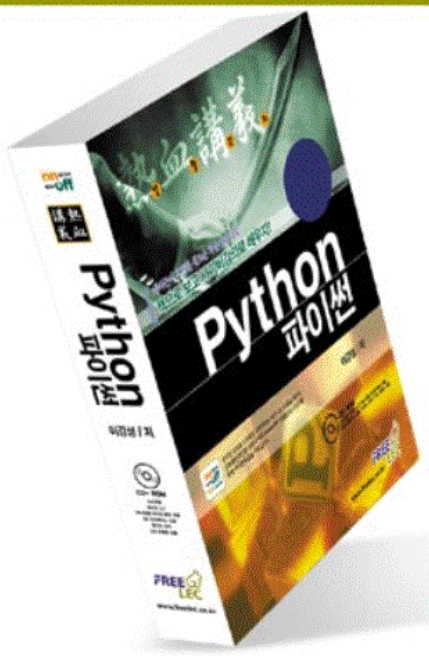
熱血講義

프리렉의 열혈강의 시리즈

Python 파이썬



열혈강의 Python 파이썬

❖제 12 강

주제 : 클래스

이강성 (gslee@mail.gwu.ac.kr)

❖강의 목차

1. 파이썬 클래스
2. 정의 및 호출
3. 클래스 멤버, 인스턴스 멤버
4. 생성자 소멸자
5. 연산자 중복
6. 특별 속성들
7. 상속
8. 협동적 메쏘드와 super()
9. 내장 자료형과 클래스의 통일
10. 다형성등..

12-1 클래스란?

- 파이썬 클래스?
  - 하나의 이름 공간
  - 함수(메쏘드)와 변수(멤버)를 갖는다
  - 상속(inheritance)이 된다
  - 연산자 중복 지원
  - ...

12-1 클래스란?

- 용어 정리
  - 클래스(객체) : class 문으로 정의되며, 멤버와 메쏘드를 갖는 객체이다.
  - 클래스 인스턴스 (객체) : 클래스를 호출하여 만들어지는 객체
  - 멤버(member) : 클래스가 갖는 변수
  - 메쏘드(method) : 클래스 내에 정의된 함수
  - 속성(attribute) : 멤버와 메쏘드
  - 수퍼 클래스(superclass, baseclass) : 어떤 클래스의 상위에 있으며 각종 속성을 서브클래스로 상속해준다.
  - 서브 클래스(subclass, derived class) : 수퍼 클래스로부터 상속된 하위의 클래스를 말한다. 수퍼 클래스로부터 각종 속성을 상속 받으므로 코드와 변수를 공유한다.

12-2 클래스 정의

- 간단한 클래스 정의

```
class Simple:    # 헤더(header)

    pass         # 몸체(body)
```

- 인스턴스 객체의 생성

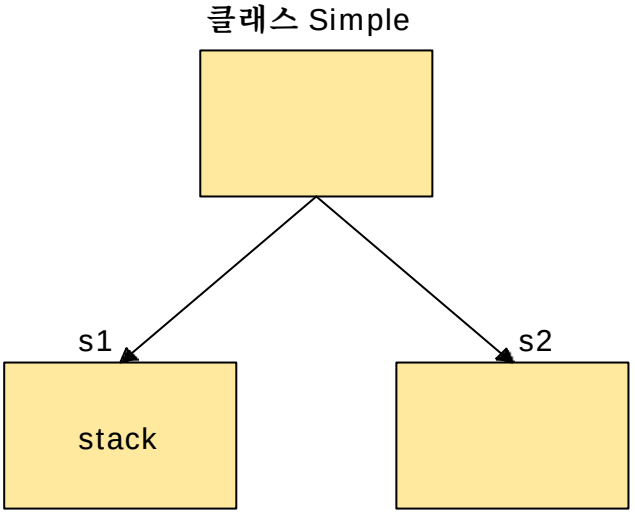
```
s1 = Simple()

s2 = Simple()
```

- 클래스와 인스턴스 객체는 각각 독립적인 이름 공간을 갖는다.

```
>>> s1.stack = []
```

12-2 클래스 정의



클래스 객체의 이름 공간과 인스턴스 객체의 이름 공간

12-3 메쏘드 정의와 호출

- 메쏘드 정의하기
  - 일반 함수를 정의하는 것과 동일하다.
  - 첫 인수로 인스턴스 객체가 온다(self)
  - self는 인스턴스 객체를 가리킨다(C++나 Java의 this와 비슷)

```
class MyClass:
    def set(self, v):
            self.value = v
    def put(self):
            print self.value
```

12-3 메쏘드 정의와 호출

- 메쏘드 호출하기(1)
  - 바운드 인스턴스 메쏘드(bound instance method) 호출

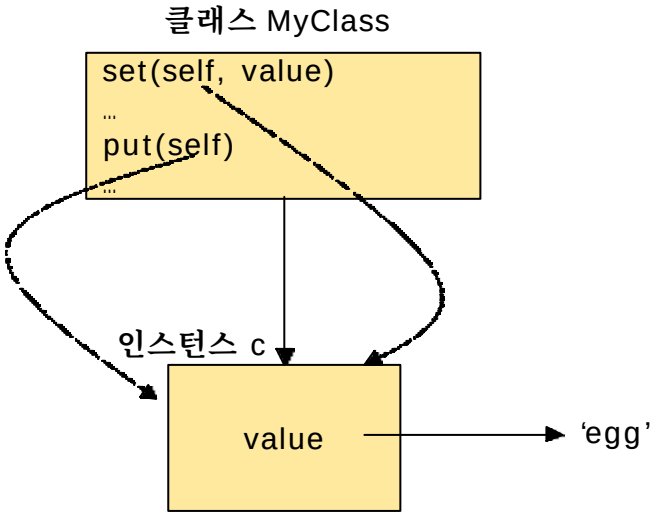
```
>>> c = MyClass()          # 인스턴스 생성
>>> c.set('egg')           # 메쏘드 set 호출
>>> c.put()      # 메쏘드 put 호출
egg
```

- 메쏘드 호출하기(2)
  - 언바운드 클래스 메쏘드(Unbound class method) 호출

```
>>> MyClass.set(c, 'egg')
>>> MyClass.put(c)
egg
```


