Jampp <> ECI

Cython

Valentin Paz Marcolla
Software Engineer

Agenda

Cython

- ¿Qué es? ¿Para qué sirve?

Distancia Levenshtein

- Definición y solución
- Pure python
- Pure C
- Cython
- Tipos y Cython
- Chequeo de bordes

De yapa

- Comparación con Pypy (HLL)
- Casos de uso (clases, memory views, GCC functions)

Cython
Performance C en Python?!?!

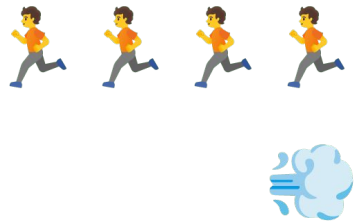
1

Compilar lo que se pueda

2

Interoperar con CPython

3



Cython
Proceso de compilación

```python
def levenshtein(seq1, seq2):
    size_x = len(seq1) + 1
    size_y = len(seq2) + 1
    matrix = [[0] * size_y for _ in range(size_x)]

    for x in range(size_x):
        matrix[x][0] = x
    for y in range(size_y):
        matrix[0][y] = y
```

```
__pyx_t_2 = PyList_New(0); if (unlikely(!__pyx_t_
__Pyx_GOTREF(__pyx_t_2);
__pyx_t_3 = __Pyx_PyObject_CallOneArg(__pyx_built
__Pyx_GOTREF(__pyx_t_3);
if (likely(PyList_CheckExact(__pyx_t_3)) || PyTup
  __pyx_t_4 = __pyx_t_3; __Pyx_INCREF(__pyx_t_4);
  __pyx_t_5 = NULL;
} else {
  __pyx_t_1 = -1; __pyx_t_4 = PyObject_
  __Pyx_GOTREF(__pyx_t_4);
  __pyx_t_5 = Py_TYPE(__pyx_t_4)->tp_i
}
```

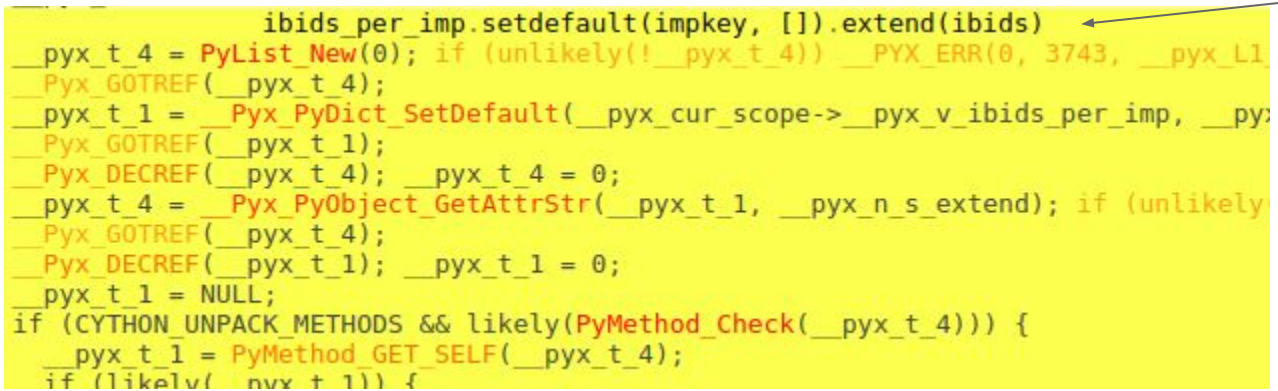
Se usa cython para traducir el código Python a C

Se compila a un .so

```
0000000 0000 0001 0001
0000010 0000 0016 0000
0000020 0000 0001 0004
0000030 0000 0000 0000
```

El ejecutor de Python (cpython) linkea las bibliotecas que tiene compiladas

Cython



Resultado del annotate

Sugerencias o instrucciones para Cython

- Tipos
- Nullificable
- Chequeo de bordes
- Objetos const
- Llamada a funciones en C
- Llamada a funciones del compilador
- Manejo de errores
- Interlineado de funciones (inlining)
- C Imports
- Clases cythonizadas (!!!)
- Métodos solo usables desde Cython
- Métodos públicos y privados

