Extending Python with C

ماهى ال Extensions ؟
هى امتدادات لل Python مكتوبة بال C او C-Like مثل ال C++ على سبيل المثال

سهل كتابة extensions لل Python ولكن مالهدف ؟
الهدف إنك تضيف built-in modules لل Python مكتوبة بال C بهدف السرعة مثلا او إضافة built-in types او عمل encapsulation لل C lib functions او System Calls او حتى إخفاء ال source code الخاص بيك :)

المتطلبات: خبرة جيدة بال C و Python
اول شئ بكل تأكيد هو إننا هنستخدم Python API/C وذلك بضم python.h لل project

ملحوظة: قم بضم python.h قبل اى header اخر. جميل نبدأ ب Hello, World :)
1- انشئ ملف helloMod.c
2- اكتب التالى

```
#include <Python.h>

static PyObject* hola(PyObject* self, PyObject* args)
{


   if (!PyArg_ParseTuple(args, "", NULL))
      return NULL;

   printf("Hola!");

   Py_RETURN_NONE;
}

static PyMethodDef HolaMethods[] =
{
    {"hola", hola, METH_VARARGS, "prints Hola\n"},
    {NULL, NULL, 0, NULL}
};

PyMODINIT_FUNC

inithola(void)
{
    (void) Py_InitModule("hola", HolaMethods);
}
```

نبدأ ب module بسيطة وهى hola هنشرح سطر سطر
اولا نعمل include ل python api header كالتالى

```
#include <Python.h>
```

اى Object فى Python تقدر تعرفه ك Pointer ل ب PyObject
احنا الأول عايزين نعرف function بسيطة تطبع كلمة Hola!

```
static PyObject* hola(PyObject* self, PyObject* args)
```

لاحظ self هو Pointer فى حال لو ال PyObject Class ودى هتكون ال method تبعه. لكن لو Function يبقة self هيكون NULL . عايزين يتم إستخدام ال Function كالتالى

```
>>> hola.hola()
Hola!
```

لاحظ شئ إن ال Function مش هتاخد اى argument و مش ليها return او ال Return ب NONE

فالأول نختبر هل فى arguments اتباصت لل Function او لأ

```
    if (!PyArg_ParseTuple(args, "", NULL))
        return NULL;
```

PyArg_ParseTuple هى Function بتسخدم فى عمل Parse او تحليل لل Arguments وفيها بيتم تحويل ال Python Values ل C Values زى ماهنشوف فى مثال قادم.

args هى ال arguments اللى هتتباصى لل Function

هى بتعبر عن ال Data Type الخاص بال Argument مثلا s او i وهكذا

```
s: String
i: Integer
.. etc
```

NULL هنا بيعبر عن ال متغيرات اللى هتاخد القيم اللى اتباصت لل args -هنطلع عليها اكثر فى مثال قادم-

```
        return NULL;
```

للخروج مباشرة من تنفيذ ال Function

بعد كدا نيجى لل ال Function هتعمله وهو طباعة كلمة Hola بإستخدام printf

```
    printf("Hola!");
```

واخيرا زى ماقلنا اننا مش عايزين اى return من ال Function فهنعمل Py_RETURN_NONE

```
    Py_RETURN_NONE;
```

هنحتاج نعرف ال methodTable وهو عبارة عن array بتشمل معلومات عن ال Function زى ال name, address وال Documentation الخاصة بيها وهكذا

```
static PyMethodDef HolaMethods[] =
{
    {"hola", hola, METH_VARARGS, "prints Hola"},
    {NULL, NULL, 0, NULL}
};
```

اول field هو ال name الخاص بال function

التانى هو ال function نفسها

التالت بيعبر عن ان ال Arguments اللى هتتباصى هى Python-Level Arguments وغالبا بنستخدم METH_VARAGS

الرابع هو الوصف الخاص بال function

ونحجز المكان التانى فى ال Array ب {NULL, NULL, 0, NULL}

نعمل Initialize لل Module بتاعتنا كالتالى

```
PyMODINIT_FUNC
inithola(void)
{
    (void) Py_InitModule("hola", HolaMethods);
}
```

PyMODINIT_FUNC هى إختصار ل Python Module Initializer Function و فيها بيتم تجهيز ال Module

بإستدعاء Py_InitModule وهى Function وهى اللى بتقوم بالتجهيز بالفعل وبتاخد 2Arguments

1- اسم ال Module

2- ال Methods Table

جميل جدا.. كدا كتبنا اول Module خاصة بينا!

