熱血講義

프리렉의 열혈강의 시리즈

Python 파이썬

이강성

열혈강의 Python 파이썬

❖제 25 강

주제 : C 확장 모듈 및 확장 형 만들기

이강성(gslee@mail.kw.ac.kr)

❖강의 목차

1. 개요
2. C 확장 모듈 만들기
3. C 확장 형 만들기

개요

- C 확장 모듈 / 형을 왜 만드나
  - 처리 시간의 향상
  - 코드의 은닉
  - 다른 C 라이브러리 사용을 위한 Wrapper 코드 제작
- 대표적 모듈
  - NumPy(http://www.pfdubois.com/numpy/)
  - wxPython(http://wxpython.org/)
- 프로토타이핑(prototyping)
  - 파이썬을 이용한 빠른 알고리즘 개발
  - 프로파일링(profiling) 결과에 따른 부분적인 코드 전환

변환 도구들

- **SWIG**
  - C 소스파일/C 라이브러리를 파이썬과 자동 연결하기 위한 도구 (http://www.swig.org/)
- **PyInline**
  - C 언어를 파이썬에 작성 가능 (http://pyinline.sourceforge.net/)
- **Pyrex**
  - 파이썬에 C자료형을 함께 코딩하고 C 확장 모듈로 만들어 주는 도구 (http://www.cosc.canterbury.ac.nz/~greg/python/Pyrex/)

변환 도구들

- 기타 참고 도구들
  - Psyco (http://homepages.ulb.ac.be/~arigo/psyco/)
    - 파이썬 코드의 실행을 2배에서 100배까지 빠르게 해주는 도구
  - py2exe (http://py2exe.sourceforge.net/)
    - 파이썬 프로그램을 .exe 실행파일로

확장 모듈과 확장 형

- 확장 모듈(extension module)
  - 파이썬 모듈을 C로 구현
- 확장 형(extension type)
  - 파이썬 클래스를 C로 구현

C 확장 모듈

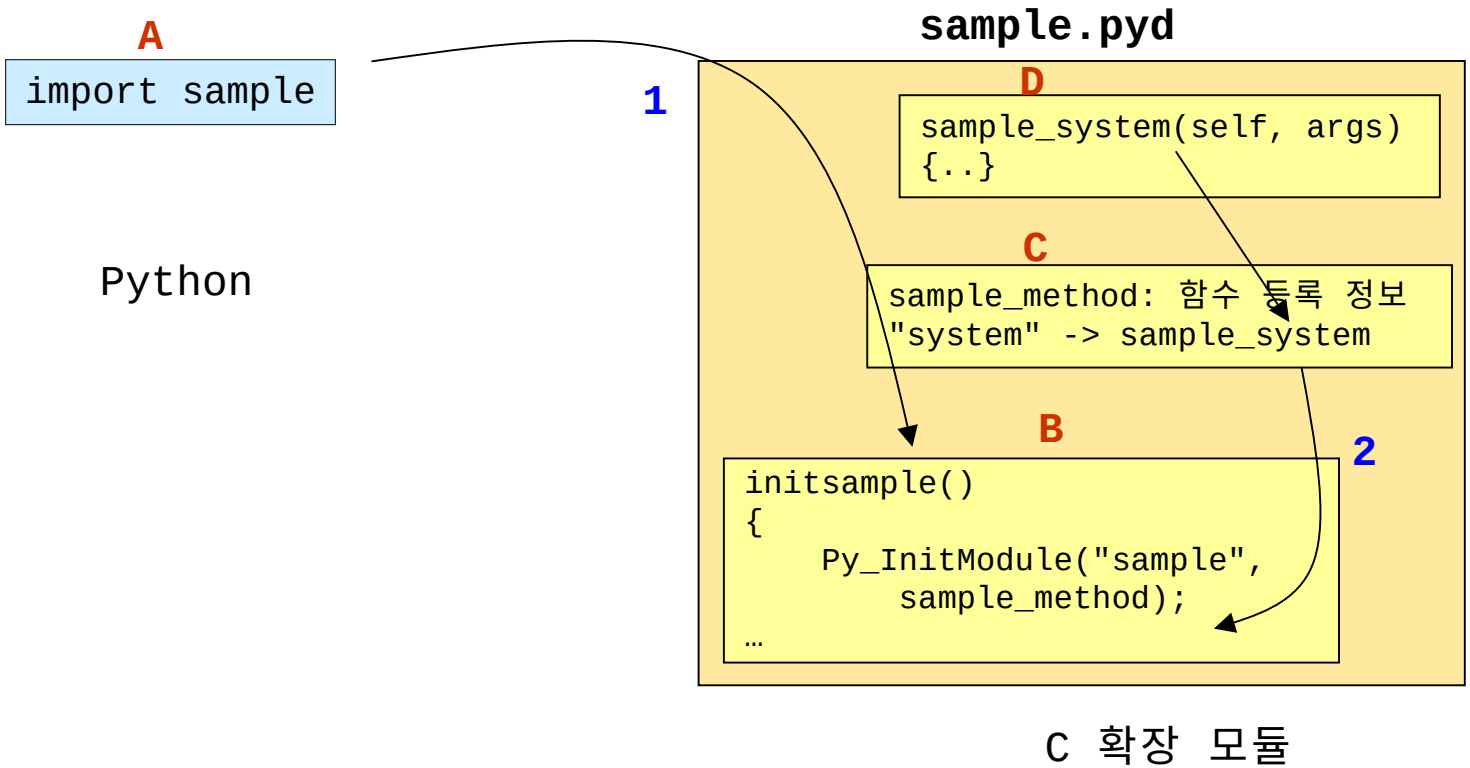
- 모듈 초기화 과정
- 함수 정의 방법
- 자료형의 변환
- 컴파일하기