Distancia Levenshtein

Distancia Levenshtein
Las operaciones

- **Inserciones:**

  *Ola => **H**ola*

  *Pesa => **S**pesa => S**o**pesa => Sopesa**r***

- **Eliminaciones:**

  *Hola => **l**ola*

  *Sopesar => S**l**pesar => Spesa**l** => **l**Pesa*

- **Sustituciones:**

  *Todos => Tod**a**s*

  *Ingerir => In**j**erir => Inj**u**rir => Injuri**a***

Distancia Levenshtein
La definición

Dadas dos cadenas A y B se define la distancia de Levenshtein (*d(A, B)*) como la mínima cantidad de operaciones que se deben hacer hasta que las cadenas sean iguales. Las operaciones pueden ser cualquiera de las anteriores (inserción, eliminación y sustitución).

1. **Todos => Todas** } d(Todos, Todas) = 1

1. **AD => ABD**
2. **ABD => ABCD**

} d(ABCD, AD) = 2

1. **ABCD => ABD**
2. **ABD => AD**

} d(ABCD, AD) = 2

1. **ABCD => ABD**
2. **ABD => AD**
3. **AD => AE**

} d(ABCD, AE) = 3

Distancia Levenshtein
La solución (recursiva)

|A| = N y |B| = M

d(A, B) = d(A[0:N), B[0:M))

d(A[0:i), “”) = i

d(“” , B[0:j)) = j

d(A[0:i), B[0:j)) = distancia del paso anterior (i - 1 ó j - 1)

+ el costo de hacer alguna operación en éste paso

Distancia Levenshtein
La definicion (recursiva)

$$\text{si } i = 0 \rightarrow j$$

$$d(A[0..i], B[0..j]) =$$

$$\text{“”} \rightarrow \text{“1”} \rightarrow \cdots \rightarrow \text{“123}\ldots\text{j”} \qquad \text{j inserciones}$$

$$\text{“123}\ldots\text{j”} \rightarrow \ldots \rightarrow \cdots \rightarrow \text{“123}\ldots\text{j”} \qquad \text{0 modificaciones}$$

Distancia Levenshtein
La definicion (recursiva)

$$d(A[0..i], B[0..j]) = \begin{cases} \text{si } i = 0 \rightarrow j \\ \text{si } j = 0 \rightarrow i \end{cases}$$

$$\text{“}123\ldots\text{i”} \rightarrow \ldots \rightarrow \cdots \rightarrow \text{“}123\ldots\text{i”}$$

0 modificaciones

$$\text{“”} \rightarrow \text{“}1\text{”} \rightarrow \cdots \rightarrow \text{“}123\ldots\text{i”}$$

i inserciones

Distancia Levenshtein

La definicion (recursiva)

$$d(A[0..i], B[0..j]) = \begin{cases} \text{si } i = 0 \rightarrow j \\ \text{si } j = 0 \rightarrow i \\ \\ \text{si } i > 0 \wedge j > 0 \rightarrow \end{cases} \qquad d(A[0..i-1], B[0..j-1]) + \mathbb{1}\{A[i] \neq B[j]\}$$

Corresponden al mismo lugar en la cadena.
Pueden ser iguales o no.

Distancia Levenshtein

La definicion (recursiva)

$$d(A[0..i], B[0..j]) = \begin{cases} \text{si } i = 0 \rightarrow j \\ \text{si } j = 0 \rightarrow i \\ \\ \text{si } i > 0 \wedge j > 0 \rightarrow & d(A[0..i-1], B[0..j-1]) + \mathbb{1}\{A[i] \neq B[j]\} \\ & d(A[0..i], B[0..j-1]) + 1 \end{cases}$$

$$\begin{array}{rr} \ldots|A[i-1]|A[i] \rightarrow & \ldots|A[i]|B[j] \\ \ldots|B[j-1]|B[j] \rightarrow & \ldots|B[j-2]|B[j-1]|B[j] \end{array}$$

Distancia Levenshtein

La definicion (recursiva)

$$d(A[0..i], B[0..j]) = \begin{cases} \text{si } i = 0 \to j \\ \text{si } j = 0 \to i \\ \\ \text{si } i > 0 \wedge j > 0 \to \min \begin{cases} d(A[0..i-1], B[0..j-1]) + \mathbb{1}\{A[i] \neq B[j]\} \\ d(A[0..i], B[0..j-1]) + 1 \\ d(A[0..i-1], B[0..j]) + 1 \end{cases} \end{cases}$$

