Formation Module Extension Python c/c++ Partie 1 & 2

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.

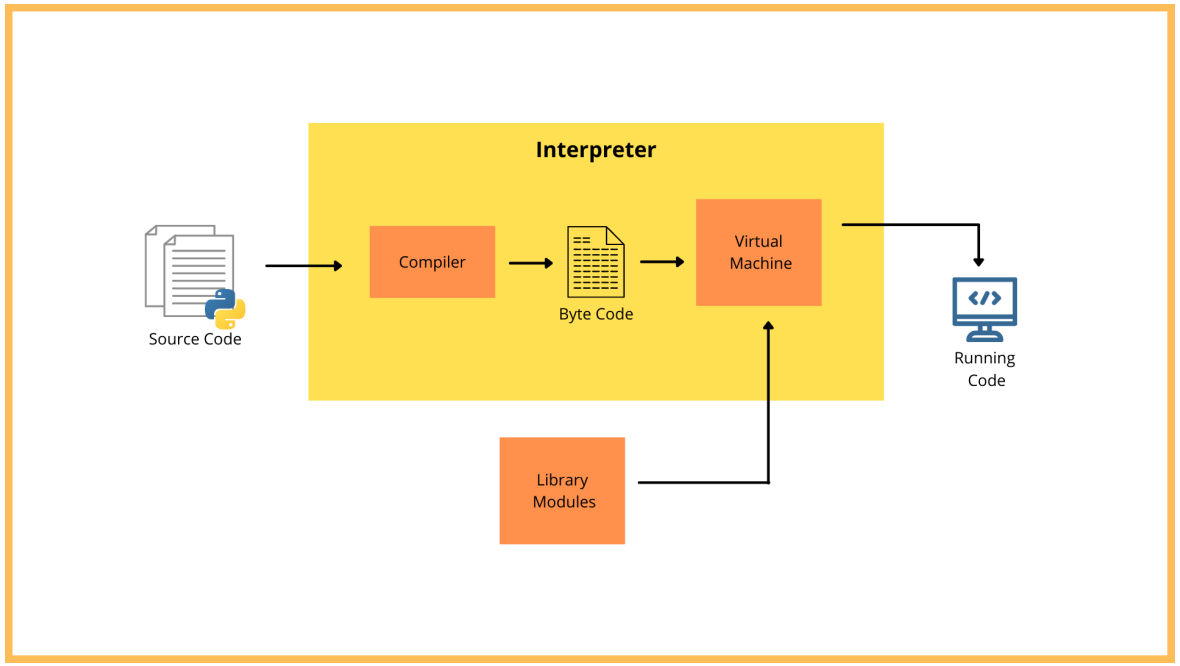
Cython

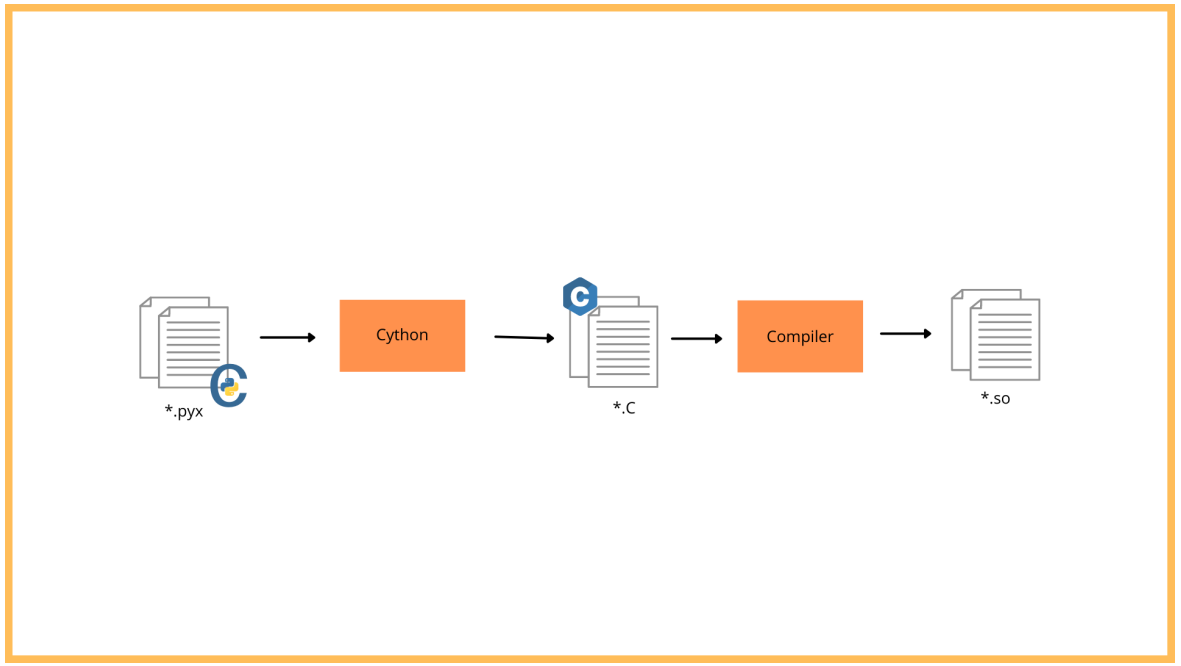
- Cython peut être considéré à la fois comme un module et comme un langage de programmation qui étend (en quelque sorte) Python en permettant l'utilisation du typage statique emprunté à C/C++.
- Tout le code Python est valide Cython, mais pas l'inverse.
- Vous pouvez directement copier votre code Python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

```
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)
```

```
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);
  }
}
```

