와 인스턴스 멤버

• 클래스 멤버 vs. 인스턴스 멤버
- 클래스 멤버
  * 클래스 이름 공간에 생성됨
  * 모든 인스턴스들에 의해 공유됨
- 인스턴스 멤버
  * 인스턴스 이름 공간에 생성됨
  * 각각의 인스턴스 마다 독립성이 보장됨

```
class Var:
    c_mem = 100 # 클래스 멤버 정의
    def f(self):
        self.i_mem = 200 # 인스턴스 멤버 정의
    def g(self):
        print self.i_mem
        print self.c_mem
```

- c_mem → 클래스 멤버
- self.i_mem → 인스턴스 멤버
- 클래스 멤버는 각각의 내용을 공유하므로 self 앞에 넣는 코딩 가능

▣ 클래스 멤버와 인스턴스 멤버

1. 클래스 멤버와 인스턴스 멤버

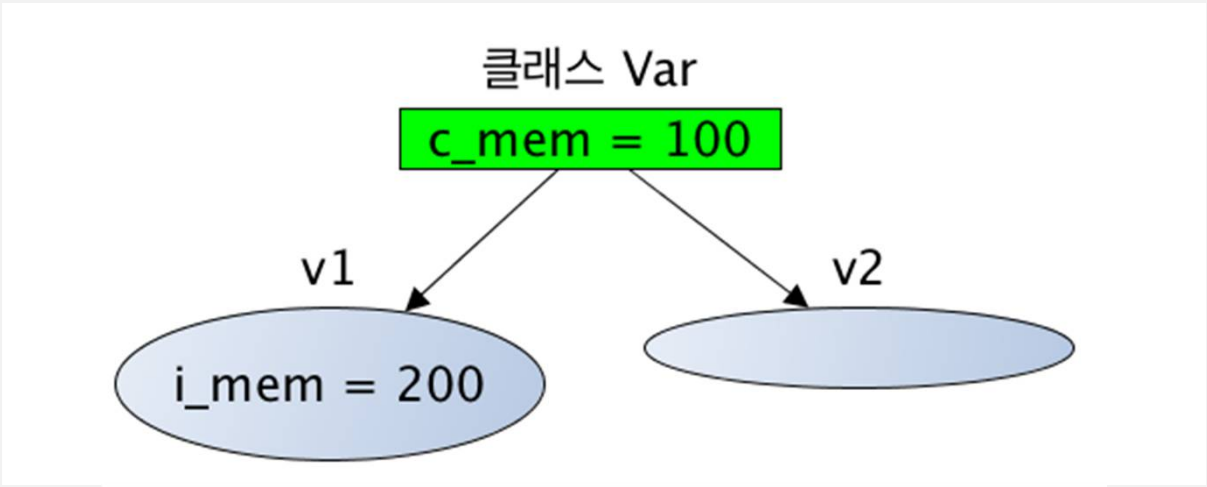
```
print Var.c_mem  # 클래스 객체를 통하여 클래스 멤버 접근

v1 = Var()       # 인스턴스 v1 생성
print v1.c_mem   # 인스턴스를 통하여 클래스 멤버 접근
v1.f()           # 인스턴스 멤버 i_mem이 생성됨
print v1.i_mem   # 인스턴스 v1을 통하여 인스턴스 멤버 접근

print
v2 = Var()       # 인스턴스 v2 생성
print v2.i_mem   # 인스턴스 v2에는 아직 f() 호출이 안되어서 i_mem 멤버 없음
                    ==> 생성자의 필요성
```

```
100
100
200

---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
<ipython-input-59-1950cf0552a9> in <module>()
      8 print
      9 v2 = Var()        # 인스턴스 v2 생성
---> 10 print v2.i_mem    # 인스턴스 v2에는 아직 f() 호출이 안되어서 i_mem 멤버
                             없음 ==> 생성자의 필요성

AttributeError: Var instance has no attribute 'i_mem'
```