12-3 메쏘드 정의와 호출

- 메쏘드 호출



12-3 메쏘드 정의와 호출

- 클래스 내부에서의 메쏘드 호출

```
class MyClass2:
    def set(self, v):
        self.value = v
    def incr(self):
        self.set(self.value + 1)
    def put(self):
        print self.value
```


12-3 메쏘드 정의와 호출

- 정적 메쏘드(static method, 2.2)*
  - 인스턴스 객체 없이 클래스에서 직접 호출할 수 있는 메쏘드

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

>>> D.spam(1,2)      # 클래스에서 직접 호출
static method 1 2
>>> d = D()
>>> d.spam(1,2)
static method 1 2 # 인스턴스를 통하여 호출
```

12-3 메쏘드 정의와 호출

- 클래스 메쏘드(class method, 2.2) *
  - 첫 인수로 클래스 객체를 받는다

```
>>> class C:
        def spam(cls, y):
            print cls, y
        spam = classmethod(spam)

>>> print C
__main__.C
>>> C.spam(5)   # 첫 인수로 C가 잠정적으로 전달된다.
__main__.C 5
>>> c = C()
>>> c.spam(5)
__main__.C 5
```


12-4 클래스 / 인스턴스 멤버

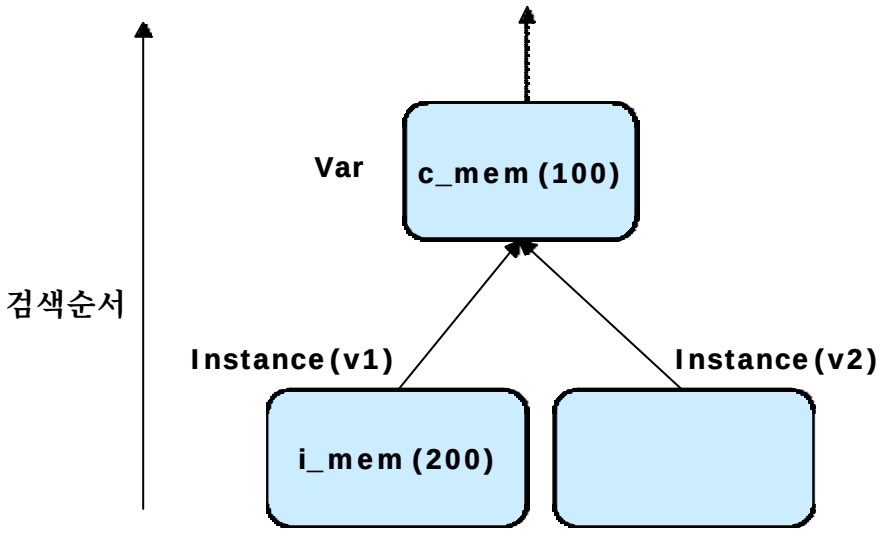
- 클래스 멤버와 인스턴스 멤버
  - 클래스 멤버
    - 클래스 이름 공간에 생성
    - 모든 인스턴스 객체들이 공유
  - 인스턴스 멤버
    - 인스턴스 객체 이름 공간에 생성
    - 인스턴스 객체 내에서만 사용

12-4 클래스 / 인스턴스 멤버

```
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

>>> Var.c_mem  # 클래스 객체를 통하여
100
>>> v1 = Var()
>>> v1.f()
>>> v1.c_mem   # 인스턴스 객체를 통하여
100
>>> v2 = Var()    # 계속..
```


12-4 클래스 / 인스턴스 멤버

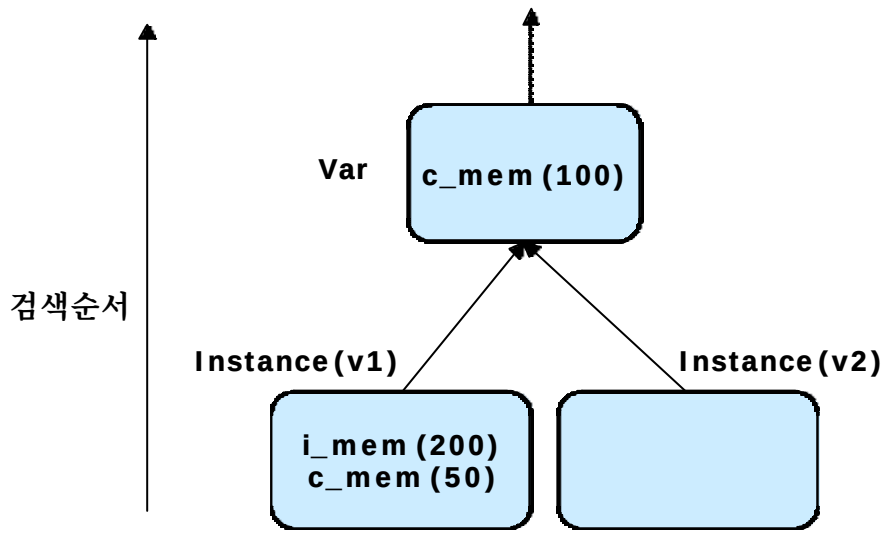


클래스 멤버와 인스턴스 멤버

12-4 클래스 / 인스턴스 멤버

```
>>> v1.c_mem     # 클래스 멤버 참조
100
>>> v2.c_mem     # 클래스 멤버 참조
100
>>> v1.c_mem = 50        # 인스턴스 이름 공간에 c_mem 생성
>>> v1.c_mem     # 인스턴스 멤버 참조* (앞 코드와 비교)
50
>>> v2.c_mem     # 클래스 멤버 참조.
100
>>> Var.c_mem    # 클래스 멤버 참조
100
```


12-4 클래스 / 인스턴스 멤버



12-5 생성자와 소멸자

```
from time import time, ctime, sleep

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

mylife = Life()
print 'Sleeping for 3 sec'
sleep(3)
```

```
Birthday Tue Aug 15 10:49:44 2000
Sleeping for 3 sec
Deathday Tue Aug 15 10:49:47 2000
```


12-6 연산자 중복

```
class MyString:
    def __init__(self, str):
        self.str = str
    def __div__(self, sep): # / 가 사용되었을 때 호출
        return self.str.split(sep)

