



Python C API

Зачем?

- Расширение Python “нативными” модулями
 - Python - медленный, C - быстрый
 - Примеры:
 - стандартные модули: `heapq` (`_heapq`), `ssl` (`_ssl`)
 - привязки к библиотекам: PySide, SciPy и другие
- Расширение “нативных” программ сценариями Python
 - “нативные” программы нужно пересобирать, Python - нет
 - Python проще и гибче

Как?

Python C API

(<http://docs.python.org/3.3/extending/>)

SWIG

(<http://www.swig.org/>)

Boost Python

(http://www.boost.org/doc/libs/1_55_0/libs/python/doc/index.html)

Самый простой пример

```
#include <Python.h>

static PyObject * do_nothing(PyObject * self) { Py_RETURN_NONE; }

static PyMethodDef methods[] = {
    {"do_nothing", do_nothing, METH_NOARGS, "do nothing function"},
    {NULL, NULL, 0, NULL}
};

static struct PyModuleDef module = {
    PyModuleDef_HEAD_INIT, "do_nothing",
    "do nothing module with do nothing function", -1, methods
};

PyMODINIT_FUNC PyInit_do_nothing(void) {
    return PyModule_Create(&module);
}
```

Как собрать?

```
# file: setup.py
from distutils.core import setup, Extension

mdl = Extension('do_nothing', sources = ['example.
c'])
setup(name = 'do_nothing',
      version = '1.0',
      description = 'Python C API Simplest Module',
      ext_modules = [mdl])
```

В терминале:

```
$ python3 setup.py build
```

```
$ sudo python3 setup.py install
```

Рецепт

- Не забываем про `#include <Python.h>`
- Описываем функции модуля:
 - `PyObject * fname(PyObject * self);`
 - `PyObject * fname(PyObject * self, PyObject * args);`
 - `PyObject * fname(PyObject * self, PyObject * args, PyObject * kws);`
- Сводим функции в массив `PyMethodDef`:
 - `{<имя функции>, <указатель на функцию>, <calling conventions>, <doc string>}`
 - типичные calling conventions:
 - `METH_NOARGS`
 - `METH_VARARGS`
 - `METH_VARARGS | METH_KEYWORDS`
- Создаем структуру модуля `PyModuleDef`
- Создаем функцию `PyMODINIT_FUNC PyInit_<имя вашего модуля>(void)`

УСЛОЖНИМ?

```
static PyObject * my_print(PyObject *self, PyObject *args)
{
    Py_ssize_t const len = PyTuple_Size(args);
    Py_ssize_t pos = 0;

    for (pos = 0; pos < len; ++pos)
    {
        PyObject * obj = PyTuple_GetItem(args, pos);
        PyObject * str = PyObject_Str(obj);
        if (pos != 0) printf(" ");
        PyObject_Print(str, stdout, Py_PRINT_RAW); /* 0 for repr */
        Py_DECREF(str);
    }
    printf("\n");

    Py_RETURN_NONE;
}
```

return None

```
Py_RETURN_NONE;
```

```
return Py_BuildValue("");
```

```
Py_INCREF(Py_None);  
return PyNone;
```


Усложним! (1/2)

```
static PyObject * my_print(PyObject *self, PyObject *args)
{
    Py_ssize_t const len = PyTuple_Size(args), pos = 0;
    if (len < 0) return NULL;

    for (pos = 0; pos < len; ++pos)
    {
        ...
        /* продолжение в следующей серии */
        ...
    }

    if (printf("\n") < 0) {
        PyErr_SetFromErrno(PyExc_RuntimeError);
        return NULL;
    }
    Py_RETURN_NONE;
}
```