import 절차

- import sample 절차
  - 모듈 sample을 찾는다
    - sys.path 경로를 기준
    - C모듈 → 파이썬 모듈
  - 모듈을 초기화 한다
    - 다음 슬라이드에서..
  - 지역 이름 공간에 모듈 이름 등을 등록한다
    - __dict__ 에 이름 등록

모듈 초기화 과정

- 모듈 초기화
  - ‘init+모듈이름’ 함수 호출



초기화 함수

- initsample()의 예

```
void initsample()
{
    PyObject *m;
    /* 모듈을 생성하고 함수를 등록한다 */
    m = Py_InitModule("sample", sample_methods);
    /* 기타의 초기화 처리 */
    ErrorObject = Py_BuildValue("s", "sample error");
    /* 기타 다른 코드들.. */
}
```

함수의 등록

- 공개 함수의 등록
  - 함수 이름, 함수 포인터, 인수 받는 형식
  - METH_VARARGS : 파이썬에서 C로 튜플 형식의 인수 전달

```
static struct PyMethodDef sample_methods[] = {
 {"system",        sample_system,     METH_VARARGS},
 {NULL,          NULL}
};
```

함수의 정의

- 함수의 정의

인수들이
튜플로 묶어서
전달됨

확장형에서만
사용됨

```
static PyObject* sample_system(PyObject *self, PyObject *args)
{
    char *command;
    int sts;
    if (!PyArg_ParseTuple(args, "s", &command))
        return NULL;
    sts = system(command);
    return Py_BuildValue("i", sts);
}
```

파이썬
인수를
C인수로 변환

C 값을
파이썬
객체로 변환

전체 샘플 코드(sample.c)

```
#include "Python.h"

static PyObject *ErrorObject;

static PyObject* sample_system(PyObject *self, PyObject *args)
{
    char *command;
    int sts;
    if (!PyArg_ParseTuple(args, "s", &command))
        return NULL;
    sts = system(command);
    return Py_BuildValue("i", sts);
}

static struct PyMethodDef sample_methods[] = {
 {"system",       sample_system,    METH_VARARGS}, /* 함수 이름과 주소 */
 {NULL,         NULL}                             /* 마지막 표시 */
};

void initsample()
{
    PyObject *m;
    /* 모듈을 생성하고 함수를 등록한다 */
    m = Py_InitModule("sample", sample_methods);  /* 모듈 등록 */
    /* 기타의 초기화 처리 */
    ErrorObject = Py_BuildValue("s", "sample error");
    /* ... */
}
```

컴파일

- setup.py 만들기

```
#!/usr/bin/env python

from distutils.core import setup, Extension

setup(name="sample",
      version="1.0",
      description="sample extension module",
      author="Gang Seong Lee",
      author_email="gslee@mail.gwu.ac.kr",
      url="http://www.python.or.kr/",
      ext_modules=[Extension("sample", ["sample.c"])]
     )
```

컴파일

- 컴파일하기
  - 리눅스/윈도우 동일함

```
$ python setup.py build
$ ls build/lib.linux-i686-2.2/
sample.so
```

- (컴파일하고) 설치하기

```
$ python setup.py install
```

테스트하기

```
>>> import sample          # 리눅스에서의 테스트
>>> sample.system('date')
Thu Jan 24 09:26:40 KST 2002
0

>>> import sample          # 윈도우에서의 테스트
>>> sample.system('calc.exe')
0
```

인수의 해석

- 직접 변환 가능 객체

```
int PyArg_ParseTuple(PyObject *arg, char *format, ...);
```

변환기호	C 자료형	파이썬 자료형	의미
s	char *	String	문자열, 유니코드 문자열
s#	char*, int	String	문자열 s와 길이
c	char	String	길이 1 문자열을 C 문자로 변환
b	char	Integer	파이썬 정수를 1바이트 정수로 변환
h	short int	Integer	파이썬 정수를 short int로 변환
i	int	Integer	파이썬 정수형을 C 정수형으로 변환
l	long	Integer	파이썬 정수형을 C의 롱형으로 변환
f	float	Float	
d	double	Float	
D	Py_complex	Complex	파이썬 복소수를 Py_complex 형으로
O	PyObject*	Object	파이썬 객체를 PyObject 형으로