>>> m = MyString("abcdabcdabcd")
>>> print m / "b"        # 문자열 "b"를 이용해서 리스트로 분리
['a', 'cda', 'cda', 'cd']
>>> print m / "bc"       # 문자열 "bc"를 이용해서 리스트로 분리
['a', 'da', 'da', 'd']
```

12-6 연산자 중복

표 13-1 수치 연산자 메쏘드

메쏘드	연산자
__add__ (self, other)	+
__sub__ (self, other)	-
__mul__ (self, other)	*
__div__ (self, other)	/
__floordiv__(self, other)	// (2.2이상)
__mod__ (self, other)	%
__divmod__ (self, other)	divmod()
__pow__ (self, other[, modulo])	Pow(), **
__lshift__ (self, other)	<<
__rshift__ (self, other)	>>
__and__ (self, other)	&
__xor__ (self, other)	^
__or__ (self, other)	|

12-6 연산자 중복

```
>>> m = MyString("abcdabcdabcd")
>>> m / 'b'  # 여기까지는 잘 됐었다
['a', 'cda', 'cda', 'cd']
>>> 'b' / m  # ???
Traceback (innermost last):
  File "<pyshell#184>", line 1, in ?
    'b' / m
TypeError: __div__ nor __rdiv__ defined for these operands
```

12-6 연산자 중복

표 13-2 피연산자가 바뀐 경우의 수치 연산자 메쏘드

메쏘드	연산자
__radd__ (self, other)	+
__rsub__ (self, other)	-
__rmul__ (self, other)	*
__rdiv__ (self, other)	/
__rfloordiv__ (self, other)	// (2.2이상)
__rmod__ (self, other)	%
__rdivmod__ (self, other)	divmod()
__rpow__ (self, other[, modulo])	pow(), **
__rlshift__ (self, other)	<<
__rrshift__ (self, other)	>>
__rand__ (self, other)	&
__rxor__ (self, other)	^
__ror__ (self, other)	|

12-6 연산자 중복

- __coerce__
  - 다른 두 자료형에 수치 자료형이 적용될 때 두 자료형을 조정해주는 메쏘드

```
>>> class MyNum:
        def __init__(self, n):
            self.n = n
        def __coerce__(self, y):
            return self.n, y
>>> a = MyNum(10)
>>> a + 20
30
>>> a * 30
300
```

12-6 연산자 중복

• 기타 수치 연산자 중복

표 13-3 수치 단항 연산자 메쏘드

메쏘드	연산자
__neg__ (self)	-
__pos__ (self)	+
__abs__ (self)	abs()
__invert__ (self)	~ 비트 반전

표 13-4 수치 형 변환 메쏘드

메쏘드	연산자
__complex__ (self)	complex()
__int__ (self)	int()
__long__ (self)	long()
__float__ (self)	float()

표 13-5 8진수 16진수 변환 메쏘드

메쏘드	연산자
__oct__ (self)	oct()
__hex__ (self)	hex()

12-6 연산자 중복

• 시퀀스 자료형의 연산자 중복

표 13-6 시퀀스 자료형 메쏘드

메쏘드	연산자
__len__ (self)	len()
__contains__ (self, item)	in 연산자 사용시 호출
__getitem__ (self, key)	self[key]
__setitem__ (self, key, value)	self[key] = value
__delitem__ (self, key)	del self[key]

12-6 연산자 중복

```
>>> class Square:
        def __init__(self, end):
            self.end = end
        def __len__(self):
            return self.end
        def __contains__(self, k):
            return 0 <= k < self.end
        def __getitem__(self, k):
            if k < 0 or self.end <= k: raise IndexError, k
            return k * k
```


12-6 연산자 중복

```
>>> s1 = Square(10)
>>> len(s1)  # s1.__len__()
10
>>> 5 in s1  # s1.__contains__(5)  멤버쉽 테스트
1
>>> s1[1]    # s1.__getitem__(1)
1
>>> s1[20]    # 범위를 벗어난 참조
…
if k < self.st or self.end <= k: raise IndexError, k
IndexError: 20
>>> list(s1)
[0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
>>> tuple(s1)
(0, 1, 4, 9, 16, 25, 36, 49, 64, 81)
```

12-6 연산자 중복

- 슬라이싱
  - slice 객체 : slice([start,] stop [, step])

```
>>> slice(10)
slice(None, 10, None)
>>> slice(1, 10)
slice(1, 10, None)
>>> slice(1, 10, 3)
slice(1, 10, 3)
>>> type(s)  # 무슨 형인지 알아본다
<type 'slice'>
>>> dir(s)   # s의 속성 리스트를 보여준다.
['start', 'step', 'stop']
```


12-6 연산자 중복

```
>>> import types
>>> class Square:
    def __init__(self, end):
        self.end = end
    def __len__(self):
        return self.end
    def __getitem__(self, k):
        if type(k) == type(0): # indexing
            if k < 0 or self.end <= k: raise IndexError, k
            return k * k
        elif type(k) == types.SliceType:  # slicing
            start = k.start or 0
            if k.stop > self.end: stop = self.end
            else: stop = k.stop
            step = k.step or 1
            return map(self.__getitem__, range(start, stop, step))
```

12-6 연산자 중복

