Formation Module Extension Python c/c++ Partie 1

Ihab ABADI / UTOPIOS

Sommaire

- Nécessité d'extension de Python par c/c++.
- Les modules pour l'interaction de Python avec d'autre langages.
  - Python C API
  - Cython
  - Ctypes
- Types python vs c/c++
- Utilisation des méthodes avec args
- Utilisation des pointeurs.
- Utilisation des structures

Nécessité d'extension de Python par c/c++.

- Pour étendre les fonctionnalités de Python
- Pour améliorer les performances
- Pour utiliser Python comme langage de collage
- Pour créer des liaisons Python pour une bibliothèque

Python C API

- Python C API est une librairie qui permet de développer des extensions Python En C ou C++
- Python-dev fournit des structures C pour exécuter des fonctionnalités C.
- Nécessite des connaissances de développement en C.
- Un exemple.

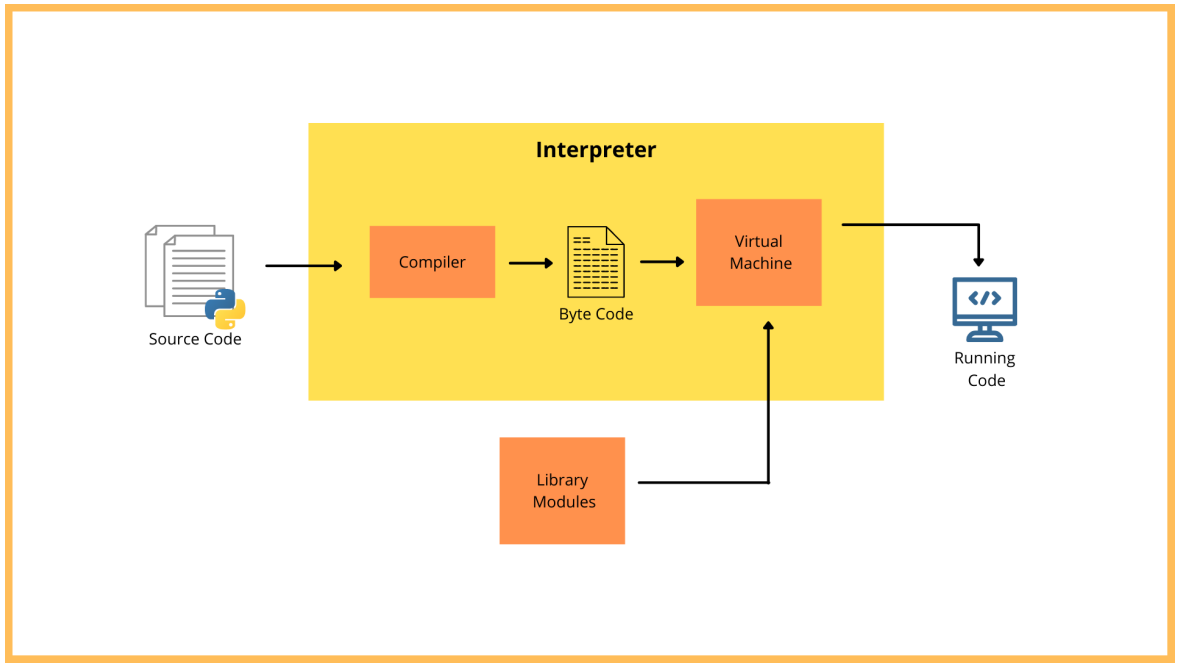
Cython

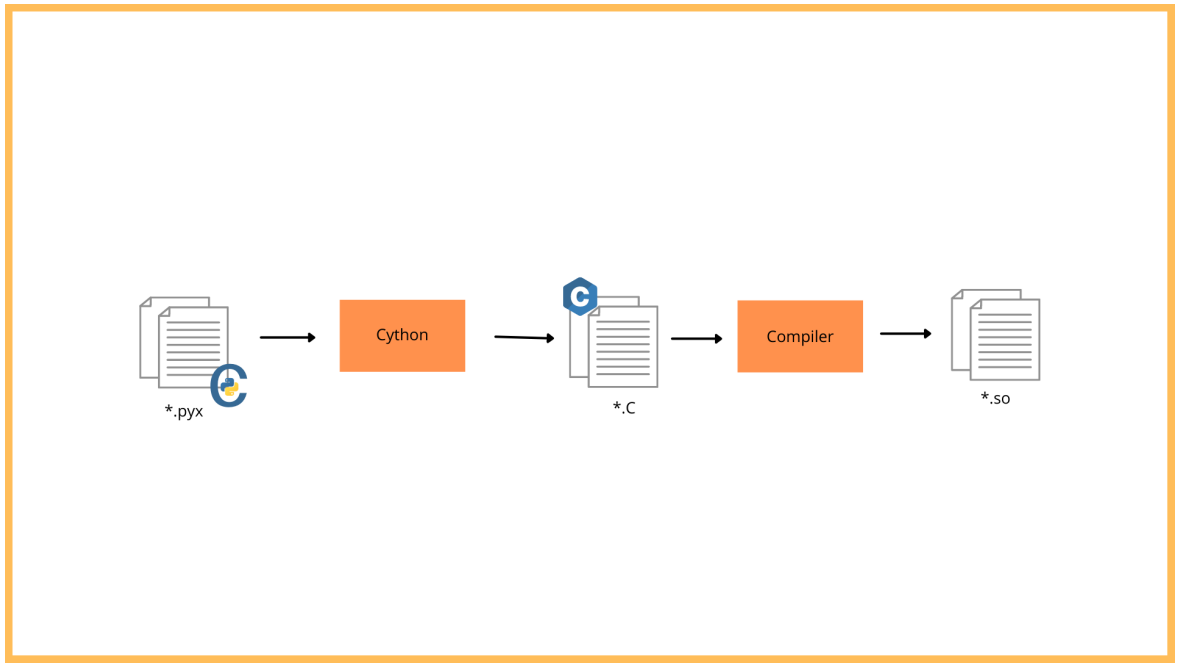
- Cython peut être considéré à la fois comme un module et comme un langage de programmation qui étend (en quelque sorte) Python en permettant l'utilisation du typage statique emprunté à C/C++.
- Tout le code Python est valide Cython, mais pas l'inverse.
- Vous pouvez directement copier votre code Python existant dans un fichier Cython, puis le compiler pour améliorer les performances.