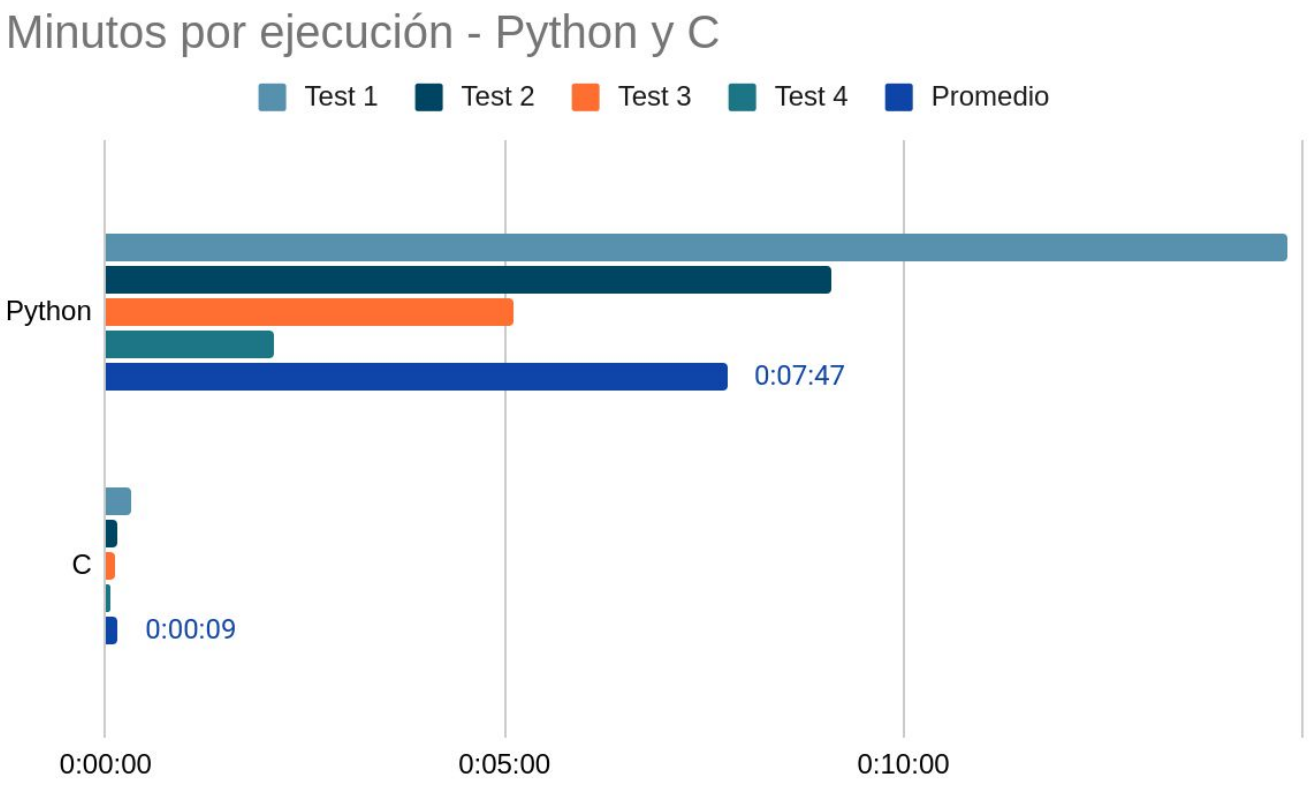
$$\ldots |A[i-1]|A[i] \to \ldots |A[i-2]|A[i-1]|A[i]$$