```
>>> s = Square(10)
>>> s[4]       # 인덱싱
16
>>> s[1:5]     # 슬라이싱
[1, 4, 9, 16]
>>> s[1:10:2]   # 간격은 2로
[1, 9, 25, 49, 81]
>>> s[:]         # 전체 범위
[0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
```


12-6 연산자 중복

- 매핑 자료형의 연산자 중복

표 13-8 매핑 자료형을 위한 메쏘드들

메쏘드	연산자
__len__ (self)	len()
__getitem__ (self, key)	self[key]
__setitem__ (self, key, value)	self[key] = value
__delitem__ (self, key)	del self[key]

12-7 특별한 속성들

- 특별한 메쏘드들
  - __repr__(self)
    - repr() 혹은 ``에 의해 호출
    - 문자열을 리턴한다
  - __str__(self)
    - print 혹은 str() 에 의해 호출
    - 문자열을 리턴한다
  - __cmp__(self, other)
    - 비교 연산(<, >, <=, >=, ==, !=)에 의해 호출
    - -1, 0, 1을 리턴한다.

표 13-9 비교 연산을 위한 메쏘드들 (2.1이상)

연산	메쏘드 이름
<	__lt__
<=	__le__
>	__gt__
>=	__ge__
==	__eq__
!=	__ne__

12-7 특별한 속성들

- 특별한 메쏘드들
  - __hash__(self)
    - 객체 m이 사전의 키로 사용될 때 해쉬 키 값을 얻을 때 호출된다. 32비트 정수를 리턴 해야 한다. __cmp__ 메쏘드도 함께 정의되어 있어야 한다.
  - __nonzero__(self)
    - 객체의 진리 값을 결정한다. 0 또는 1을 리턴
  - __call__ (self[, args...])
    - 호출 가능한 클래스 인스턴스 만들기

12-7 특별한 속성들

- 특별한 메쏘드들
  - __slots__
    - 허용 가능한 인스턴스 변수 목록 만들기 (2.2)
  - get/set 메쏘드로 속성 정의하기 (property)

```
>>> class D(object):
        def __init__(self):
            self.__degree = 0
        def get_degree(self):      # degree를 읽을 때 호출되는 메쏘드
            return self.__degree
        def set_degree(self, d):   # degree에 기록할 때 호출되는 메쏘드
            self.__degree = d % 360
        degree = property(get_degree, set_degree)
```


12-7 특별한 속성들

```
>>> d = D()
>>> d.degree = 10
>>> print d.degree
10
>>> d.degree = 370
>>> print d.degree
10
>>> d.degree = -370
>>> print d.degree
350
```

12-7 특별한 속성들

- 클래스 인스턴스 속성 제어하기

표 13-10 인스턴스 객체 속성을 다루기 위한 메쏘드들

메쏘드 이름	설명
__getattr__(self, name)	정의되어 있지 않은 속성을 참조할 때, 이 메쏘드가 호출된다. 속성 이름 name은 문자열이다
__getattribute__(self, name)	__getattr__ 과 같으나 속성이 정의되어 있어도 호출된다(2.2이상)
__setattr__(self, name, value)	self.name = value 와 같이 속성에 치환(대입)이 일어날 때 호출된다
__delattr__(self, name)	del self.name 시에 호출된다

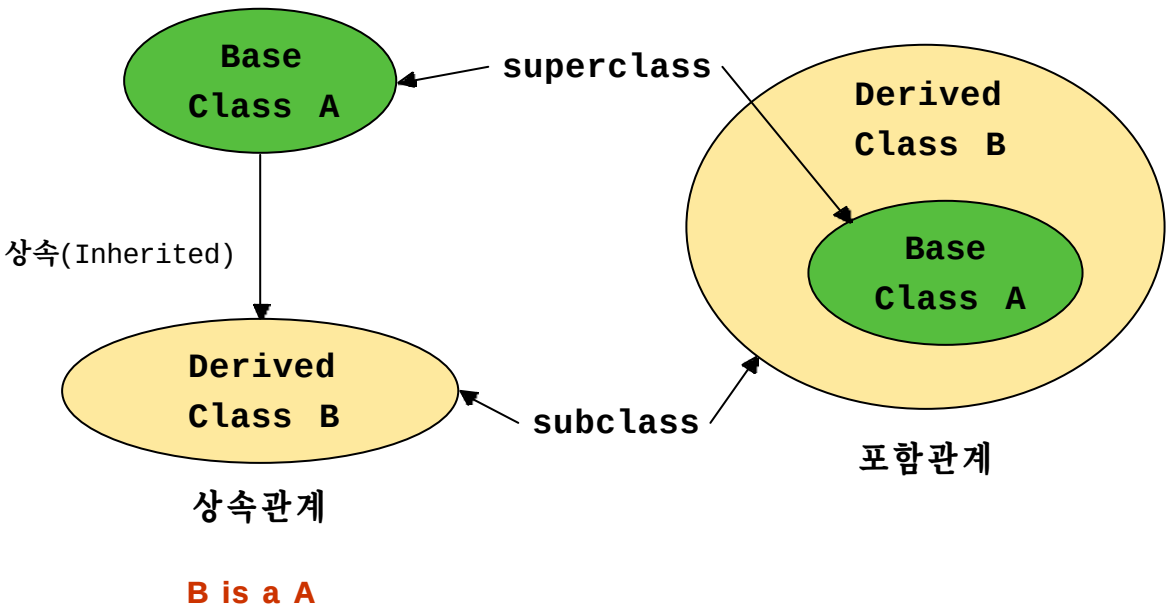
12-8 상속

- 상속(Inheritance) :
  - 기존의 클래스의 속성과 행동을 그대로 받아들이고 추가적으로 필요한 기능을 새로운 클래스에 덧붙이는 것
  - 소프트웨어의 재사용 관점에서 상속은 중요한 역할을 하며, 프로그램의 개발 시간을 단축시켜준다.
  - A 클래스를 수퍼클래스로 하는 클래스 B를 생성되었다면 B "is-a" A 관계라고 한다.

12-8 상속

- 수퍼 클래스(superclass)
  - 베이스 클래스(base class)
  - 보편적으로 일반적인 특징(연산등)을 갖는다
- 서브 클래스(subclass)
  - 파생클래스(derived class)
- 다중 상속 (Multiple Inheritance)
  - 두 개 이상의 수퍼 클래스로부터 상속 받는 것.

12-8 상속



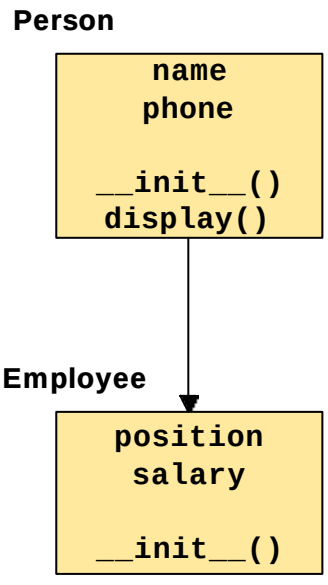
B is a A

12-8 상속

