CtoPython

David Peña Peralta

2023-09-09

Table of contents

Preface **3**

1 Nuevo lenguaje, nuevas reglas. **4**
1.1 El punto y coma ;. 4
1.2 Las llaves {}. 5
1.2.1 Dos puntos : 5
1.3 for en Python 5
1.3.1 Indentación. 5
1.4 PEP-8 6

2 Estructura de un Código. **7**
2.1 El préambulo. 7
2.1.1 `#include<...>` 7
2.2 `#define` 8
2.3 `struct` 8
2.3.1 El contenido. 8

References **9**

Preface

El lenguaje de programación de C, es uno de los lenguajes más antiguos (Salio en 1969)

1 Nuevo lenguaje, nuevas reglas.

La primera cosa a considerar al cambiar de lenguaje de programación son las nuevas reglas. Con reglas nos referimos a la escritura de código en general. En C por ejemplo, somos conscientes que toda linea debe terminar en un `;` . El código de cada `entorno` (for, while, switch, funciones, entre otros.) se delimitan por llaves `{ }` (por ejemplo, for(…){}), sus argumentos se encierran entre paréntesis `( )`. Entonces repasaremos los cambios que se presentan Python.

Comenzaremos mostrando él primer código que se realiza aprendiendo en C.

```c
#include <stdio.h>

int main(){
    printf("Hello World");
    return 0;
}
```

Ahora, veríamos el mismo código realizado en Python.

```python
print("Hola Mundo")
```

```
Hola Mundo
```

Ambos códigos hacen lo mismo, sin embargo notamos muchos cambios a simple vistas. Por el momento ignoraremos el preámbulo, y la función `int main()`.

1.1 El punto y coma ;.

En C, el punto y coma `;` es ley y me atrevo a opinar que es el mayor causante errores de compilacion. Pues bien, en Python estos desaparecen por "completo".

Notamos entonces que la función `print()` de Python, (la única línea del código de Python) no posee un punto y coma, a pesar que su "similar" en C `printf()` si lo tiene.

El punto y coma no te generará error, ya que puede usarse para separar líneas de código. En general, para tus códigos de Python, puedes prescindir del `;`.

1.2 Las llaves {}.

En la función `int main()` notamos que su contenido encuentra encerrado por llaves `{}`. Esto se aplica para funciones creadas por el usuario y para las funciones por defecto (`for()`, `switch()`, entre otros). En Python, las llaves desaparecen por "completo", entonces es sencillo preguntarse *¿Cómo Python delimita las funciones?*.

1.2.1 Dos puntos :

Así como en C, el punto y coma `;` es ley. En Python es el dos puntos `:`, pero se usa de forma diferente, en las funciones básicas. Por ejemplo

- En C tenemos `for(...){`
- En Python tenemos `for[...]:`.

1.3 for en Python

El `for` en Python no usa corchetes `[]`, solo señala que hay algo antes de los dos puntos, más adelante veremos como se traslada el `for` de C a Python.

1.3.1 Indentación.

Ya vimos como abrir una función (`:`), para separar el bloque de código de la función del resto del código se usa la indentación (tecla 'tab'). Es decir,

codigo base p1 función codigo de función codigo base p2

En este "ejemplo", vemos que el código de la función tiene una indentación, mientras que el código ajeno, esta alineado con el nombre de la función.

¿Donde terminá "función"?

Aquí función, hace referencia a `for`, `while`, `if` , entre otros. Más adelante trataremos mejor las funciones creadas por el usuario.

1.4 PEP-8

Algo importante a la hora de programar son tres cosas.

- Reproducibilidad. (Funcione en cualquier parte)
- Claridad (Se entienda que esta pasanddo)
- Facil de Leer.

En el caso de C, no hay reglas sobre como escribir un código más alla de la que se pueden encontrar en libros como "Clean Code" de Robert C.Martin.

Por el lado de Python, existe un proyecto llamado PEP-8 (https://peps.python.org/pep-0008/), en el que se estipulo un formato para un código de Python. Su relevancia es tal que, por ejemplo en un IDE de Python (PyCharm, más adelante hablaremos de él) puede configurarse para que, mediante advertencias, su código vaya tomando el formato PEP-8.

2 Estructura de un Código.

Un código en C en general consta de dos elementos. El préambulo, donde se encuentran los `#include<...>`, funciones creadas por el usuario, estructuras, entre otros, y el contenido (Lo que se encuentra en el `int main(){...}`).

A continuación recordaremos en que consistía el préambulo y el contenido en C. Y contaremos como funciona en Python.

2.1 El préambulo.

De forma básica, el preámbulo es lo que se encuentra fuera del `int main(){...}`. Donde encontramos

- `#include<...>`.
- `#define`.
- `struct{...}`
- Funciones creada por el usuario.

2.1.1 `#include<...>`

En C, el comando `#include` se usa para llamar librerías nativas y funciones creadas por el usuario. Por el lado de Python esto se realiza mediante dos "funciones" `import` y `from`, creadas para cumplir con las necesidades del lenguaje. La "función" `from` se verá en el apartado "Novedades interesantes de Python" más adelante. Por el momento nos centraremos en `import`.

La función `import` es la que se encarga de llamar las librerías nativas de Python y las funciones creadas por el desarrollador. Teniendo el siguiente formato.

```
import name
```

Una vez la librería es cargada, al usar una de sus funciones, indicarle a Python de donde la estamos sacando. Por ejemplo, si tengo una librería llamada 'algebra_lineal', y tengo una función para la factorización LU, llamada 'FLU'. En el código deberá aparecer algo como: algebra_lineal.FLU(…), lo cual no era necesario en C y aparte la escritura puede ser más lenta

cuando las librerías tienen nombres largos. Lo cual es cierto, sin embargo hay dos cosas a comentar:

Declarar la librería de origen permite tener funciones con el mismo nombre. Algo que en puede causar muchos problemas e incomodidades en C según el contexto, y La función `import` posee un parámetro opcional en el que nos permite asignar una abreviatura a la librería. Por ejemplo, con la librería 'algebra_lineal' y la función 'FLU'

```
import algebra_lineal as aL
y = aL.FLU(A)
```

Más adelante detallaremos elementos como 'y', entonces notamos que al agregar `as` y una abreviatura, no necesitamos poner todo el nombre de la librería para llamar una función. Esto funciona de igual manera para las librerías nativas.

Como última cosa a mencionar como diferencia entre `#include` e `import` es que en C requerimos un script tipo C (.c) y otro tipo header (.h) para poder hacer la llamada, en Python solo se requiere el script de la librería (.py). Más adelante mencionaremos las bondades que tiene la función `import` a la hora de programar.

2.2 `#define`

Al no haber una función principal (`int main(){...}`), no son necesarias las constantes globales tipo `#define` como es en C, por lo tanto "desaparecen" dentro del lenguaje Python.

2.3 `struct`

Más adelante veremos la alternativa de `struct`, al estar apenas notando las primeras diferencias, por el momento les basta saber que existe.

2.3.1 El contenido.

Sabemos que un script en C no es más que una función entera llamada 'main' (`int main(){...}`). En Python no se requiere una función para ejecutar el código escrito. Siendo necesario el clásico `return 0` de C.

References