$$\ldots |B[j-1]\ B[j] \to \ldots |B[j-1]|B[j]$$

Implementaciones
Python y C
PD bottom-up

Demo

Resultados Python y C



C es **50x** más rápido que Python

Hinting
Tipos

PyInt_FromSsize_t

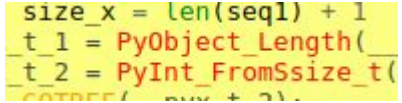
PyObject

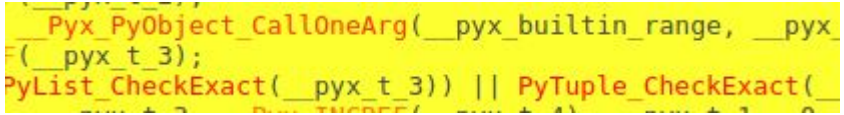
PyNumber

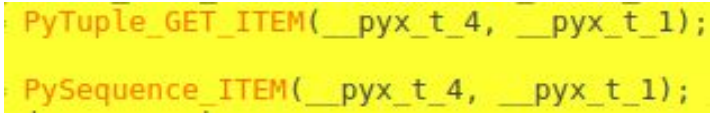
PyList_CheckExact

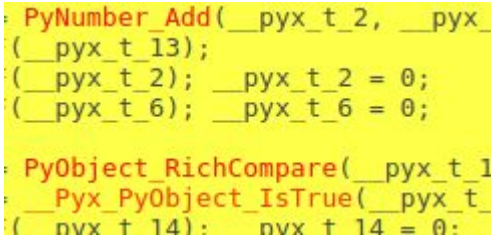
PyObject_CallOneArg

Son tipos de Python poco específicos, no nativos. Para hacer indexing, sumas y restas son lentos.