```
class Person:
    def __init__(self, name, phone=None):
        self.name = name
        self.phone = phone
    def display(self):
        return '<Person %s %s>' % (self.name, self.phone)

class Employee(Person):
    def __init__(self, name, phone, position, salary):
        Person.__init__(self, name, phone)
        self.position = position
        self.salary = salary
```


12-8 상속

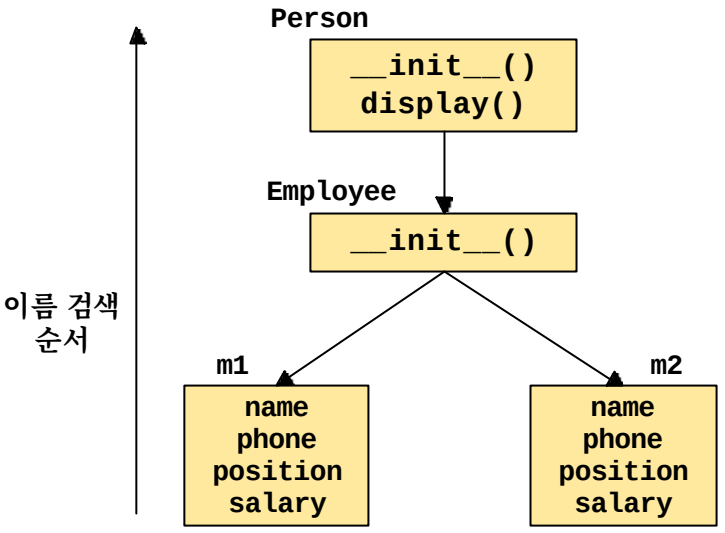


클래스 상속 관계

12-8 상속

```
m1 = Employee('손창희', 5564, '대리', 200)
m2 = Employee('김기동', 8546, '과장', 300)
print m1.name, m1.position
print m2.name, m2.position
```

```
손창희 대리
김기동 과장
```



이름 공간의 관계

12-8 상속

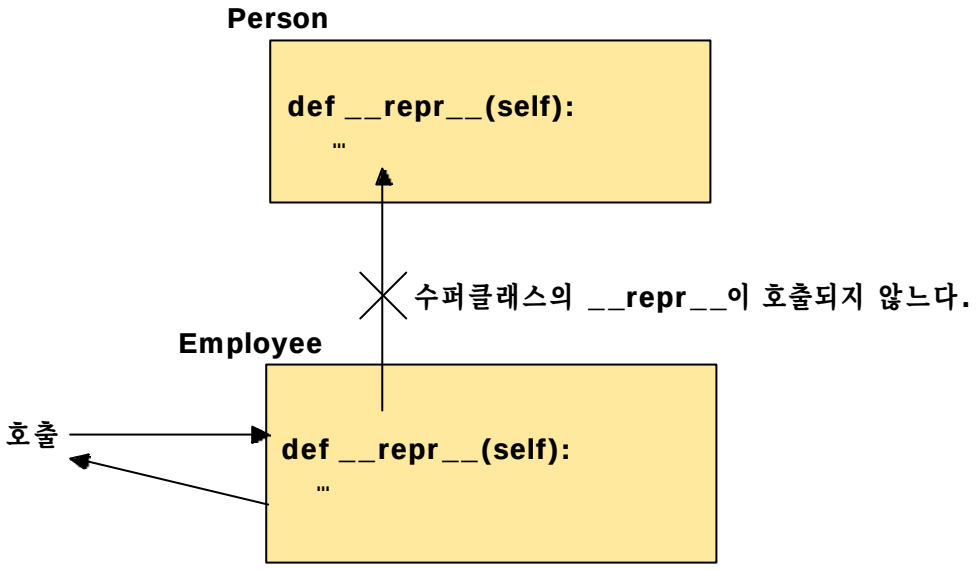
- 메쏘드의 대치

```
class Person:
    ...
    def __repr__(self):
        return '<Person %s %s>' % (self.name, self.phone)


class Employee(Person):
    ...
    def __repr__(self):
        return '<Employee %s %s %s %s>' % (self.name,
self.phone, self.position, self.salary)
```

12-8 상속

- 메쏘드의 대치



12-8 상속

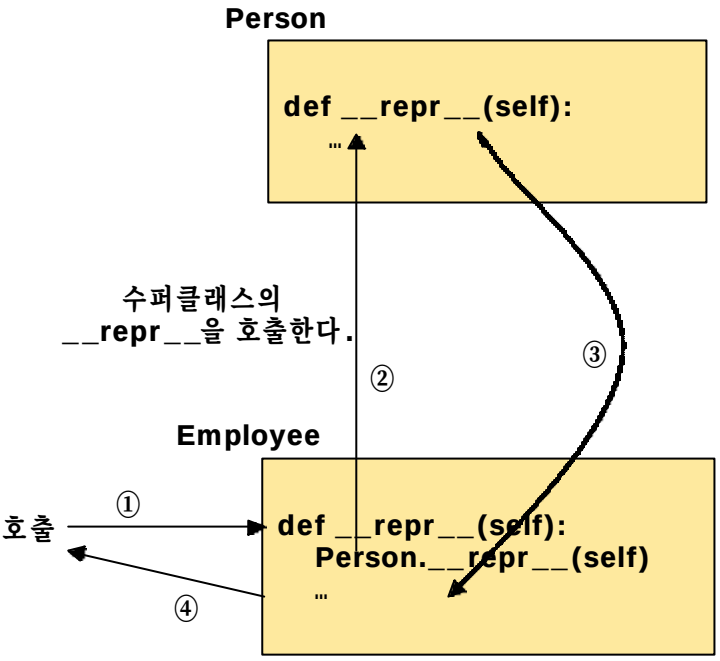
- 메쏘드의 확장

