COMBINANDO “C” CON “PYTHON”

C es un lenguaje que ofrece una mayor velocidad de cómputo con respecto a Python. Sin embargo, en el análisis y organización de datos Python es un lenguaje muy útil. Por estas razones, conseguir usar ambos lenguajes de forma coordinada puede ser una idea interesante.
En este artículo se presenta una manera sencilla y eficiente de combinar Python con C para aprovechar las ventajas que ambos ofrecen. En estas instrucciones se mostrará el uso basado en Ctypes y Cython.

ctypes

Ctypes es una herramienta de la librería estándar de Python para crear vínculos. Permite cargar y usar librerías de C/C++ e intercambiar información entre Python y C.

cython

Cython es un compilador que permite mezclar Python y C en el mismo código, aprovechando las ventajas de velocidad que aporta C y las herramientas que ofrece Python.

Uso online a través de Colab (Ctypes & Cython)

El siguiente Colab contiene un ejemplo de código para comprobar la primalidad de un número mezclando Python y C, basado en Ctypes.

El funcionamiento para Ctypes es el siguiente:

1. El programa en Python manda el número que se quiere comprobar al programa en C.
2. El programa en C devuelve una estructura que contiene el número que se ha probado, el tiempo necesitado para el cálculo y la primalidad del número.
3. El programa en Python recibe la información devuelta por el programa en C y se imprime por pantalla.

También se incluye el programa de primalidad usando Cython en Colab.

Recomiendo hacer una copia del siguiente Colab para poder experimentar el uso de *Ctypes* y *Cython*.

Colab_Primalidad_C+Python
Para poder acceder a este notebook en Colab, debes iniciar sesión con una cuenta de la UC3M.

Uso offline (Ctypes)

Las siguientes instrucciones han sido probadas en Ubuntu 18.04 + Python 3.8. El programa viene preparado para ser usado también en Windows, pero no aseguro que funcione.

1.- Clonar repositorio: use_C_from_Python o descargar este zip.

2.- Instalar paquetes necesarios *(recomendable usar un environment de python).*

```
pip install -r requirements.txt
```

3.- Crear programa en C con sus funciones.

4.- Modificar el archivo "tasks.py".

Este archivo permite compilar y ejecutar los programas en C y Python.
Por cada programa de C o Python que se añade, se debe crear en "tasks.py" un "@invoke.task()"

- **Copiar y modificar la tarea ya definida para el programa de C:**

```
@invoke.task()
def build_functions(c, path=None):
…
```

  - ❏ Modificar el nombre `build_functions` de la definición por otro.
  - ❏ Dentro de esta función `build_functions` se debe modificar:
    - 2 veces el nombre del programa de C (`functions.c`) , usando el nombre del programa en C creado.
    - 1 vez nombre del archivo compilado de C (`functions.o`) , usando el nombre del programa en C creado.
    - 1 vez nombre de la librería que quieras darle (`libfunctions.so`)

    El número de veces significa que dicho nombre aparece varias veces dentro de la función

- **Copiar y modificar la tarea ya definida para el programa de Python:**

```
@invoke.task()
def test_ctypes(c):
…
```

  - ❏ Modificar el nombre `test_ctypes` de la definición por otro.
  - ❏ Dentro de esta función `test_ctypes` se debe modificar:
    - el nombre del programa de Python que se vaya a crear para ejecutar funciones de C (`ctypes_test.py`)

- **Editar el siguiente párrafo introduciendo las 2 funciones nuevas definidas antes para el programa en C y Python:**

```
@invoke.task(
    clean,
    build_functions,
    test_ctypes,
    …,
    …
)
```

5.- Crear un programa en Python con el nombre definido en el paso 4, y siguiendo la estructura de ejemplo de "ctypes_test.py"

En este programa en Python, se cargará al principio del main la librería en C que se genera a través de "invoke".

Posteriormente, basta con llamar a las funciones en C a través de "`c_lib.xxxx`". Por ejemplo, "`r = c_lib.csum(2,5)`", donde la variable "`r`" contendrá el resultado de la función en C.

6.- ¿Cómo se ejecuta y compila todo? A través de "invoke". Ejemplo de usos:

- **`invoke --list`** Muestra todos los archivos que puede compilar

```
all                 Build and run all tests
build-functions     Build the shared library for the C code
clean               Remove any built objects
test-ctypes         Run the script to test ctypes
```

- **`invoke all`** Compila y ejecuta todos los archivos. En este caso, compila build-functions y después ejecuta el programa de Python test-ctypes.
- **`invoke build-functions`** Compila build-functions
- **`invoke test-ctypes`** Ejecuta el programa de Python
- **`invoke clean`** Limpia archivos generados

En "tasks.py" siempre se usa guión bajo (_), mientras que al llamar "invoke ..." se usan guiones (-). Invoke convierte automáticamente guión bajo (_) a guión (-) porque en Python no se pueden usar guiones para nombres

Posibles problemas al ejecutar invoke:

- Si aparece algún error para encontrar la librería en C compilada, es posible que el problema sea la dirección del directorio..
    - En el archivo "`ctypes_test.py`", revisa la ruta definida donde se encuentra la librería compilada de C
      `lib_path = os.path.join (cur_path, "libfunctions.so")`

Comparación tiempos Python+C (ctypes) vs Cython vs Python

Se ha ejecutado el programa de primalidad para los mismos números usando Python+C y usando solo Python. Todas las pruebas han sido realizadas en Colab.

Las diferencias son muy notables y se ve claramente como la ejecución en Python+C es mucho más rápida que usando sólo Python.

A su vez, Cython consigue unos resultados muy similares a C, por lo que puede ser un camino intermedio entre C y Python. Es posible que para otros programas diferentes al cálculo de la primalidad, la diferencia entre usar Ctypes (Python+C) y Cython sea mayor. Pero en cualquier caso, Cython normalmente aportará una gran ventaja con respecto a Python.

Número	Primo	Tiempo (s) Python + C	Tiempo (s) Cython	Tiempo (s) Python
776159	Si	6.00e-06	9.29e-06	0.0001137
776161	No	6.00e-06	6.91e-06	0.0001089
98982599	Si	5.19e-05	7.70e-05	0.0011548
98982601	No	5.90e-05	7.62e-05	0.0008313
9984605927	Si	0.000472	0.0007998	0.0103352
9984605929	No	0.000474	0.0007572	0.010947
999498062999	Si	0.004575	0.0059542	0.107771
999498063001	No	0.004681	0.0054001	0.10662
99996460031327	Si	0.045364	0.0541632	1.05895
99996460031329	No	0.045464	0.0537860	1.05291
9999940600088207	Si	0.468416	0.543366	10.5446
9999940600088209	No	0.477073	0.545745	10.4499
999999594000041207	Si	4.631327	5.456306	103.6095
999999594000041209	No	4.5989	5.452800	103.4551
4611685283988009527	Si	9.9139	11.67639	283.4044
4611685283988009529	No	9.9197	11.70953	279.2823
9223371593598182327	Si	14.036	16.51730	411.5495
9223371593598182329	No	14.026	16.56543	411.7476

Referencias:

https://realpython.com/python-bindings-overview/#ctypes