인수의 해석

- 예제

파이썬 호출	C 인수 해석	설명
	`int ok;  int i, j;  long k, l;` `char *s;  int size;`	
`f()`	`ok = PyArg_ParseTuple(args, "");`	전달된 인수가 없음을 확인. 전달된 인수가 없을 때 참 리턴
`f('whoops!')`	`ok = PyArg_ParseTuple(args, "s", &s);`	문자열 주소를 S에 넘겨준다
`f(1, 2, 'three')`	`ok = PyArg_ParseTuple(args, "lls",` `&k, &l, &s);`	두 개의 정수 객체를 C의 long으로, 또 문자열을 받는다.
`f((1,2), 'three')`	`ok = PyArg_ParseTuple(args, "(ii)s#",` `&i, &j, &s, &size);`	터플(혹은 리스트)안의 두 개의 정수를 읽고, 문자열을 받는다. 문자열의 길이는 size에 저장.

`f('spam')`
`f('spam', 'w')`
`f('spam', 'wb',`
`100000)`
	`char *file; char *mode = "r"; int`
	`bufsize = 0;`
	`ok = PyArg_ParseTuple(args, "s|si",`
	`&file, &mode, &bufsize);`
		| 는 선택적인 인수를 나타낸다.
		하나는 필수이고,
		나머지는 선택적이다.

인수의 해석

- 고급 인수 해석하기
  - 리스트, 튜플, 사전등의 인수의 해석
  - 변환기호 **O**를 이용하여 **PyObject*** 로 읽어들인 후 각 자료형에 맞게 처리

인수의 해석

- 자료형 검사

```
int PyNumber_Check(PyObject *o)
int PySequence_Check(PyObject *o)
int PyMapping_Check(PyObject *o)
int PyIter_Check(PyObject *o)
```

수치형

```
int PyInt_Check(PyObject* o)
int PyLong_Check(PyObject *p)
int PyFloat_Check(PyObject *p)
int PyComplex_Check(PyObject *p)
```

시퀀스

```
int PyString_Check(PyObject *o)
int PyUnicode_Check(PyObject *o)
int PyBuffer_Check(PyObject *p)
int PyTuple_Check(PyObject *p)
int PyList_Check(PyObject *p)
```

사전형

```
int PyDict_Check(PyObject *p)
```

기타 자료형은 Python/C API 추가로 참조

인수의 해석

- 리스트 객체 API

함수	설명
int PyList_Check (PyObject *p)	리스트 형인지 검사. 리스트 형이면 참 리턴
PyObject* PyList_New (int len)	새로운 리스트 객체 만든다
int PyList_Size (PyObject *list)	리스트의 크기 len(L)
PyObject* PyList_GetItem (PyObject *list, int index)	리스트 연산 L[index]와 같다
int PyList_SetItem (PyObject *list, int index, PyObject *item)	리스트 연산 L[index] = item
int PyList_Insert (PyObject *list, int index, PyObject *item)	리스트 연산 L.insert(index, item)
int PyList_Append (PyObject *list, PyObject *item)	L.append(item)
PyObject* PyList_GetSlice (PyObject *list, int low, int high)	L[low:high]
int PyList_Sort (PyObject *list)	L.sort()
int PyList_Reverse (PyObject *list)	L.reverse()
PyObject* PyList_AsTuple (PyObject *list)	tuple(L)

인수의 해석

- 사전형 객체 API

함수	설명
PyObject* PyDict_New ()	새 사전을 만든다
PyObject* PyDict_GetItem (PyObject *p, PyObject *key)	p[key]
PyObject* PyDict_GetItemString (PyObject *p, char *key)	p[key]
int PyDict_SetItem (PyObject *p, PyObject *key, PyObject *val)	p[key] = val
int PyDict_SetItemString (PyDictObject *p, char *key, PyObject *val)	p[key] = val
PyObject* PyDict_Items (PyObject *p)	p.items() PyListObject 리턴
PyObject* PyDict_Keys (PyObject *p)	p.keys() PyListObject 리턴
PyObject* PyDict_Values (PyObject *p)	p.values() PyListObject 리턴

인수의 해석

- 예제
  - sum 함수의 구현

```
PyObject *list, *o;
int n, i;
double sum = 0.0;

if (!PyArg_ParseTuple(args, "O", &list))
    return NULL;

if (PyList_Check(list)) {
    n = PyList_Size(list);
    for (i = 0; i < n; i++)
        o = PyList_GetItem(list, i);
        if (PyInt_Check(o))
            sum += PyInt_AsLong(o);
        elif (PyFloat_Check(o))
            sum += PyFloat_AsDouble(o);
}
return Py_BuildValue("d", sum);
```

파이썬 객체의 생성

- C에서 파이썬 객체를 만드는 방법 1
  - Py_BuildValue를 이용한 방법

```
PyObject *Py_BuildValue(char *format, ...);
```