```
class Person:
    def __init__(self, name, phone=None):
        self.name = name
        self.phone = phone
    def __repr__(self):
        return '<Person %s %s>' % (self.name, self.phone)

class Employee(Person):
    def __init__(self, name, phone, position, salary):
        Person.__init__(self, name, phone)
        self.position = position
        self.salary = salary
    def __repr__(self):
        s = Person.__repr__(self)
        return s + ' <Employee %s %s>' % (self.position, self.salary)
```

12-8 상속

- 메쏘드의 확장



12-8 상속

- 상속 클래스의 예

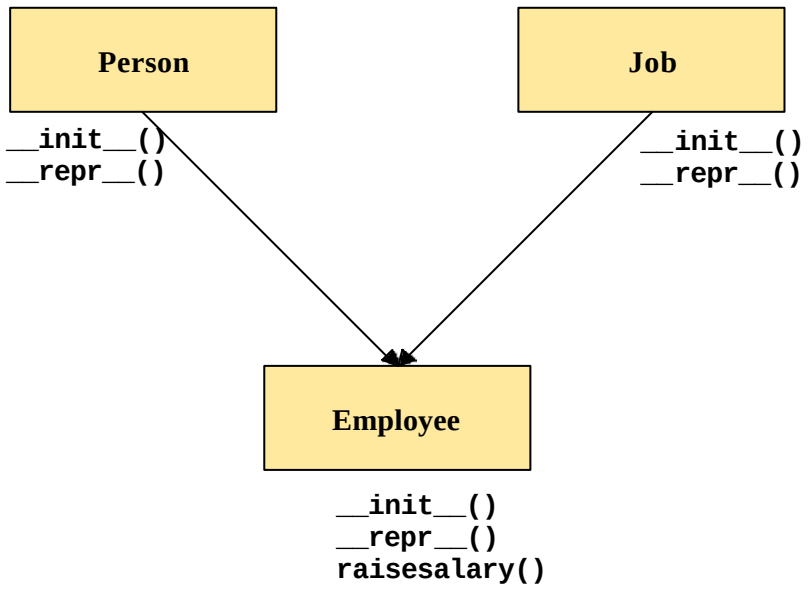
12-8 상속-다중 상속

- 다중 상속
  - 클래스가 두 개 이상의 수퍼클래스로부터 상속 받는 것

```
class Employee(Person, Job):
    def __init__(self, name, phone, position, salary):
        Person.__init__(self, name, phone)
        Job.__init__(self, position, salary)
    def raisesalary(self, rate):
        self.salary = self.salary * rate
    def __repr__(self):
        return Person.__repr__(self) + ' ' + Job.__repr__(self)
e = Employee('gslee', 5244, 'prof', 300)
e.raisesalary(1.5)
print e
```

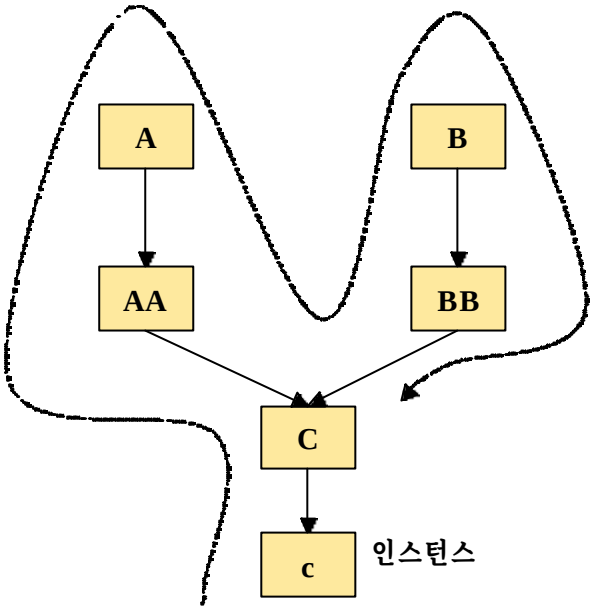

12-8 상속-다중 상속

- 다중 상속



12-8 상속-다중 상속

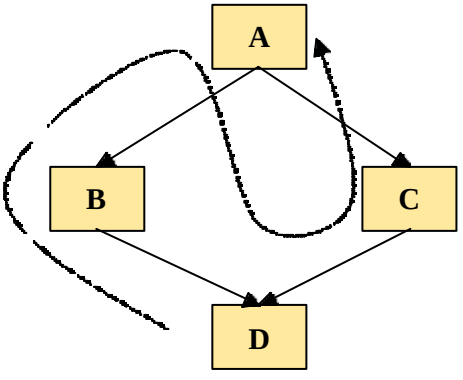
- 메쏘드 처리 순서



12-8 상속-다중 상속

- 다이아몬드형 다중 상속 (1)
- 메쏘드 처리 순서

```
class A:
class B(A):
class C(A):
class D(B, C):
```

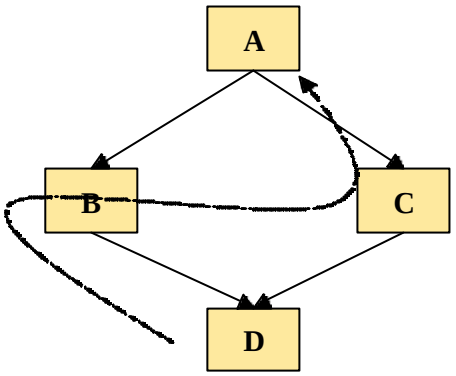


D → B → A → C → A

12-8 상속-다중 상속

- 다이아몬드형 다중 상속 (2)
- 메쏘드 처리 순서(MRO-method resolution order)

```
class A(object):
class B(A):
class C(A):
class D(B, C):
```



D → B → C → A

12-8 상속-가상 함수

- 가상 함수
  - 메쏘드의 호출이 참조되는 클래스 인스턴스에 따라서 동적으로 결정되는 함수 (dynamicbinding).
  - 파이썬 클래스의 모든 메쏘드는 가상 함수

12-8 상속-가상 함수

```
class Base:
    def f(self):
        self.g()                    # 함수 g() 호출
    def g(self):
        print 'Base'
class Derived(Base):
    def g(self):                    # 파생 클래스의 함수 g()
        print 'Derived'

b = Base()
b.f()               # Base의 g() 호출
a = Derived()
a.f()               # Derived의 g() 호출
```

