



Module 04 – Interop

Python C interoperability

- There are various tools which make it easier to bridge the gap between Python and C/C++
- **Cython** - programming language, a superset of Python with a foreign function interface for invoking C/C++ routines. It is actually a Python and C source code translator that integrates on a low level.
- **ctypes** is a Python module allowing to create and manipulate C data types in Python. These can then be passed to C-functions loaded from dynamic link libraries.
- **elmer** - compile and run python code from C, as if it was written in C
- **weave** - include C code lines in Python program
- etc

ctypes introduction

- **ctypes** is a foreign function library for Python.
- It provides C compatible data types and allows calling functions in DLLs or shared libraries.
- **ctypes** exports the **cdll/** class for loading dynamic link libraries.
- **cdll** loads libraries which export functions using the standard C decl calling convention
- The **LoadLibrary()** method used to load the dll
- Lets start with simple example

C Test.dll Example

- First, lets create C simple dll named Test
 - Win32->Win32 project
 - Pick dll type, choose Empty project
- Lets write some code

```
#include <stdio.h>
```

```
extern "C" {  
    __declspec(dllexport) void Print() {  
        printf("declspec say hello"); }  
  
    __declspec(dllexport) int Add(int a, int b) {  
        return a+b; }  
}
```

Python code Example

- First step for interoperability is to load dll library.
- Python loads C dlls with LoadLibrary function placed in cdll
- LoadLibrary(dll path) -> loaded dll object
 - dll path doesn't have to have .dll extension
- Loaded dll object now has access to dll's functions

```
import ctypes

loaded_dll = ctypes.cdll.LoadLibrary(r"C:\...\TestDll")
res = loaded_dll.Add(2,3)
print(res)
```

ctype types

- int is the default parameter type or return value type and the only type python can work with without casting
- Lets see C dll function simple example: `__declspec(dllexport) double double_func(double d)`

```
{  
return ++d;  
}
```

```
import ctypes  
d = ctypes.cdll.LoadLibrary(r"C:\...\TestDll")  
res = d.double_func(1.23)  
print (res)
```

ctype types - cont'd

- We get the following result when trying to run the python program:

Traceback (most recent call last):

File "C:\Python27\ctypes.py", line 7, in <module>

res = d.double_func(1.23)

ctypes.ArgumentError: argument 1: <type 'exceptions.TypeError'>:

Don't know how to convert parameter 1

- We need a types conversion table

Types conversion table

<u>cctype</u>	<u>Ctype</u>	<u>Python type</u>
• c_char	char	1-character string
• c_wchar	wchar_t	1-character unicode string
• c_byte	char	int/long
• c_ubyte	unsigned char	int/long
• c_short	short	int/long
• c_ushort	unsigned short	int/long
• c_int	int	int/long
• c_uint	unsigned int	int/long
• c_long	long	int/long
• c_ulong	unsigned long	int/long
• c_longlong	__int64 or long long	int/long
• c_ulonglong	unsigned __int64 or unsigned long long	int/long

Types conversion table — cont'd

<u>ctypes type</u>	<u>C type</u>	<u>Python type</u>
• c_float	float	float
• c_double	double	float
• c_longdouble	long double	float
• c_char_p	char * (NUL terminated)	string or None
• c_void_p	void *	int/long or None

- An example of python code:

```
import ctypes
val = ctypes.c_double(11.22)
print (val)      # c_double(11.22)
print (val.value) #11.22
```

TestDll.dll Example

Demo



```
#include <stdio.h>
#include <stdlib.h>

typedef struct Point {
    int x;
    int y;
} Point;

extern "C" {
    __declspec(dllexport) double double_func(double d) {
        return ++d;
    }

    __declspec(dllexport) char* str_func(char* str) {
        return str;
    }

    __declspec(dllexport) void swap(int* p1, int* p2) {
        int temp = *p1;
        *p1 = *p2;
        *p2 = temp;
    }

    __declspec(dllexport) float* pointer_func(float val) {
        float* ptr = (float*)malloc(sizeof(float));
        *ptr = -val;
        return ptr;
    }

    __declspec(dllexport) Point* struct_func(Point p) {
        Point* cp = (Point*)malloc(sizeof(Point));
        cp->x = p.x;
        cp->y = p.y;
        return cp;
    }
}
```

Python Code

- When dll is loaded python has access to its functions
- functions within the dll has restype property that helps define return value type

```
import ctypes
d = ctypes.cdll.LoadLibrary(r"C:\...\TestDll")

par = ctypes.c_double(12.3)
d.double_func.restype = ctypes.c_double
res = d.double_func(par)
print(res)    #13.3
```

Python Code - cont'd

```
par = ctypes.c_char_p('hiall')
d.str_func.restype = ctypes.c_char_p
print (d.str_func(par)) # hi all
```

- ctypes has byref function for integration with C functions that receives arguments by reference
- byref works with ctypes types only .

```
a = ctypes.c_int(2); b = ctypes.c_int(5)
print ("before swap a = {}, b = {}".format( a.value, b.value))
#2,5
res = d.swap(ctypes.byref(a), ctypes.byref(b))
print ("after swap a = {}, b = {}".format((a.value, b.value)) #5,2
```

Python Code - cont'd

- ctypes module also has POINTER type for integrating C pointers d.
pointer_func.restype = ctypes.POINTER(ctypes.c_float) res=d.
pointer_func(ctypes.c_float(15))

print (res)	#<__main__.LP_c_float object at 0x01E98260>
print (res.contents)	#c_float(-15.0)
print (res.contents.value)	#15.0

```
print (res)          #<__main__.LP_c_float object at
0x01E98260>
print (res.contents)  #c_float(-15.0)
print (res.contents.value)  #15.0
```

Python Code - cont'd

```
import ctypes

class PointClass(ctypes.Structure):
    _fields_ = [
        ("x", ctypes.c_int),
        ("y", ctypes.c_int)
    ]

d.strct_func.restype = ctypes.POINTER(PointClass)
c = PointClass(100, 200)
res = d.strct_func(c)

print(res)                # <__main__.LP_PointClass object at 0x01E982B>
print(res.contents)       # <__main__.PointClass object at 0x01E98210>
print(res.contents.x, res.contents.y)  # 100 200
```

Lab 01

Lab



Questions