  - PyArg_ParseTuple의 format 문자열과 동일

표 25-3 Py_BuildValue() 함수 호출의 예

함수 호출	리턴 값
Py_BuildValue("")	None
Py_BuildValue("i", 123)	정수 123
Py_BuildValue("iii", 123, 456, 789)	터플 (123, 456, 789)
Py_BuildValue("s", "hello")	문자열 'hello'
Py_BuildValue("{s:i,s:i}", "abc", 123, "def", 456)	사전 {'abc': 123, 'def': 456}

파이썬 객체의 생성

- C에서 파이썬 객체를 만드는 방법 2
  - 자료형 생성에 해당하는 API 이용

표 25-4 C의 단순 자료형에서 파이썬 객체로의 변환

함수	설명
PyObject* PyInt_FromLong (long ival)	C의 long을 파이썬의 int로
PyObject* PyLong_FromLong (long v)	C의 long을 파이썬의 long으로
PyObject* PyFloat_FromDouble (double v)	C의 double을 파이썬의 float로
PyObject* PyString_FromString (const char *v)	C의 string을 파이썬의 string으로

파이썬 객체의 생성

- C에서 파이썬 객체를 만드는 방법 2
  - 파이썬 자료형 생성

```
PyObject* PyTuple_New(int len)
PyObject* PyList_New(int len)
PyObject* PyDict_New()
...
```

리턴하기

- 리턴 자료형
  - PyObject*
    - return PyFloat_FromDouble(5.3)
- None의 리턴
  - Py_INCREF(Py_NONE);
  - return Py_None;
- 예외 발생
  - 예외 종류 설정 후 리턴
    - void PyErr_SetString (PyObject *type, char *message)
    - PyErr_SetString(PyExc_IndexError, "my exception");
    - return NULL; /* NULL은 예외 발생 */

예외가 발생한 경우

- 예외의 설정
  - PyErr_SetString으로 예외의 종류를 설정하고 NULL을 리턴
- 이미 예외가 발생한 경우
  - return NULL; /* 이것만 수행 */

```
if (!PyArg_ParseTuple(args, "ddd", &from, &to, &step))
        return NULL;
```

```
>>> sample.frange(0.0, 1.0)
Traceback (most recent call last):
  File "<stdin>", line 1, in ?
TypeError: function requires exactly 3 arguments; 2 given
```

예외의 발생

- 표준 예외의 발생
  - 표준 예외 변수 (PyExc_...)
  - Python/C API 4.1절

C Name	Python Name
PyExc_Exception	Exception
PyExc_StandardError	StandardError
PyExc_ArithmeticError	ArithmeticError
PyExc_LookupError	LookupError
PyExc_AssertionError	AssertionError
PyExc_AttributeError	AttributeError
...	...

```
PyErr_SetString(PyExc_IndexError, "my exception");
return NULL;
// 파이썬에서 raise IndexError, "my exception" 과 동일
```

예외의 발생

- 사용자 예외의 발생
  - 사용자 예외 형 만들고 / 예외 발생
  - 예

```
static PyObject *ErrorObject; /* 외부 변수로 선언 */
...
/* 문자열 객체 생성 */
/* initsample에서 한번만 실행 */
ErrorObject = Py_BuildValue("s", "sample error");
...
/* 적당한 곳에서 예외 발생 */
PyErr_SetString(ErrorObject, "my exception");
return NULL;
```

예외의 검출

- C에서의 예외 검출은
  - 대부분의 API가 예외를 발생할 수 있다
  - 명시적이어야 한다
  - 예외의 전달이 이루어지지 않는다.
- 예외 발생했는지 검사
  - API가 NULL 혹은 -1을 리턴
  - 명시적 예외 발생 확인
    - PyObject* PyError_Occured()
      - 리턴 NULL – 예외 없을 때
      - 예외 타입에 대한 레퍼런스
- 발생한 예외 삭제
  - PyError_Clear()

예외의 검출

- 예 (파이썬 코드)

```
def incr_item(dict, key):
    try:
        item = dict[key]
    except KeyError:
        item = 0
    dict[key] = item + 1
```

- 예 (C 코드)

```
int incr_item(PyObject *dict, PyObject *key)
{
    /* Objects all initialized to NULL for Py_XDECREF */
    PyObject *item = NULL, *const_one = NULL, *incremented_item = NULL;
    int rv = -1; /* Return value initialized to -1 (failure) */
```

예외의 검출

```
item = PyObject_GetItem(dict, key);
if (item == NULL) {
    /* Handle KeyError only: */
    if (!PyErr_ExceptionMatches(PyExc_KeyError))
        goto error;

    /* Clear the error and use zero: */
    PyErr_Clear();
    item = PyInt_FromLong(0L);
    if (item == NULL)
        goto error;
}
```