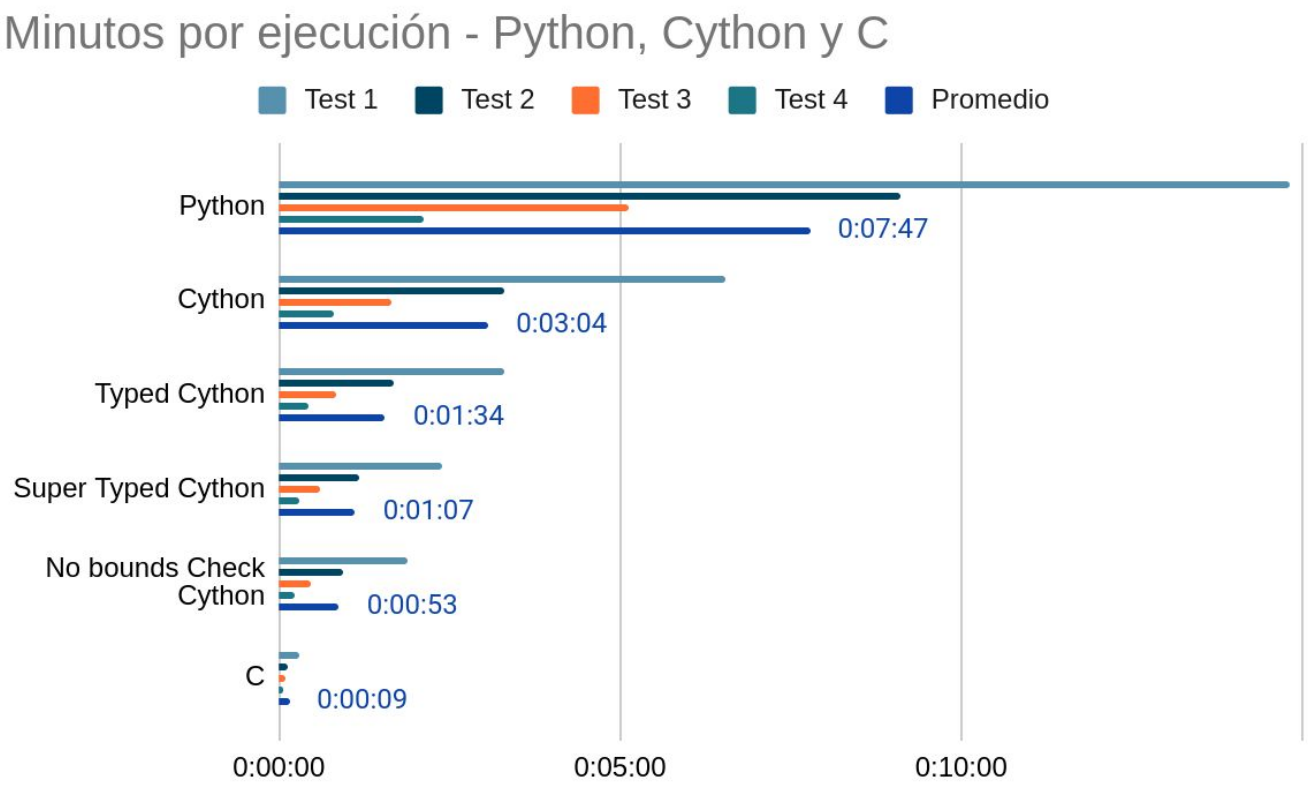








Resultados Cython



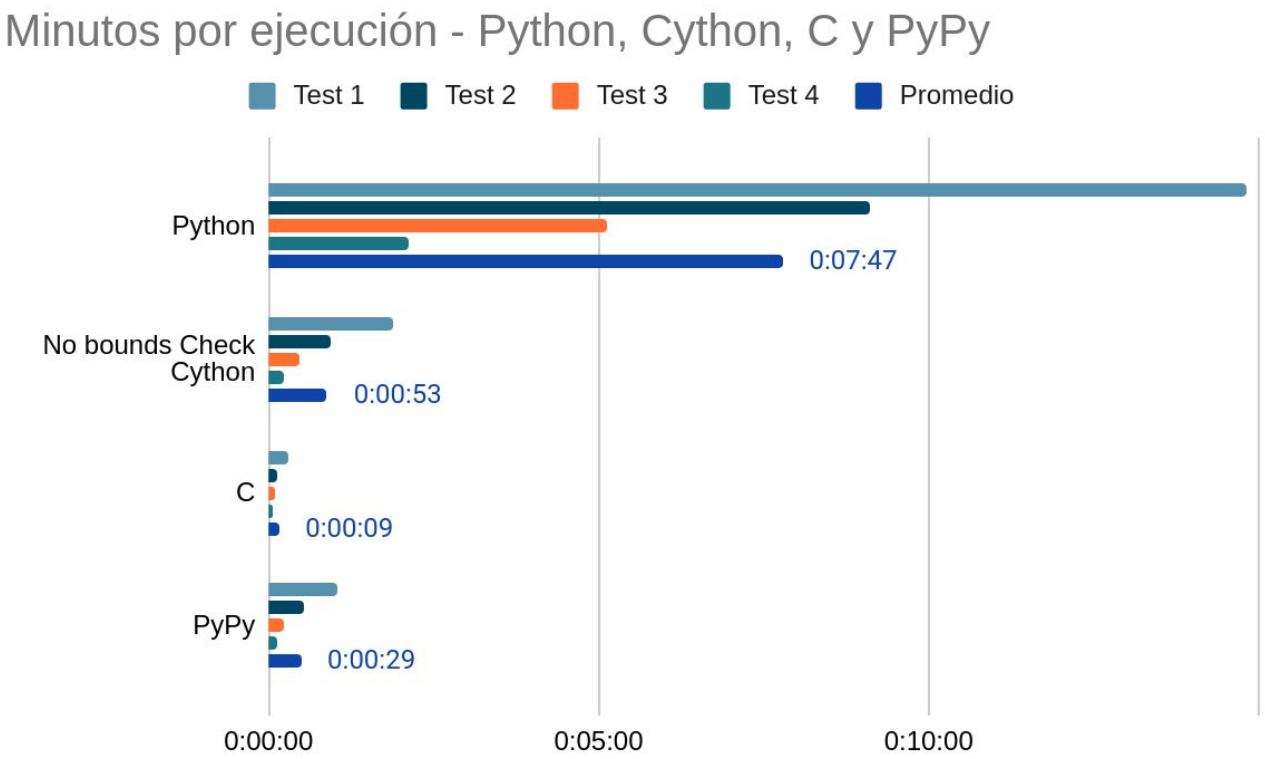
Cython es **2.5x** más rápido que Python

Cython tipado es **1.56x** más rápido que Cython

La versión final en Cython es **1.77x** más rápida que Cython tipado.
Y solo **5.8x** más lenta que C.

Con Cython mejoramos **8.8x** la perfomance de Python.

Contra PyPy



PyPy es un intérprete de Python con compilación **Justo-A-Tiempo** basado en **metatrazas**.

Es **16x** más rápido que Python, **1.8x** que Cython y **3.2x** más lento que C.