```
Base
Derived
```


12-8 상속의 예

- 명령어 해석기의 설계
  - cmd.Cmd 부클래스를 하나 만들고
  - 수퍼 클래스의 생성자 루틴을 수행하고,
  - 프롬프트 모양을 정해주고(self.prompt),
  - 나머지 여러분이 필요로 하는 명령어에 필요한 멤버를 만들고 이용하면 된다.
- 명령어 정의하는 법
  - 명령어 메쏘드 : do_명령어
  - 도움말 메쏘드 : help_명령어
- 명령어 해석기 실행
  - cmdloop() 메쏘드 실행

12-8 상속의 예

```
importsys, cmd
class MyCmd(cmd.Cmd):
    def__init__(self):
        cmd.Cmd.__init__(self)
        self.prompt = "--> "
        self.list = []
    def do_add(self, x):              # add 명령
        if x and (x not in self.list):
            self.list.append(x)
    def help_add(self):       # add 도움말
        print 'help for add'
    def do_show(self, x):     # show 명령
        printself.list
    def do_EOF(self, x):              # EOF 키가 입력되었을 때 (Ctrl-Z 또는 Ctrl-D)
        sys.exit()

if__name__=='__main__':
    c = MyCmd()
    c.cmdloop()
```


12-8 상속의 예

```
--> help

Documentedcommands(typehelp<topic>):
========================================
add
Undocumentedcommands:
======================
EOF             help            show

--> help add
help for add
--> add gslee
--> show
['gslee']
--> add python
--> show
['gslee','python']
-->
```

12-9 상속관계 정보

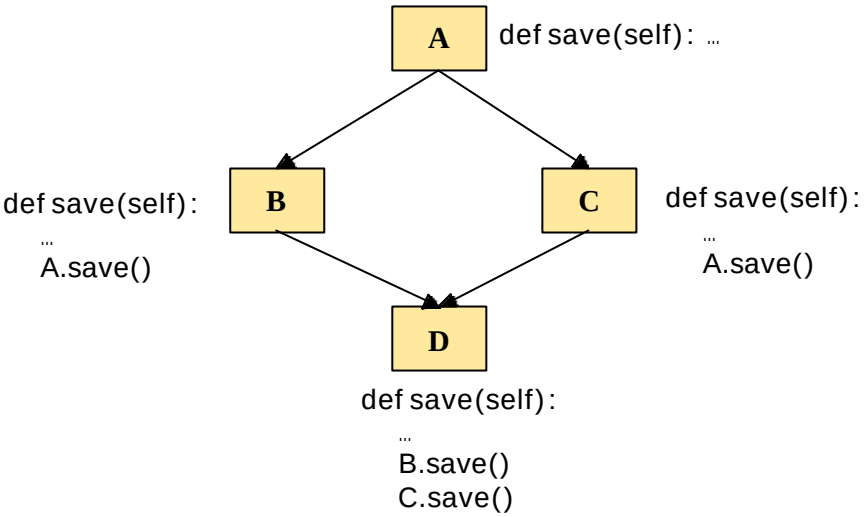
- 상속관계의 클래스 정보 알아내기
  - isinstance()
    - 객체가 어떤 클래스에 속해있는지 확인하기
  - issubclass()
    - 클래스간의 상속 관계 알아보기
  - __bases__
    - 수퍼 클래스 목록 얻기
  - __class__
    - 인스턴스 객체의 클래스 알아내기
  - __dict__
    - 클래스 객체와 인스턴스 객체의 이름공간 얻기

12-10 협동적 클래스와 super

- 협동적 클래스란?
  - 상속 관계에 있는 클래스들이 super 콜을 이용하여 협동적으로 하나의 작업을 수행하는 것
  - super 를 이용하면 협동적 일을 쉽고 간결하게 해결
    - super(자신의클래스이름, self).메쏘드()
    - 수퍼 클래스를 지정하지 않아도 MRO(method resolution order)에 따라서 수퍼 클래스의 메쏘드를 자동으로 호출한다
    - 단, 최상위 클래스는 object의 섭클래스이어야 한다.

12-10 협동적 메쏘드와 super

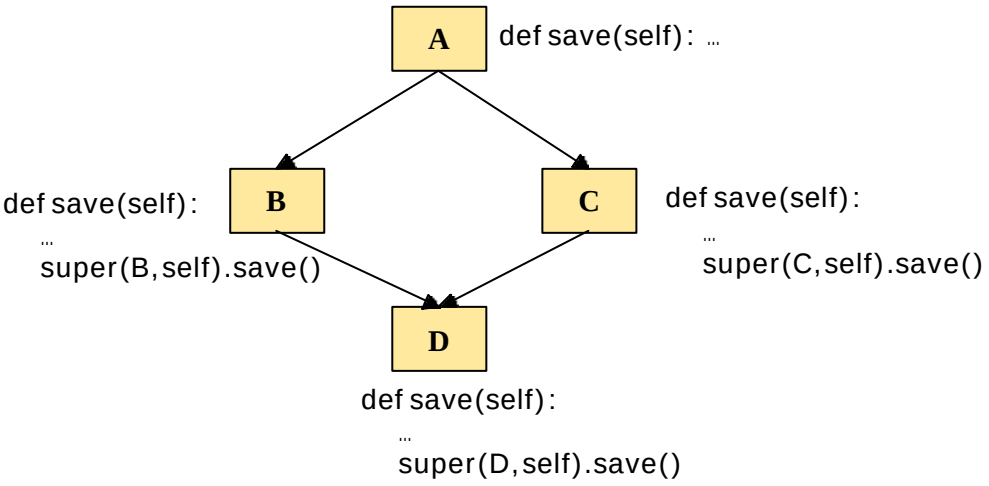
- 문제가 발생하는 경우



save 호출 순서 : D B A C A

12-10 협동적 메쏘드와 super

- super를 이용한 문제 해결



save 호출 순서 : D B C A

12-11 내장 자료형과 클래스

- 자료형 이름과 생성자의 일치(2.2)
  - 자료형의 이름은 생성자의 이름과 같다.
- 클래스 형과 자료형의 일치(2.2)
  - 클래스 이름이 곧 자료 형이 된다.

12-11 내장 자료형과 클래스

- 자료형 이름과 생성자의 일치(2.2)

표 13-11. 내장 함수에서 변경된 팩토리 함수로서의 내장 자료형

내장 자료형	설명
int(number_or_string[, base_number])	int('123')
long(number_or_string)	long('1234')
float(number_or_string)	float('123.45')
complex(number_or_string[, imag_number])	complex('3+4j')
str(obj)	str(123)
unicode(string[, encoding_string])	unicode('abc')
tuple(iterable)	tuple('spam')
list(iterable)	list('spam')
type(object) 또는 type(name_string, bases_tuple, methods_dict)	type('spam')

12-11 내장 자료형과 클래스

- 자료형 이름과 생성자의 일치(2.2)

표 13-12. 새로 추가된 팩토리 함수로서의 내장 자료형

내장 자료형	설명
dict()	사전 객체를 생성한다.
object()	모든 내장 객체의 베이스클래스인 object 객체 생성
classmethod(function)	클래스 메쏘드 생성
staticmethod(function)	정적 메쏘드 생성
super(class_or_type, instance)	
property(get_function[, set_function])	

12-11 내장 자료형과 클래스

- 내장 자료형 서브 클래싱(연산자 중복)