예외의 검출

```
    const_one = PyInt_FromLong(1L);
    if (const_one == NULL)
        goto error;

    incremented_item = PyNumber_Add(item, const_one);
    if (incremented_item == NULL)
        goto error;

    if (PyObject_SetItem(dict, key, incremented_item) < 0)
        goto error;
    rv = 0; /* Success */
    /* Continue with cleanup code */

 error:
    /* Cleanup code, shared by success and failure path */

    /* Use Py_XDECREF() to ignore NULL references */
    Py_XDECREF(item);
    Py_XDECREF(const_one);
    Py_XDECREF(incremented_item);

    return rv; /* -1 for error, 0 for success */
}
```

레퍼런스 카운트

- 쓰레기 수집에 필요한 변수
- 개념들..
  - C에서는 레퍼런스 카운트를 명시적으로 다루어야 한다
  - 객체는 공유되며 레퍼런스는 소유된다
  - 레퍼런스 카운트 증가 후 반드시 감소해야 한다
  - 리턴되는 레퍼런스는 새 레퍼런스이어야 한다
  - 함수로 전달된 인수는 모두 빌린 레퍼런스이다

레퍼런스 카운트

- New Reference(새 레퍼런스)
  - 객체의 새로운 레퍼런스
- Borrowed Reference(빌린 레퍼런스)
  - 다른 소유의 레퍼런스를 잠깐 빌리는 것
- API에 명시되어 있음
  - 예

```
PyObject* PyNumber_Add(PyObject *o1, PyObject *o2)
Return value: New reference.

Returns the result of adding o1 and o2, or NULL on failure. This
is the equivalent of the Python expression "o1 + o2".
```

레퍼런스 카운트

- 레퍼런스를 다루는 API

```
Py_INCREF(Py_Object*)   # 레퍼런스 카운트 증가
Py_DECREF (Py_Object*)  # 레퍼런스 카운트 감소
Py_XINCREF (Py_Object*)         # NULL인  경우  무시하고 처리
Py_XDECREF (Py_Object*)         # NULL인  경우  무시하고 처리
```

레퍼런스 카운트

- 예

```
for (i = 0; i < n; i++) {
        item = PySequence_GetItem(sequence, i); /* new reference */
        /* ... */
        Py_DECREF(item); /* 새 레퍼런스를 받았으므로 사용이 끝났으면 카운트를
감소한다 */
 }

for (i = 0; i < n; i++) {
        item = PyList_GetItem(list, i); /* borrowed reference */
        /* ... */
        /* 빌린 레퍼런스를 받았으므로 카운트를 조정할 필요가 없다 */
}
```

레퍼런스 카운트

- API가 인수로 전달되는 객체로의 레퍼런스를 설정하는 경우
  - 전달된 객체의 레퍼런스를 그대로 이용하는 경우(steal)
  - 레퍼런스 카운트를 새로 설정하는 경우

레퍼런스 카운트

- 예

```
PyObject* o;

for (k = 0; k < n; k++) {
        o = PyInt_FromLong ((long)k)    /* New Reference */
        item = PyTuple_SetItem(tup, k, o); /* steal */
        /* ... */
        /* 카운트 값이 훔쳐졌으므로 감소시킬 필요 없다. */
}

for (k = 0; k < n; k++) {
        o = PyInt_FromLong ((long)k)    /* New Reference 받는다 */
        item = PySequence_SetItem(tup, k, o);
        /* ... */
        Py_XDECREF (o)  /* 카운트 값이 그대로 있으므로 감소시켜야 한다. */
}
```