هنحتاج نضم ال Extension لل Python ولكن إزاى ؟

بكل بساطة افتح ال Editor المفضل عندك وهنعمل setup script بال Python بإستخدام Distutils

```
from distutils.core import setup, Extension
```

```
modExt = Extension('hola', sources = ['hola.c'])

setup (name = 'HolaPackage',
       version = '1.0',
       description = 'Simple demo',
       ext_modules = [modExt])
```

الخطوات سهلة وسلسة كالتالى:

1- إستدعينا setup, Extension من Disutils.core
2- عملنا Extension Object كالتالى

```
modExt = Extension('hola', sources = ['hola.c'])
```

وفيه بنحدد ال source واسم ال extension

3- تجهيز ال setup script بإننا نسجل فيه إسم ال Package و الإصدار والوصف و ال extensions كالتالى

```
setup (name = 'HolaPackage',
       version = '1.0',
       description = 'Simple demo',
       ext_modules = [modExt])
```

كل ماعليك هو

```
Python setup.py build
```

وبعد كدا تعمل Install لل Package كالتالى

```
Python setup.py install
```

نجرب ال Hola Module كالتالى

1- اعمل اى Test Script وليكن HolaTest.py
2- اعمل import ل hola Module كالتالى

```
import hola
```

3- استدعى ال hola function كالتالى

```
hola.hola()

#output:
Hola!
```

4- لنوضيح ال return الخاص بال Function اكتب

```
print hola.hola()

#Output:
Hola!
None
```

بعد ماطلعنا على الأساسيات نجرب نكتب module فيها function بتقبل argument ك name و age وتطبعهم و Function اخرى لحساب القيمة المطلقة لرقم وواحدة تقسم عددين وواحدة تعمل return ب Tuple, Dictionary

الفكرة بإختصار:

1- نعرف ال Functions
2- نضمهم لل Methods Table
3- نعمل Initialize لل Module
4- نجهز ال setup script
5- نعمل Build و Install لل Module
6- نستخدم ال Module عن طريق TestScript مثلا كالتالى

1- تعريف ال Functions

ال Hola Function

```
static PyObject* hola(PyObject* self, PyObject* args)
{

    const char* name;
    int age;

    if (!PyArg_ParseTuple(args, "si", &name, &age))
        return NULL;

    printf("Name: %s", name);
    printf("Age: %i", age);


    Py_RETURN_NONE;
}
```

```
 if (!PyArg_ParseTuple(args, "si", &name, &age))
```

لاحظ إننا هنا توقعنا إن هيتباصى لل Function التالى s وهى string و i وهى integer
وقمنا بإسناد هذه القيم ل name و age

ملحوظة: انت لن تقوم بالتعديل على name فافضل تعريف إنه يكون const فيكون تعريفه كالتالى

```
const char* name;
```

MyABS Function

```
static PyObject* myabs(PyObject* self, PyObject* args)
{

    int number;

    if (!PyArg_ParseTuple(args, "i", &number))
        return NULL;

    if (number<0){
      number=-number;
    }
    return Py_BuildValue("i", number);

}
```

لاحظ اننا توقعنا إن هيتباصى لل Function التالى i وهو integer وبيعبر عن الرقم
اسندنا القيمة الى number (التحويل من Python Value إلى C Value)

ال Return لازم يكون عبارة عن Python Value (او PyObject) وهو ال Return Type الخاص بال myabs Function كما
لاحظت، فبالتالى هنحتاج نحول من ال C Value إلى Python Value ودا هيتم عن طريق إستخدام Py_BuildValue
وهنا تم تحديد إن هيتم عمل return ل i وهو Integer وقيمته مساوية ل number

holaDict Function

```
static PyObject* holaDict(PyObject* self, PyObject* args)
{

    const char* key;
    int value;
```

```
    if (!PyArg_ParseTuple(args, "si", &key, &value))
        return NULL;

    return Py_BuildValue("{s:i}", key, value); //Returns a Dict.
}
```