```
>>> class MyList(list):
        def __sub__(self, other):   # '-' 연산자 중복 함수 정의
            L = self[:]             # 전체 리스트를 복사해서 얻는다
            for x in other:
                if x in L: L.remove(x)
            return L

>>> L = MyList([1,2,3,'spam', 4,5])
>>> L = L - ['spam']
>>> print L
[1, 2, 3, 4, 5]
```

12-11 내장 자료형과 클래스

- 내장 자료형 서브 클래싱하기(Stack)

```
>>> class Stack(list):
        push = list.append


>>> s = Stack()
>>> s.push(1)
>>> s.push('spam')
>>> s
[1, 'spam']
>>> s.pop()
'spam'
```


12-11 내장 자료형과 클래스

- 내장 자료형 서브 클래싱하기(Queue)

```
>>> class Queue(list):
        enqueue = list.append
        def dequeue(self):
            return self.pop(0)


>>> q = Queue()
>>> q.enqueue(1)
>>> q.enqueue(2)
>>> q
[1, 2]
>>> q.dequeue()
1
```

12-11 내장 자료형과 클래스

- 내장 자료형 서브 클래싱(XML 형식 출력하기)

```
>>> class xmldic(dict):
    def __repr__(self):
        res = ['\n<dictionary>']
        for k,v in self.items():
            res.append('<member>')
            res.append('<name>%s</name>' % k)
            res.append('<value>%s</value>' % repr(v))
            res.append('</member>')
        res.append('\n</dictionary>')
        return '\n'.join(res)
```


12-11 내장 자료형과 클래스

- 실행 결과

```
>>> d1 = xmldic({'one':1, 'two':2})
>>> d1

<dictionary>
<member>
<name>two</name>
<value>2</value>
</member>
<member>
<name>one</name>
<value>1</value>
</member>

</dictionary>
```

12-12 다형성

- 다형성(Polymorphism)
  - "여러 개의 형태를 가진다"는 의미
  - 상속 관계에 있는 다른 클래스들의 인스턴스들이 같은 멤버 함수 호출에 대해 각각 다르게 반응하는 것

12-12 다형성

```
class Animal:
    def cry(self):
        print '...'
class Dog(Animal):
    def cry(self):
        print '멍멍'
class Duck(Animal):
    def cry(self):
        print '꽥꽥'
class Fish(Animal):
    pass
for each in (Dog(), Duck(), Fish()):
    each.cry()
```

12-13 상속과 합성

- 상속(inheritance)
  - "is-a"관계, 사람(A)은 동물(B)이다
  - B로부터 상속받은 클래스 A생성
- 합성(composition)
  - "has-a" 관계, 자동차(A)는 엔진(B)을 갖는다
  - 클래스 A안에 B 인스턴스 객체를 갖는다

12-13 상속과 합성

- 상속의 예

```
class Set(list):
    def union(self, A):
        res = self[:]
        for x in A:
            if x not in res:
                res.append(x)
        return Set(res)

A = Set([1,2,3])
B = Set([3,4,5])
print A.union(B)
```

12-13 상속과 합성

- 합성의 예

```
class Set:
    def __init__(self, d=None):
        self.data = d
    def union(self, A):
        res = self.data[:]
        for x in A:
            if x not in res:
                res.append(x)
        return Set(res)
    def __getitem__(self, k):
        return self.data[k]
    def __repr__(self):
        return `self.data`
```

```
A = Set([1,2,3])
B = Set([3,4,5])
print A.union(B)
```


12-14 캡슐화

- 캡슐화(encapsulation)
  - 필요한 메쏘드와 멤버를 하나의 단위로 묶어 외부에서 접근 가능하도록 인터페이스를 제공 하는 것
- 캡슐화 방식
  - 비공개 방식(black box)
  - 공개 방식(white box)*
- 정보 은닉(information hiding)
  - 정보를 숨기는 것

12-14 캡슐화

- Private 이름 변경 기능
  - __ 로 시작하는 이름을 정의. 앞에 _클래스 이름이 추가된다.
  - 예: 클래스 Encapsulation의 __x → _Encapsulation__x
- 내부적인 이름들을 _로 시작하여 내부용(private)임을 나타낸다.

12-15 위임

- 위임(Delegation)이란?
  - 상속 메카니즘 대신에 사용되는 기법
  - 자신이 처리할 수 없는 메시지 (메쏘드 호출)을 수신하면, 그 메시지를 처리할 수 있는 다른 객체로 전달하는 것
  - 혹은 중간에 있는 클래스가 다른 객체의 메쏘드 호출을 대신 위임받아 처리하는 것
- 구현
  - __getattr__을 이용
    - 정의되지 않은 속성을 참조할 때 호출된다
    - 형식 : __getattr__(self, name)

12-15 위임

```
class Delegation:
    def __init__(self, data):
        self.stack = data
    def __getattr__(self, name):
        print 'Delegating %s ..' % name,
        return getattr(self.stack, name)

a = Delegation([1,2,3,1,5])
print a.pop()
print a.count(1)
```


12-16 문서 문자열

- 클래스나 메쏘드 몸체(body)에 처음으로 오는 문자열은 문서 문자열로 간주된다

```
class Ham:
    "Ham class __doc__ string"
    def func(self):
        "Ham class func __doc__ string"
        pass
```