레퍼런스 카운트

- 레퍼런스 카운트를 잘 못 설정하면?
  - 메모리 누수 현상 발생
  - Segmentation Fault 발생

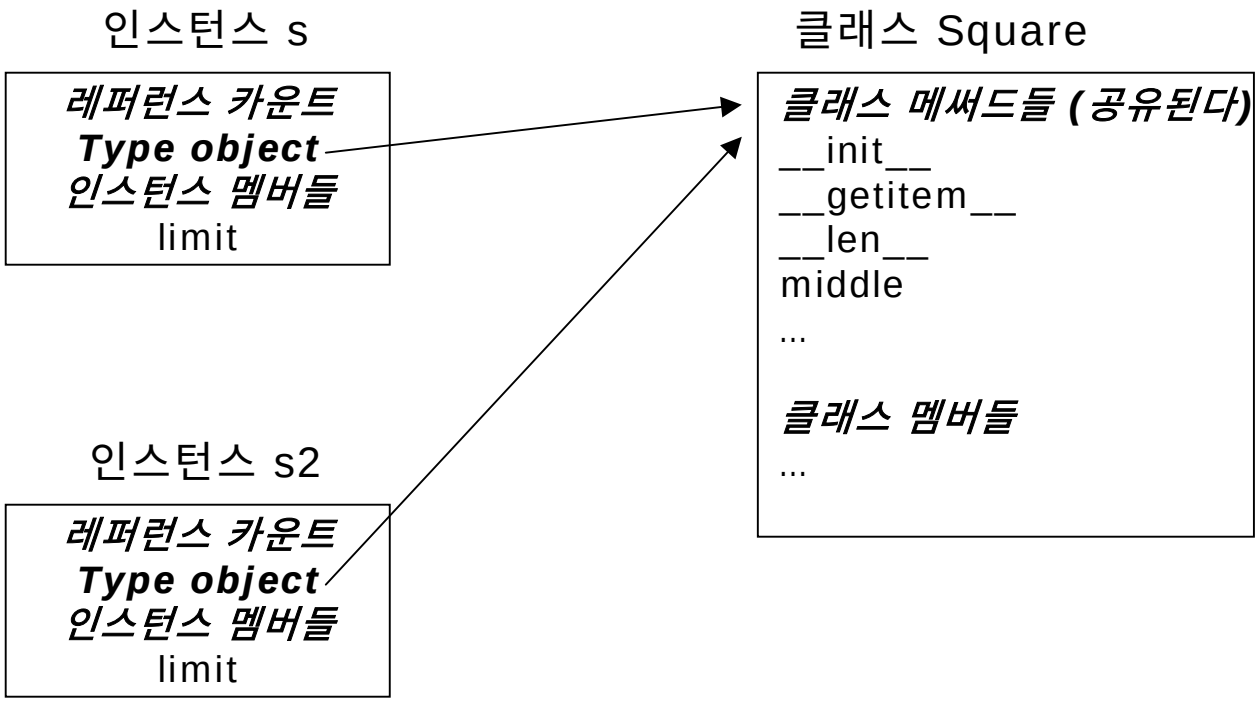
C 확장 형 만들기

- C로 파이썬 클래스 만들기
- 클래스 객체와 인스턴스 객체의 관계
  - 파이썬 소스

```
class Square:
    def __init__(self, limit):
        self.limit = limit
    def __getitem__(self, k):
        if k < 0 or self.limit <= k:
            raise IndexError
        return k*k
    def __len__(self):
        return self.limit
    def middle(self):
        return self.limit / 2

s = Square(10)
print s[3]  # 9
print s.middle() # 5
```

클래스/인스턴스 객체의 관계



확장 형 구현

- 초기화 함수

```
static struct PyMethodDef square_methods[] = {
    {"Square",  square_new,  METH_VARARGS, "Create new Square object"},
    {NULL,      NULL}
};

void initsquare()
{
    PyObject *m, *d;
    Squaretype.ob_type = &PyType_Type;
    m = Py_InitModule("square", square_methods);   /* 모듈 초기화 */
...
}
```

Square()
Square(10)
에 의해 호출될 함수

확장 형 구현

- 생성자 정의

```
static PyObject* square_new(PyObject *self, PyObject *args)
{
    int limit;
    if (!PyArg_ParseTuple(args, "i", &limit)) /* args는 생성자 인수 */
        return NULL;
    return (PyObject *)newsquareobject(limit); /* 인스턴스 객체를
넘겨준다 */
}
```