لاحظ ان بيتم إعادة Dictionary Object واحنا حددنا كدا بالجزئية دى{s:i}
إذا حبيت تعمل return ب List فكل ماعليك هو إنك تعدل ال Format للتالى

```
    return Py_BuildValue("[s,i]", key, value); //Returns a List object
```

وإذا حبيت تعمل return ب Tuple فكل ماعليك هو إنك تعد ال Format كالتالى (s,i)

hola Tuple Function

```
static PyObject* holaTuple(PyObject* self, PyObject* args)
{
    const char* name;
    int age;


    if (!PyArg_ParseTuple(args, "si", &name, &age))
        return NULL;

    return Py_BuildValue("(s,i)", name, age); //Returns a Tuple
}
```

divTwo Function

```
static PyObject* divTwo(PyObject* self, PyObject* args)
{

    int first;
    int second;
    int result;

    if (!PyArg_ParseTuple(args, "ii", &first, &second))
        return NULL;

    printf("First: %i\n", first);
    printf("Second: %i\n", second);

    if(second==0){  //DivByZeroError!
            PyErr_SetString(PyExc_ZeroDivisionError, "DivByZero");
            return NULL; //Get out!
    }
    result = first/second;

    return Py_BuildValue("i", result);
}
```

إذا كان المقسوم عليه يساوى 0 يبقة فى Error! ونقدر نبلغ ال Interpreter بيه بإستخدام PyErr_SetString
نوع ال Error هو PyExc_ZeroDivisionError وال message هتكون DivByZero

2- الضم لل Methods Table كالتالى

```
static PyMethodDef SimpleModuleMethods[]=
{
```

```
    {"hola", hola, METH_VARARGS, "prints name and age"},
    {"myabs", myabs, METH_VARARGS, "returns the abs of a number"},
    {"divTwo", divTwo, METH_VARARGS, "DIV 2 "},
    {"holaTuple", holaTuple, METH_VARARGS, "returns a tuple"},
    {"holaDict", holaDict, METH_VARARGS, "returns a dict"},

    {NULL, NULL, 0, NULL}
};
```

لاحظ إن آخر عنصر فى ال Array هو حاجز..

3- عمل Initialize لل Module كالتالى بنستدعى فيها ال Py_InitModule Function كالتالى

```
PyMODINIT_FUNC

initsimplemodule(void)
{
    (void) Py_InitModule("simplemodule", SimpleModuleMethods);
}
```

4- ال Setup Script كالتالى

```
from distutils.core import setup, Extension

modExt = Extension('simplemodule', sources = ['simplemodule.c'])

setup (name = 'SimpleModPackage',
       version = '1.0',
       description = 'hola, myabs',
       ext_modules = [modExt])
```

بنوضح فيه اسم ال Package والإصدار والوصف وال extension اللى بيشمل اسم ال module وال source

5- عمل Build و Install كالتالى

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

6- عمل Test Script واختبار ال Module كالتالى

```
#!bin/python

import simplemodule as sm

sm.hola("Ahmed", 18)
print sm.myabs(-10) #10
print sm.myabs(7) #7
print sm.holaDict("python", 1)
print sm.holaTuple("ahmed", 999)
print sm.divTwo(2, 0)

#Output:
Name: Ahmed
Age: 18
10
7
{'python': 1}
('ahmed', 999)
First: 2
Second: 0
```

```
Traceback (most recent call last):
  File "C:\Python25\Projects\exten\smTest.py", line 10, in <module>
    print sm.divTwo(2, 0)
ZeroDivisionError: DivByZero
```

References:

1- http://docs.python.org/ext/
2- Programming Python 3rd Edition

Related:

1- Style Guide for C Code
2- SWIG
3- CXX

--Ahmed Youssef