Cython

- Python est plus efficace que C étant donné qu'il s'agit d'un langage de haut niveau.
- Python est efficace mais lent. C, en revanche, est moins efficace mais plus rapide que Python.
- Cython vise donc à apporter tous les avantages de C à Python tout en maintenant l'efficacité de Python.

Cython


Interpreter
Source Code
Compiler
Byte Code
Virtual Machine
Running Code
Library Modules


*.pyx
Cython
*.C
Compiler
*.so

Cython

- Utiliser Cython partout ne garantit pas toujours une vitesse accrue.
- vous pouvez l'utiliser dans des fonctions qui impliquent un grand nombre d'opérations mathématiques et d'itérations de boucle..
- Un autre cas d'utilisation intéressant est lorsque vous disposez déjà d'une bibliothèque C ou C++ nécessitant une interface Python. Dans ce cas, vous pouvez utiliser Cython pour créer un wrapper pour la bibliothèque.

Cython – Démo Python vs Cython

```python
import time

def demo_python():
    t1 = time.time()
    total = 0
    for k in range(1000000000):
        total = total + k
    print "Total =", total

    t2 = time.time()
    t = t2 - t1
    print("%.20f" % t)
```

```python
import time
def demo_cython():
    cdef unsigned long long int total
    cdef int k
    cdef float t1, t2, t

    t1 = time.time()

    for k in range(1000000000):
        total = total + k
    print "Total =", total

    t2 = time.time()
    t = t2 - t1
    print("%.100f" % t)
```

Exercice Cython

Exercice

- Créer une fonction en Cython pour calculer La suite de Fibonacci et comparer le temps d’exécution à une fonction en Python

Cython – Déclaration de variable

- Déclarations variables typées statiquement
- Mot clef cdef permet déclaration statique
- Typage proche du C
- Variables typées statiques se comportent comme variables C;
- L'opérateur = effectue des copies comme en C
- Déclaration simple : cdef double x
- Déclaration simple avec initialisation : cdef double y=0.
- Déclarations multiples : cdef int i, j = 0, k

Cython – Déclaration de variable

- Exemples de déclarations
  Type C ⬅=➔ Déclaration Cython
- Pointeurs ⬅=➔ cdef int* pt_i
- Tableau statique ⬅=➔ cdef float tab[3]
- Pointeurs de fonctions ⬅=➔ cdef void (*f)(int,double)

Correspondance entre les types python et les types C

Type Python	Type C
bool	bint
int	[unsigned] char
long	[unsigned] short, int, long, long long
float	float, double, long double
complex	float/double complex
bytes	char*
str	std::string (C++)
unicode	
dict	struct

Cython – Déclaration de fonctions

- On déclare une fonction C à l'aide du mot clef cdef;
- Génère une fonction pure C
- On peut y manipuler des objets Python mais antagoniste à l'idée d'optimisation
- Ne peut pas être appelée par une fonction Python non définie via Cython
- Possibilité d'inline pour la fonction

Cython – Gestion d'exceptions

- Une fonction Python retourne toujours un objet Python : permet de gérer facilement les exceptions
- Pour les fonctions C ou C/Python : pas possible de remonter l'exception à l'appelant;
- Il faut utiliser une clause d'exception : soit retourner un entier particuliers soit Cython gère une exception (plus couteuse)

```
cpdef int divide_ints(int i, int j) except ? -1:
    return i / j

cpdef int divide_ints(int i, int j) except *:
    return i / j
```

Cython

- Cython utilise un garbage collector:
- Compteur de référence
- G.C détruit les objets sans références périodiquement
- Démo

Exercice Cython

Exercice

- Créer une fonction en Cython pour trouver les nombres premiers et comparer le temps d'exécution à une fonction en Python

Exercice Cython

Exercice

- Développer une application en Cython qui permet de parcourir un dossier et fusionner la totalité des fichiers csv d’un sous dossier dans un seul fichier.

Parallélisme multithreading et le GIL

- Le GIL (Global Interpreter Lock) oblique qu'un seul thread principal exécute du Bytecode Python;
- Le GIL est nécessaire seulement pour aider à la gestion mémoire des objets python;
- Du code C ne travaillant pas avec des objets Python peuvent être exécuter sans le GIL;
- On peut demander à Cython d'outrepasser le GIL pour du parallélisme dans certaines parties du code;
  Pour cela, il faut s'assure de ne pas utiliser ou retourner des objets Python

Cython

- Cython nous permet également d'appeler des fonctions ou librairies c/c++ dans un script python.
- Démo

Exercice Cython

Exercice

- Créer un script cython qui permet d'exécuter la fonction c suivante:

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


void getDirectory(const char* folder) {
  DIR *d;
  struct dirent *dir;
  d = opendir(folder);
  if (d) {
    while ((dir = readdir(d)) != NULL) {
      printf("%s\n", dir->d_name);
    }
    closedir(d);
  }
}
```