Square 인스턴스 객체 자료형을 만들어서 리턴

확장 형 구현

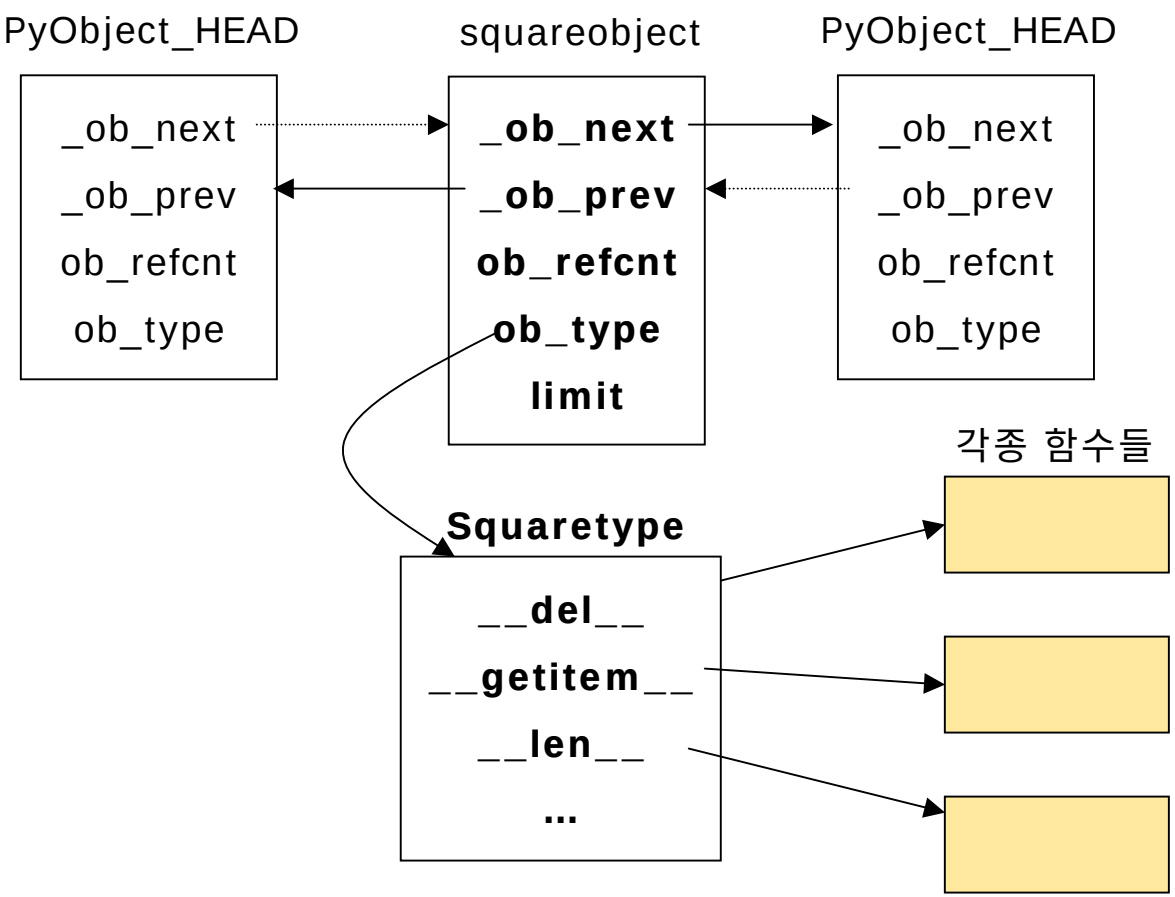
- 인스턴스 객체 정의 함수

```
typedef struct {          /* Square 인스턴스 객체 */
    PyObject_HEAD         /* 파이썬 헤더 : 레퍼런스 카운트와 형정보를 갖는다 */
    int limit;            /* 인스턴스 객체 멤버를 여기부터 선언한다 */
} squareobject;


static squareobject* newsquareobject(int limit)
{
    squareobject *self;
    self = PyObject_NEW(squareobject, &Squaretype); /* squareobject
객체 생성 */
    if (self == NULL)     /* 객체 생성에 실패하면 예외 발생 */
        return NULL;
    self->limit = limit;  /* 생성자에 전달된 인수 처리 */
    return self;          /* 새로운 squareobject 형 리턴 */
}
```

확장 형 구현

- 인스턴스 객체와 클래스 형의 구조



확장 형 구현

- 클래스 형 선언

```
static PyTypeObject Squaretype = {
      /* 타입 헤더 */                          /* 모든 인스턴스가 공유한다 */
      PyObject_HEAD_INIT(NULL) /* 타입 정보는 initsquare에 의해 채워짐
*/
      0,                                      /* ob_size */
      "Square",                                /* tp_name */
      sizeof(squareobject),                    /* tp_basicsize */
      0,                                      /* tp_itemsize */

      /* 표준 메써드들 */
      (destructor)  square_dealloc,            /* tp_dealloc */
      (printfunc)   0,                         /* tp_print */
      (getattrfunc) square_getattr,            /* tp_getattr */
         ...

      /* 자료형 종류 */
      0,                                  /* 수치형 메써드들 */
      &square_sequence,                /* 시퀀스형 메써드들 */
         ...
};  /* 추가의 다른 메써드들은 object.h 참조 */
```

확장 형 구현

- 표준 메쏘드 정의

```
static PyObject* square_dealloc(squareobject* self)
{
    PyObject_Del(self);
}

static PyObject* square_getattr(squareobject* self, char* name)
{
    return Py_FindMethod(square_inst_methods, (PyObject *)self, name);
}
```

확장 형 구현

- 사용자 메쏘드 정의
  - __getattr__ 에 의해서 검색 실행됨

```
static struct PyMethodDef square_inst_methods[] = {     /* 추가의 메써드들 */
 {"middle",        square_middle,     METH_VARARGS, "middle point"},
 {NULL,          NULL,     0, NULL}          /* 마지막 표시 */
};

static PyObject* square_middle(squareobject* self, PyObject* args)
{
    if (!PyArg_ParseTuple(args, ""))
        return NULL;
    return Py_BuildValue("i", self->limit / 2);
}
```

