**UNIVERSIDAD AUTÓNOMA DE BAJA CALIFORNIA**


Ingeniero en Software y Tecnologías Emergentes

Paradigmas de Programación

# Modelo de memoria de los lenguajes C y Python.

Integrantes:

Fernando Haro Calvo 372106

Andrea Rivas Gomez 372820

Teresa Rivas Gomez  372565

Danna Guadalupe Sandez Islas 373080

Angel Daniel Solano Meza 372453

GRUPO: 941

PROFESOR: Jose Carlos Gallegos Mariscal

11 de marzo de 2024

## Introducción

En el siguiente documento se encuentra todo lo investigado sobre el tema del modelo de memoria en lenguaje python y en C. Como se da a entender, realizamos una búsqueda en la web para entender con más profundidad este tema, que tocamos brevemente en clase.

“El modelo de memoria es un aspecto fundamental en la programación que influye en cómo se gestionan y almacenan los datos en la memoria de un sistema.”.

Como forma de resaltar las diferencias entre cada lenguaje, también nos dimos a la tarea de realizar una tabla comparativa, resaltando las principales.

## Desarrollo

Lenguaje C:

Cuando en C declaramos una variable, eso automáticamente reservaba memoria para el tipo de la variable, incluso sin contenido.

```c
int main() {
    int edad;
    edad = 30;
}
```

La primer línea ya reservaba los 4 bytes para guardar un int. La segunda línea alteraba el contenido con el número 30.

El lenguaje C implementa el tipo de gestión manual, es decir que cada programa debe explícitamente pedir memoria y liberarla. Para esto, necesitamos un API. En C tenemos un conjunto de funciones para:

- Pedir memoria: malloc()
- Liberar memoria: free()

Python:

En Python (y en otros lenguajes como Java, C# etc), no es la declaración de la variable la que reserva memoria. Una variable, desde el punto de vista del programador, no ocupa memoria. Lo que ocupa memoria es el objeto al que apunta. En nuestro caso el entero 30.

```
edad = None
edad = 30
```

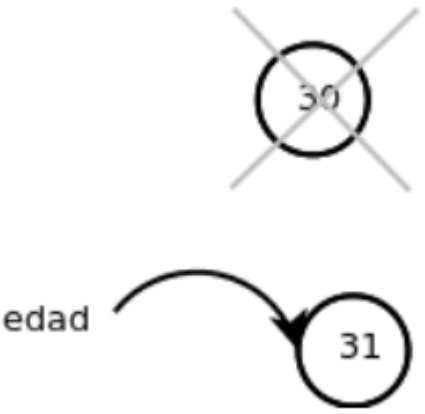
Acá vemos un código similar al de C. La primer línea es forzada, ya que no se puede declarar una variable sin que referencia a algo

A fin de simplificar, imaginemos que al ejecutarse la segunda línea, se crea el objeto "30" y se guarda en algún lugar en memoria. Transparente para nosotros. Luego, la variable edad es simplemente una referencia a ese objeto. Similar, aunque mucho más simple, que los punteros en C.

Si luego hacemos:

```
edad = 31
```

Pasará que la variable ahora hace referencia a un nuevo objeto entero alocado en memoria. El viejo objeto sigue existiendo en memoria, solo que nadie lo apunta ahora. Con lo cual el garbage collector va a ir tras él en algún momento.




Esto ocurre de la misma forma para todas las variables y parámetros.

Y no existen variables de tipo puntero como en C. Como así tampoco se puede saber la dirección de memoria de un dato.

En Python, tenemos un mecanismo de gestión implícita de memoria que se denomina garbage collector (o del castellano recolector de basura).

Consiste en que, cuando uno de los objetos deja de ser referenciado (es decir que no hay ninguna variable activa que lo esté referenciando), pasan a ser detectados por el garbage collector, quien va a liberar el sector de memoria que ocupaban.

El garbage collector es generalmente un subproceso del programa el cual periódicamente va a estar recorriendo la memoria, y analizando si cada sector tiene referencias o no. Cuando encuentra un sector no referenciado, lo libera, además se encarga de optimizar el uso de la memoria, especialmente su fragmentación.

**Tabla comparativa:**

| Aspecto | Lenguaje C | Lenguaje Python |
|---|---|---|
| Variables | Cuando se crea una variable, se reserva memoria para su tipo, incluso sin tener contenido. | La variable no toma espacio de memoria, en cambio apunta a un objeto. |
| Memoria dinámica | Debemos manejar la creación y destrucción manualmente. | La memoria se crea y se destruye sin la necesidad de programarlo explícitamente. |
| Punteros | Permite el uso directo de punteros, lo que brinda un control preciso sobre la memoria. | No permite el manejo directo de punteros, lo que reduce la complejidad pero limita el control sobre la memoria. |
| Eficiencia | Es altamente eficiente en términos de uso de memoria y velocidad de ejecución. | Tiende a ser menos eficiente que C debido a su naturaleza interpretada y dinámica. |
| Flexibilidad | Ofrece una gran flexibilidad pero requiere un mayor cuidado por parte del programador. | Proporciona mayor facilidad y flexibilidad en el desarrollo, a costa de cierta eficiencia. |

## Conclusión

En conclusión, como pudimos observar tanto en la información escrita como en la expuesta en la tabla comparativa, la mayor diferencia entre cada modelo de ambos lenguajes, es la manera en que se gestiona la memoria y la eliminación de la misma, donde en uno se debe realizar de manera manual y en el otro, el mismo lenguaje se encarga de realizar este procedimiento.

## Referencias


*Gestión automática (implícita) de memoria dinámica*. (s. f.). Google.com. Recuperado 9 de marzo de 2024, de https://sites.google.com/site/programacioniiuno/temario/unidad-1---manejo-de-memoria-dinmica/gestin-de-memoria/gestin-automtica-implcita-de-memoria-dinmica?authuser=0

*Programación en C con Memoria Dinámica*. (s. f.). Google.com. Recuperado 9 de marzo de 2024, de https://sites.google.com/site/programacioniiuno/temario/unidad-1---manejo-de-memoria-dinmica/programacin-en-c-con-memoria-dinmica?authuser=0

*Programación II - 03/05/2013 - clase 5 - introducción a python y manejo de memoria dinámica «automático»*. (s. f.). Google.com. Recuperado 9 de marzo de 2024, de https://sites.google.com/site/programacioniiuno/mi-cursada/bitacoras/03052013-clase5-introduccionapythonymanejodememoriadinamicaautomatico