¡Pero ojo!
No funciona bien siempre.

Preguntas?

Cython++

Clases

```
cdef class ConjuntoAproximado:
    cpdef public int seed
    cpdef int _closed
    cdef unsigned char[:] elementos

    cdef inline _insertar(self, int indice, int valor, bint rebalancear)
    cpdef public insertar_elementos(self, elementos)
    cpdef public insertar_elemento(self, elementos, rebalancear=?)
```

Una llamada a insertar_elemento se transforma en una llamada en C, sin costo extra.

Y una llamada a _insertar se compila en linea.

Ocupan memoria como si fueran una struct de C.

Usar atributos sin costo natural de Python.

En casos simples: sin conversiones de tipos o comparaciones complejas.

```
+318:         if self._closed == 0:
    __pyx_t_1 = ((__pyx_v_self->_closed == 0) != 0);
    if (__pyx_t_1) {
  /* … */
    }
+319:             self._closed = 1
      __pyx_v_self->_closed = 1;
```

Cython > PyPy

(a veces)

HyperLogLog
Exprimiendo NumPy

Python: 66s

Pypy: 331s

PyPy es **48X** más lento que CPython

Cython: 7.4s

9.4X más rápido que CPython

NumPy no funciona bien con PyPy

Si las trazas son muy largas (profundas) PyPy se paga una penalidad demasiado alta

```
@cython.cfunc
@cython.inline
@cython.locals(length=c
@cython.returns(int)
def _compute_value(hash,
    n_hash = uint64(hash)
    value = (n_hash << le
    # should count the nu
    if not value:
        return 65

    if cython.compiled:
        lz = clz(value)
```

```
cdef inline int clz(unsigned long long x):
    return __builtin_clzll(x)
```

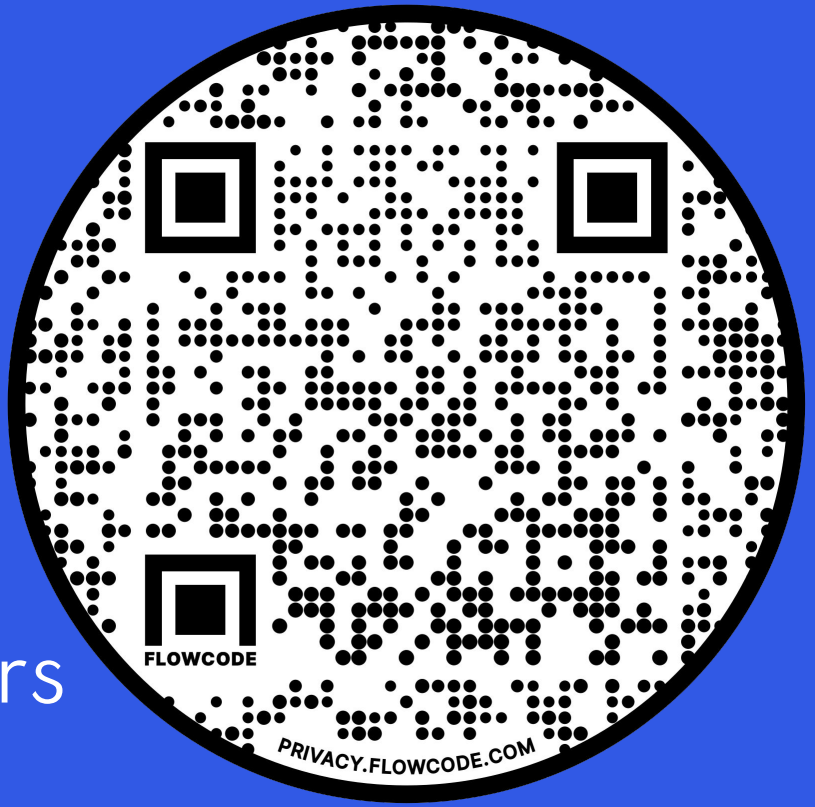
Usando funciones de GCC, inlining, cfunc y mucho Numpy.

¿Preguntas (bis)?

We are hiring!

www.jampp.com/jobs

Data Analysts
ML Engineers
Performance Engineers
Data Engineers
Cloud Engineers
Full Stack Engineers
…..& more!



Gracias!