CTYPES

- ctypes est une bibliothèque de fonctions étrangères pour Python.
- Il fournit des types de données compatibles C et permet d'appeler des fonctions dans des DLL ou des bibliothèques partagées.
- Il peut être utilisé pour envelopper ces bibliothèques en Python
- ctypes premet de charger des bibliothèques à l’aide de l’objet cdll
- Pour windows c’est windll et oledll

CTYPES – Demo 1

- Code C => fichier demo.c

```
float add_float(float a,float b)
{return a+b;}
```

- Code Python

```
from ctypes import *

demo_c = cdll.LoadLibrary("demo.so")
## => demo.dll pour windwos
#demo_c = windll.LoadLibrary("demo.dll")

print(demo_c.add_int(10,20))
```

CTYPES

- Ctypes nous permet de définir le types des paramètres avant de les injecter dans les fonctions fournit par la dll.
- Exemple avec une méthode qui accepte des Floats comme arguments:

```
from ctypes import *

demo_c = cdll.LoadLibrary("demo.so")
## => demo.dll pour windwos
#demo_c = windll.LoadLibrary("demo.dll")

demo_c.add_float.restype = ctypes.c_float
print(demo_c.add_float(c_float(10.0), c_float(20.0)))
```

CTYPES VS C Types

ctypes type	C type	Python type
c_bool	_Bool	bool (1)
c_char	char	1-character bytes object
c_wchar	wchar_t	1-character string
c_byte	char	int
c_ubyte	unsigned char	int
c_short	short	int
c_ushort	unsigned short	int
c_int	int	int
c_uint	unsigned int	int
c_long	long	int
c_ulong	unsigned long	int
c_longlong	__int64 or long long	int
c_ulonglong	unsigned __int64 or unsigned long long	int
c_size_t	size_t	int
c_ssize_t	ssize_t or Py_ssize_t	int
c_float	float	float
c_double	double	float
c_longdouble	long double	float
c_char_p	char * (NUL terminated)	bytes object or None
c_wchar_p	wchar_t * (NUL terminated)	string or None
c_void_p	void *	int or None

Exercice

Exercice

- Soit la bibliothèque cart.so (cart.dll) qui permet d'exécuter les méthodes suivantes :
- is_correct_product qui accepte un argument de type int et renvoie un bool
  - Cette fonction renvoie vrai si l'argument est strictement supérieur à 10 et pair, faux dans les autres cas
- total_item qui accepte deux arguments, le premier est int et le deuxième un flaot, et renvoie un float.
  - Cette fonction renvoie 0 si un des arguments est négatif.
- Ecrire en python et à l'aide du framework pytest un ensemble de fonction test pour tester notre librairies

CTYPES

- Ctypes permet de définir des arguments à passer par référence.
- Le passage par référence se fait à l'aide de la méthode byref de ctypes.
- Nous pouvons utiliser le passage byref dans le cas ou la fonction attend un pointeur.
- Démo

```
a = ctypes.c_float(10)
b = ctypes.c_float(20)
c = ctypes.c_float()
result = demo_c.add_float_ref(ctypes.byref(a), ctypes.byref(b),
ctypes.byref(c))
print(c.value)
```