Усложним! (2/2)

```
PyObject * obj = PyTuple_GetItem(args, pos);

if (obj == NULL) {
    PyErr_SetString(PyExc_RuntimeError, "cannot fetch tuple argument");
    return NULL;
}

if (pos != 0 && printf(" ") < 0) {
    PyErr_SetFromErrno(PyExc_RuntimeError);
    return NULL;
}

if (PyObject_Print(obj, stdout, Py_PRINT_RAW) < 0) {
    if (!PyErr_Occurred())
        PyErr_SetString(PyExc_RuntimeError, "print failed");
    return NULL;
}
```

Обработка ошибок

- Для функций возвращающих указатель (PyObject *) NULL - ошибка! (PyTuple_GetItem)
- Для функций возвращающих число (Py_ssize_t, int и тд) смотреть документацию!
- PyErr_SetString(PyObject * type, char const * message) - **бросить исключение**
- PyErr_SetFromErrno(PyObject *) - **бросить исключение из errno**
- PyErr_NewException(char const * name, PyObject * base, PyObject * dict) - **создать новый эксепшен, типичный вызов:**
 - PyObject * err = PyErr_NewException("module.exception", NULL, NULL);
- PyErr_Occurred() - **проверить произошло ли исключение, и какое именно**
- Список стандартных эксепшенов здесь: http://docs.python.org/3.0/c-api/exceptions.html#PyErr_NewException

Создание классов

- Создаем структуру хранящую данные:

```
typedef struct {  
    PyObject_HEAD  
    int ival;  
} MyNewIntObject;
```

- Создаем структуру описывающую тип:

```
PyTypeObject MyNewIntType = {  
    PyVarObject_HEAD_INIT(NULL, 0)  
};
```

- Добавляем тип в модуль:

```
if (PyType_Ready(type) < 0) return NULL;  
Py_INCREF(type);  
PyModule_AddObject(module, "MyNewInt", (PyObject *)type);
```

Стандартные методы

```
PyTypeObject * type = &MyNewIntType;
. . .
type->tp_name = "myc.MyNewInt";
type->tp_basicsize = sizeof(MyNewIntObject);
type->tp_flags = Py_TPFLAGS_DEFAULT;
type->tp_new = PyType_GenericNew;
. . .
type->tp_doc = "New Int Type";
. . .
type->tp_init = (initproc)MyNewInt_init;
type->tp_repr = (reprfunc)MyNewInt_repr;
type->tp_str = (reprfunc)MyNewInt_repr;
type->tp_hash = (hashfunc)MyNewInt_hash;
. . .
```

Пример

```
static int MyNewInt_init(MyNewIntObject *self,
                        PyObject *args,
                        PyObject *kwargs)
{
    self->ival = 0;
    if (!PyArg_ParseTuple(args, "|i:MyNewInt.__init__()", &self->ival))
        return -1;
    return 0;
}
```

```
static PyObject * MyNewInt_repr(MyNewIntObject *self)
{ return PyUnicode_FromFormat("%i", self->ival); }
```

```
static long MyNewInt_hash(MyNewIntObject *self)
{ return (unsigned)self->ival; }
```

Нестандартные методы

```
static PyObject * MyNewInt_dump(MyNewIntObject *self)
{
    printf("%d\n", self->ival);
    Py_RETURN_NONE;
}
```

```
static PyMethodDef MyNewIntTypeMethods[] = {
    . . .
    {"dump", (PyCFunction)MyNewInt_dump, METH_NOARGS, "dump value"},
    . . .
    {NULL, NULL, 0, NULL}
};
```

```
. . .
type->tp_methods = MyNewIntTypeMethods;
. . .
```

Встраивание Python

```
#include <Python.h>

int main(int argc, char *argv[])
{
    PyObject * module = NULL;

    Py_Initialize();

    PySys_SetPath(L".");
    module = PyImport_ImportModule(argv[1]);
    if (module == NULL && PyErr_Occurred()) PyErr_Print();
    Py_XDECREF(module);

    Py_Finalize();

    return 0;
}
```


Как собрать?

```
python3-config --cflags
```

```
python3-config --ldflags
```



Q&A