확장 형 구현

- 자료형에 따른 메쏘드 선언

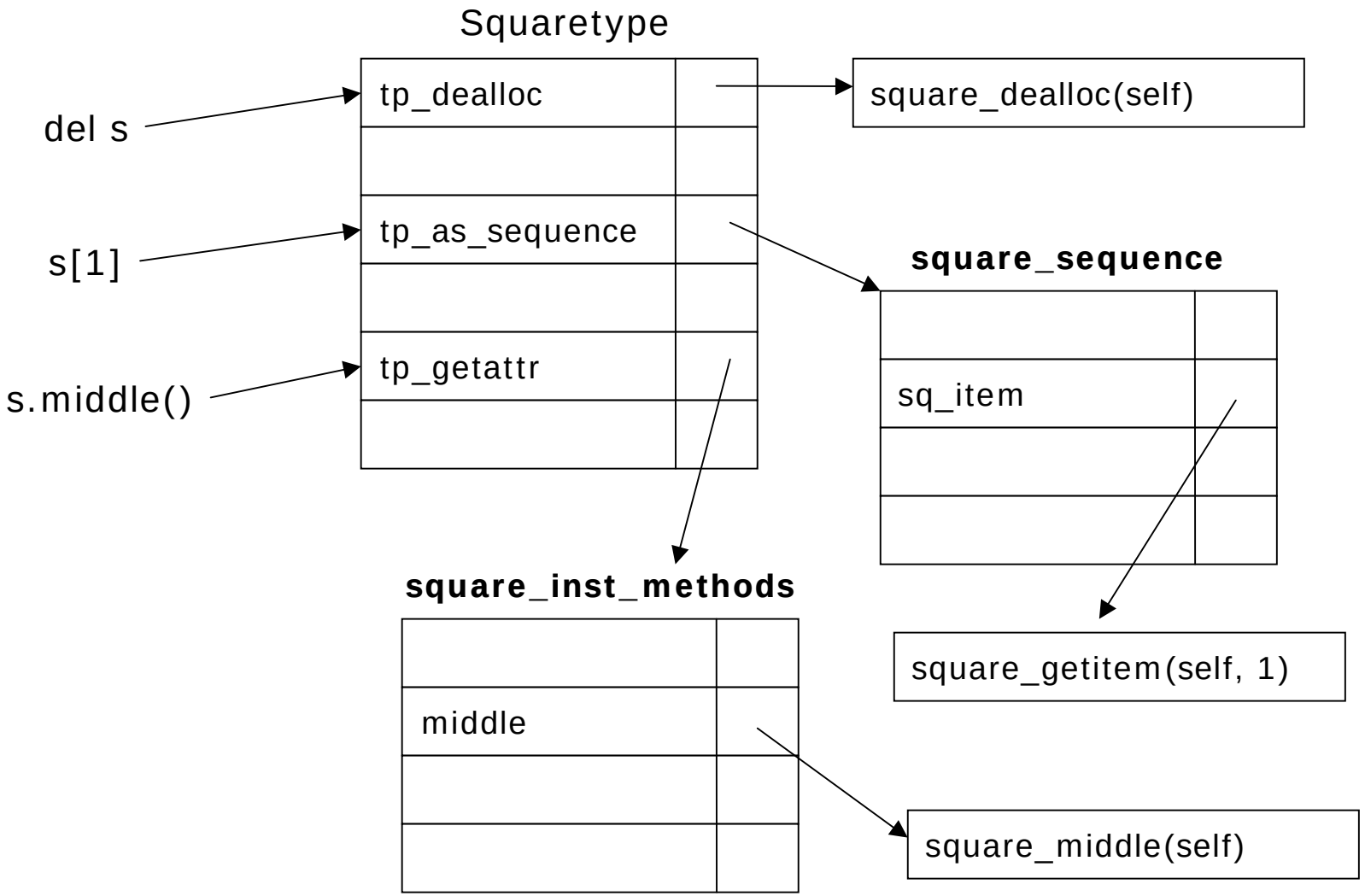
```
static PySequenceMethods square_sequence = {
      (inquiry)       square_length,            /* len(x)   */
      (binaryfunc)    0,                        /* x + y    */
      (intargfunc)    0,                        /* x * n    */
      (intargfunc)    square_getitem,           /* x[i], in */
      (intintargfunc) square_slice,             /* x[i:j]   */
      (intobjargproc)     0,                    /* x[i] = v */
      (intintobjargproc)  0,                    /* x[i:j]=v */
      (objobjproc) 0,                       /* in */
      /* Added in release 2.0 */
      (binaryfunc) 0;
      (intargfunc) 0;
};
```

확장 형 구현

- 자료형에 따른 메쏘드 정의

```
static int square_length(squareobject* self)
{
    return self->limit;   /* 길이 정보는 정수를 그냥 넘기면 된다 */
}
```

연산자 처리 흐름


Squaretype
del s
tp_dealloc
square_dealloc(self)
s[1]
tp_as_sequence
square_sequence
s.middle()
tp_getattr
sq_item
square_inst_methods
square_getitem(self, 1)
middle
square_middle(self)

전체 프로그램

- 소스 square.c 참조

컴파일

- setup.py

```
#!/usr/bin/env python

from distutils.core import setup, Extension

setup(name="square",
      version="1.0",
      description="simple class example - square",
      author="Gang Seong Lee",
      author_email="gslee@mail.kw.ac.kr",
      url="http://www.python.or.kr/",
      ext_modules=[Extension("square", ["square.c"])]
     )
```

컴파일

- 컴파일과 설치

```
$ python setup.py build
$ python setup.py install
```

- 실행

```
>>> import square
>>> dir(square)
['Square', '__doc__', '__file__', '__name__', 'error']
>>> s = square.Square(10)
>>> for k in s:
...     print k,
...
0 1 4 9 16 25 36 49 64 81
```