CTYPES

- Ctypes permet de définir des arguments pointeurs également.
- La définition d'un argument pointeur se fait à l'aide de la méthode pointer de ctypes.
- Démo

```
a = ctypes.c_float(10)
b = ctypes.c_float(20)
c = ctypes.c_float()

a_pointer = ctypes.pointer(a)
b_pointer = ctypes.pointer(b)
c_pointer = ctypes.pointer(c)
demo_c.add_float_ref(a_pointer,b_pointer, c_pointer)
```

Exercice

Exercice

- Soit la bibliothèque ex_2.so (ex_2.dll) qui permet d'exécuter les méthodes suivantes :
- concat_two_string qui accepte deux arguments de type pointeurs de char et renvoie un pointeur de char
- Ecrire en python et à l'aide du framework pytest un ensemble de fonction test pour tester notre librairies

CTYPES

- Ctypes permet de définir des arguments sous forme de tableau.
- Le passage des tableaux peut se faire de la même façon que les pointeurs.
- Démo

```
list1 = [1,4,5,6]
list2 = [5,6,8,9]
tab1 = (ctypes.c_int * len(list1))(*list1)
tab2 = (ctypes.c_int * len(list2))(*list2)
result = (ctypes.c_int * len(list2))(0,0,0,0)
demo_c.add_two_array(tab1, tab2, result, len(list1))
list = [result[i] for i in range(len(list1))]
print(list)
```

Exercice

Exercice

- Soit la bibliothèque ex_3.so (ex_3.dll) qui permet d'exécuter les méthodes suivantes :
- Reverse qui accepte un premier argument de type tableau int et un deuxième argument qui correspond à la taille du tableau et fait un reverse du tableau.
- Ecrire en python et à l'aide du framework pytest un ensemble de fonction test pour tester notre librairies

CTYPES

- Ctypes permet de passer également des structures aux fonctions développées en C
- La définition d'une structure se fait en créant une classe qui hérite de classe Structure de ctypes.
- Les champs de la structure sont à définir dans les _fields_ de la classe.
- démo

Exercice

Exercice

- Soit la bibliothèque ex_4.so (ex_4.dll) qui permet d'exécuter les méthodes suivantes :
- Calcule_total qui accepte comme arguments, un tableau de produit panier, la taille du tableau, et qui renvoie le total du panier en float.
  - Un produit panier est une structure avec un int qty et float price.
  - Cette fonction renvoie 0 si un des produit panier possède une qty ou un prix négatif.
- Ecrire en python et à l'aide du framework pytest un ensemble de fonction test pour tester notre librairies

Exercice

Exercice

- Soit la bibliothèque ex_4.so (ex_4.dll) qui permet d'exécuter les méthodes suivantes :
- Calcule_total qui accepte comme arguments, un tableau de produit panier, la taille du tableau, et qui renvoie le total du panier en float.
  - Un produit panier est une structure avec un int qty et float price.
  - Cette fonction renvoie 0 si un des produit panier possède une qty ou un prix négatif.
- Ecrire en python et à l'aide du framework pytest un ensemble de fonction test pour tester notre librairies

TP - chuck-norris – CodinGame puzzle

TP

- Soit la bibliothèque chuck-norris.so (chuck-norris.dll) qui permet d'exécuter les méthodes suivantes :
  - toBinCode qui accepte comme argument un char et renvoie un pointeur de char qui correspond au binaire du caractère.
  - convert_to_chuck_norris qui accepte comme argument une chaine de caractère qui se termine par un retour à la ligne « \n » et qui renvoie l'unaire de chuck norris.
- Ecrire en python et à l'aide du framework pytest un ensemble de fonction test pour tester notre librairies.
- Rappel du puzzle chuck norris:
  - Le message en entrée est constitué de caractères ASCII (7 bits)
  - Le message encodé en sortie est constitué de blocs de 0
  - Un bloc est séparé d'un autre bloc par un espace
  - Deux blocs consécutifs servent à produire une série de bits de même valeur (que des 1 ou que des 0) : - Premier bloc : il vaut toujours 0 ou 00. S'il vaut 0 la série contient des 1, sinon elle contient des 0
    - Deuxième bloc : le nombre de 0 dans ce bloc correspond au nombre de bits dans la série