- v2 = Var 클래스의 인스턴스
- v2는 아직 f함수로 호출이 안됨 → i_mem 존재하지 않음

▣ 클래스 멤버와 인스턴스 멤버

1. 클래스 멤버와 인스턴스 멤버



- "인스턴스 이름.멤버 이름"으로 멤버를 참조할 때 멤버의 검색 순서
  1) 인스턴스 멤버
  2) 인스턴스가 없다면 클래스 멤버

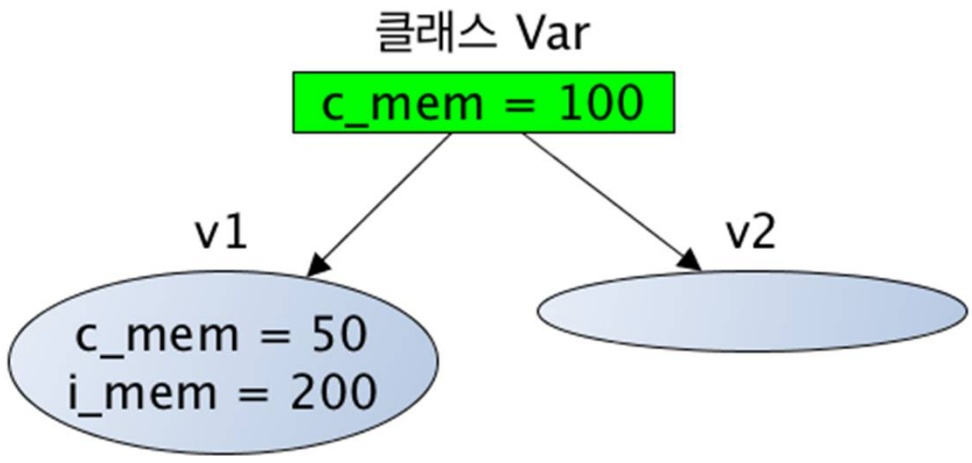
- 클래스는 1개, 객체는 생성자를 통해 독립적으로 만들어짐

▣ 클래스 멤버와 인스턴스 멤버

1. 클래스 멤버와 인스턴스 멤버

```
print v1.c_mem   # 인스턴스 v1을 통해 클래스 멤버 참조
print v2.c_mem   # 인스턴스 v2를 통해 클래스 멤버 참조

print
v1.c_mem = 50    # 인스턴스 이름 공간에 c_mem생성
print v1.c_mem   # 인스턴스 v1을 통해 인스턴스 멤버 참조
print v2.c_mem   # 인스턴스 v2을 통해 클래스 멤버참조 (인스턴스 멤버가 없으므로,
                   클래스 멤버 참조)
print Var.c_mem  # 클래스 멤버참조
```

```
100
100

50
100
100
```



- v1 이름공간에서 c_mem을 찾음
- 없으면 위 클래스로 올라가서 c_mem 참조함
- v1.c_mem → v1이 c_mem을 가지고 있으므로 값 출력됨

파이썬 프로그래밍

클래스와 객체

한국기술교육대학교
온라인평생교육원

▣ 생성자와 소멸자

1. 생성자와 소멸자의 메소드

• __init__: 생성자 메소드
객체가 생성될 때 자동으로 불리어지는 메소드
self 인자가 정의되어야 함
__del__: 소멸자 메소드
객체가 소멸 (메모리에서 해제)될 때 자동으로 불리어지는 메소드
self 인자가 정의되어야 함
개발자가 특별히 작성하지 않아도 될 메소드
이유: 파이썬에서는 메모리나 기타 자원들의 해제가 자동으로 되기 때문에
[참고] __ (연속된 두 개의 언더라인)의 의미: 예약된 이름

다음 코드에 대한 설명
mylife = Life() 로서 인스턴스 mylife가 생성되는 순간 __init__ 생성자 메소드 호출
sleep(3)에 의해 3초간 sleep 상태
3초 이후 함수가 리턴됨 --> 로컬 변수가 메모리에서 해제됨
--> __del__ 소멸자 메소드 호출

- 생성자 메소드는 많이 정의가 됨
- '__'로 시작 → 예약된 이름들 (직접 다시 정의할 필요 X)

▣ 생성자와 소멸자

1. 생성자와 소멸자의 메소드

```
# _*_ coding:utf-8 _*_
from time import ctime, sleep

class Life:
    def __init__(self):             # 생성자
        self.birth = ctime()        # 현재시간에 대한 문자열을 얻는다.
        print 'Birthday', self.birth # 현재 시간 출력
    def __del__(self):              # 소멸자
        print 'Deathday', ctime()    # 소멸 시간 출력

def test():
    mylife = Life()
    print 'Sleeping for 3 sec'
    sleep(3) #3초간 sleep(block)상태에 있음 (CPU 점유 못함)

test()
```

```
Birthday Fri Dec  5 11:06:36 2014
Sleeping for 3 sec
Deathday Fri Dec  5 11:06:39 2014
```

- ctime의 c는 current → 현재시간
- 생성자는 self 부분에 생성하고자 하는 객체 들어감
- 리턴이 없어도 객체가 생성되어 mylife에 할당됨
- 소멸자가 호출되어지는 코드가 명시적으로 존재하지 X
- 함수가 끝나면, 함수 안에 정의되어진 객체는 자동으로 없어짐
- 소멸되기 직전에 del이라는 함수가 수행되는 것
- 프로그램이 끝난 것을 확인하기 전까지 계속 수행

▣ 생성자와 소멸자

1. 생성자와 소멸자의 메소드

- 인자를 받는 생성자 호출 가능
- [참고] __str__: print 예약어나 str() 내장함수 호출에 대응되는 메소드

```
class Integer:
    def __init__(self, i):
        self.i = i
    def __str__(self):
        return str(self.i)

i = Integer(10)
print i
print str(i)
```

```
10
10
```

- 생성자를 정의할 때 인자를 self말고 인자 하나 더 삽입
- '__str__' 메소드 → 프린트 i, 내장함수 str() 사용 시 str 메소드 자동 호출