Librería CTypes

Antonio Espín Herranz

Carga de DLLs

- La librería ctypes exporta cdll, windll y oledll

- Carga de librería **DLLs** en **Windows**:

```
>>> from ctypes import *
>>> print(windll.kernel32) # doctest: +WINDOWS
    <WinDLL 'kernel32', handle ... at ...>
>>> print(cdll.msvcrt) # doctest: +WINDOWS <CDLL
    'msvcrt', handle ... at ...>
>>> libc = cdll.msvcrt # doctest: +WINDOWS
>>>
```

Carga de DLLs

- En **Linux**:

```
>>> cdll.LoadLibrary("libc.so.6") # doctest: +LINUX
    <CDLL 'libc.so.6', handle ... at ...>
>>> libc = CDLL("libc.so.6") # doctest: +LINUX >>>
    libc # doctest: +LINUX
<CDLL 'libc.so.6', handle ... at ...> >>>
```

Llamada a Funciones de C

- Una vez tenemos cargada la librería podemos llamar a funciones de C:

```
from ctypes import *

# Cargar las librerías:
libc = cdll.msvcrt

# Pruebas con las funciones de C
libc.printf(b"%s %d %f\n\n", b"hola", c_int(67), c_double(8.99))
# Tenemos que utilizar funciones de conversión de tipos. Si no lanzará
    excepciones de conversión de tipos: ctypes.ArgumentError
```

También es posible definir atajos a las Funciones nativas de C:
printf = libc.printf

Tipos de datos fundamentales

ctypes type	C type	Python type
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
c_float	float	float
c_double	double	float
c_char_p	char * (NUL terminated)	string or None
c_wchar_p	wchar_t * (NUL terminated)	unicode or None
c_void_p	void *	int/long or None

Llamar a funciones con tipos personalizados

- class Bottles(object):

  def __init__(self, number):

  self.***_as_parameter_*** = number

  bottles = Bottles(42)

  printf(b"%d bottles ", bottles) # 42 bottles

  – ***ctypes*** *busca un atributo llamado* ***_as_parameter_*** *y lo utiliza para llamar a la función* ***printf****.*
  – *Puede ser de tipo int, string o Unicode.*

Especificar tipos en las funciones

- También podemos indicar cuales son los tipos de los parámetros antes de la llamada a la función.

- Por ejemplo, en una llamada a printf, especificamos los tipos y luego llamamos con los valores.

printf.argtypes = [c_char_p, c_char_p, c_int, c_double]

printf(b"String '%s', Int %d, Double %f\n", b"Hi", 10, 2.2)

String 'Hi', Int 10, Double 2.200000

Tipos de retorno

- Por defecto se asume que las funciones de C devuelven un tipo int.

- Cuando utilizamos funciones que devuelven otros tipos también lo vamos a tener que especificar, con el atributo restype.

- Ejemplo, utilizando la función **strchr** de C (*devuelve un puntero al carácter dentro de la cadena*)

```
strchr = libc.strchr
print(strchr(b"abcdef", ord("d")))  8059983
strchr.restype = c_char_p # c_char_p es un puntero a string
print(strchr(b"abcdef", ord("d"))) 'def'
print strchr(b"abcdef", ord("x"))) None
```

*Se utiliza* ***ord*** *para pasar el entero que representa la letra.*

Especificar tipos argumentos / retorno

- Se pueden especificar simultáneamente los tipos de los argumentos y el de retorno:

```
>>> strchr.restype = c_char_p
>>> strchr.argtypes = [c_char_p, c_char]
>>> strchr(b"abcdef", b"d") 'def'
```

Paso de parámetros por referencia

- Tenemos la función byref(param) para pasar un parámetro por referencia:

```
# Inicializar las variables:
i = c_int()
f = c_float()
s = create_string_buffer(b'\000' * 32)
print(i.value, f.value, repr(s.value))
libc.sscanf(b"1 3.14 Hello", "%d %f %s", byref(i),byref(f), s)
print(i.value, f.value, repr(s.value))
```

- **repr** Una representación imprimible del objeto.

Struct / Union

- Las estructuras y uniones deben derivar de las clases base **Structure** y **Union**.

- Cada subclase debe definir un atributo **_fields_** que debe ser una lista de 2 tuplas, conteniendo el nombre del campo y del tipo.

Ejemplo

```
class POINT(Structure):
    _fields_ = [("x", c_int),("y", c_int)]

point = POINT(10, 20)
print(point.x, point.y)

point = POINT(y=5)
print (point.x, point.y)
```

Anidar Estructuras

```
class RECT(Structure):
    _fields_ = [("upperleft", POINT),("lowerright", POINT)]

rc = RECT(point)
print(rc.upperleft.x, rc.upperleft.y)
print (rc.lowerright.x, rc.lowerright.y)

# Se puede inicializar así:
r = RECT(POINT(1, 2), POINT(3, 4))
r = RECT((1, 2), (3, 4))
```

Arrays

- # Inicializar un array de 10 enteros:
- TenIntegers = c_int * 10
- ii = TenIntegers(1, 2, 3, 4, 5, 6, 7, 8, 9, 10)
- print(ii)
- for i in ii: print (i, end = ‘ ‘)

Punteros

- n = c_int(40)
- pi = pointer(n)
- print (pi)
- print (pi.contents)

Cargar una DLL

- Con la librería **ctypes** también podemos cargar una DLL y llamar a sus funciones:

```
from ctypes import *

# Indicar el path de la DLL
miDLL = cdll.LoadLibrary("miDLL.dll")

suma = miDLL.sumar(c_int(3), c_int(4))
resta = miDLL.restar(c_int(8), c_int(9))

print ("Suma: ", suma)
print ("Resta: ", resta)
```

Referencias

- http://starship.python.net/crew/theller/ctypes/tutorial.html
- https://docs.python.org/2/c-api/