we design and create professional quality software

Language - Agnostic

Language-agnostic programing

La programación o secuencias de comandos independientes del lenguaje es un paradigma de desarrollo de software en el que se elige un lenguaje en particular debido a su idoneidad para una tarea en particular, y no simplemente por el conjunto de habilidades disponibles dentro de un equipo de desarrollo.

Modern Portfolio Designed

Te quedo? Listo.
(Rodrigo Bueno)

python

PYTHON

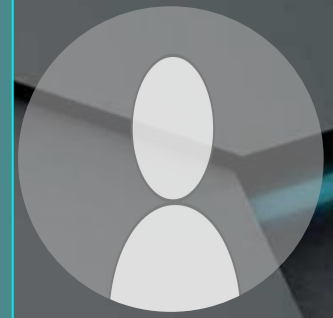
Tipos en PYTHON.

TIPOS DE DATOS EN PYTHON

INMUTABLES EN PYTHON

MUTABLES EN PYTHON

CONCLUSION



python

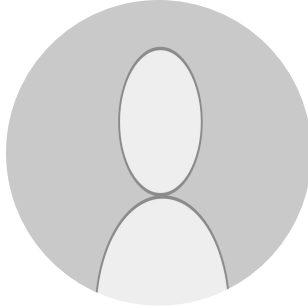
PYTHON

Introducción a la Mutabilidad e Inmutabilidad

INSTALACION Y ENTORO EN PYTHON.

python

Conceptualización



Mutabilidad e inmutabilidad en Python (y en otros lenguajes de programación), que las variables puedan cambiar (mutables) o no (inmutables) ¿Acaso no puede cambiar el contenido de cualquier variable, salvo que sean “finales” (esto es más de Java)? Sí, que sean “finales” quiere decir que la variable va a ser una constante y que no va a cambiar nunca a lo largo de la ejecución del código del programa. Realmente el valor de las variables por “**debajo**” cambian unas sí y otras no aunque parezca que realmente cambian.

Nota sobre los lenguajes a lo que se aplica: se aplica de manera muy parecida o exactamente igual en la mayoría (por no decir «todos») de los lenguajes; como en Java, JavaScript, PHP, etc.

Antes de empezar veamos un ejemplo muy sencillo en Python para ir descubriendo detalles:

```
mi_texto = "un texto"
```

Podremos cambiar el valor de la variable añadiendo algo más

```
mi_texto = mi_texto + " y algo más"
```

Por lo que al imprimir la variable por pantalla:

```
print(mi_texto)
```

Mostrará:

```
un texto y algo más
```

Viendo que el valor de tipo String de la variable mi_texto se puede cambiar, entonces te pregunto ¿Sting es mutable? Pues no, String es Inmutable porque en memoria RAM no ha ampliado el “un texto” guardado previamente, sino que lo ha copiado junto con el añadido de “ y algo más” para guardar al completo “un texto y algo más” en otro lado de la memoria RAM, y la variable apuntará al último valor guardado. Es decir, un String siempre se va a crear de nuevo (inmutable) aunque nosotros creamos que se modifica (falsa creencia de mutabilidad).

Nota En otros lenguajes, como C o C++, existe un tipo de variable especial llamado *puntero*, que se comporta como una referencia a una variable, como es el caso de las variables mutables del ejemplo anterior.
En Python no hay punteros como los de C o C++, pero todas las variables son referencias a una porción de memoria, de modo que cuando se asigna una variable a otra, lo que se está asignando es la porción de memoria a la que refieren. Si esa porción de memoria cambia, el cambio se puede ver en todas las variables que apuntan a esa porción.

Veamos esto con detalle.

python

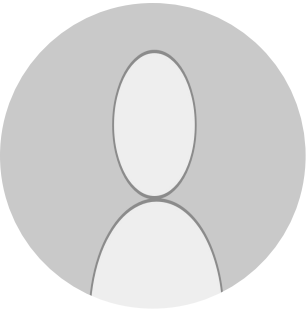
PYTHON

INMUTABLES

Tipos de Datos

python

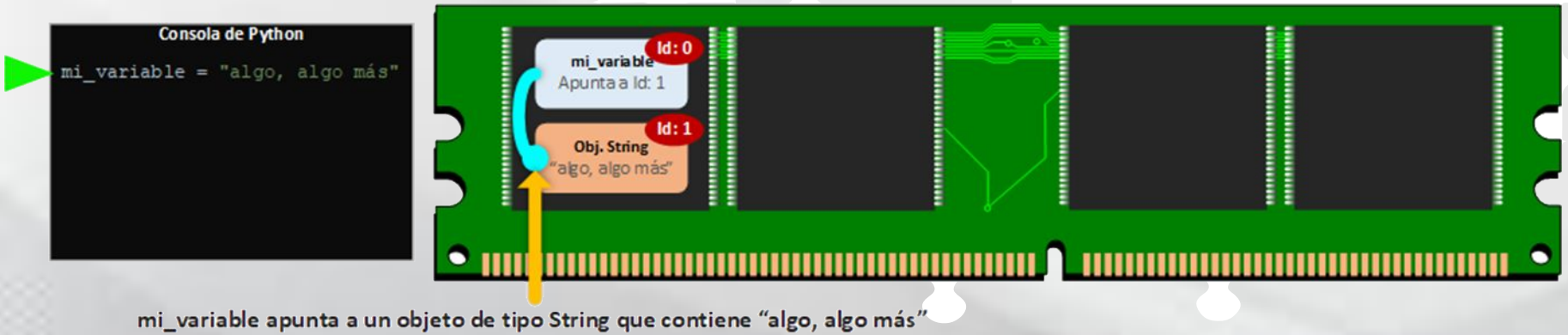
Inmutables



Los **inmutables** son los más sencillos de utilizar en programación (suelen ser los tipos simples de datos: String, Integer, Boolean, etc.) pues hacen exactamente lo que se espera que hagan en cada momento, y paradójicamente para funcionar así son los que más castigan a la memoria (no están optimizados para ocupar menos memoria al copiarse, y degradan más la vida útil de la memoria al escribirse más veces).

Un objeto inmutable una vez creado, como su propio nombre indica, su estado no puede cambiar una vez ha sido creado ¿Pero si he cambiado muchas veces valores tipo String o int después de crearlos? En apariencia, pero no cambian nada se explican a continuación (se sugiere que a medida que leas cada uno de los próximos párrafos te fijes en la imagen que lo acompaña debajo; en las siguientes imágenes se lee primero desde la izquierda la “Consola de Python” con una flecha verde que indica la línea de código añadida, y luego la parte de la derecha con la memoria y sus ocupantes).

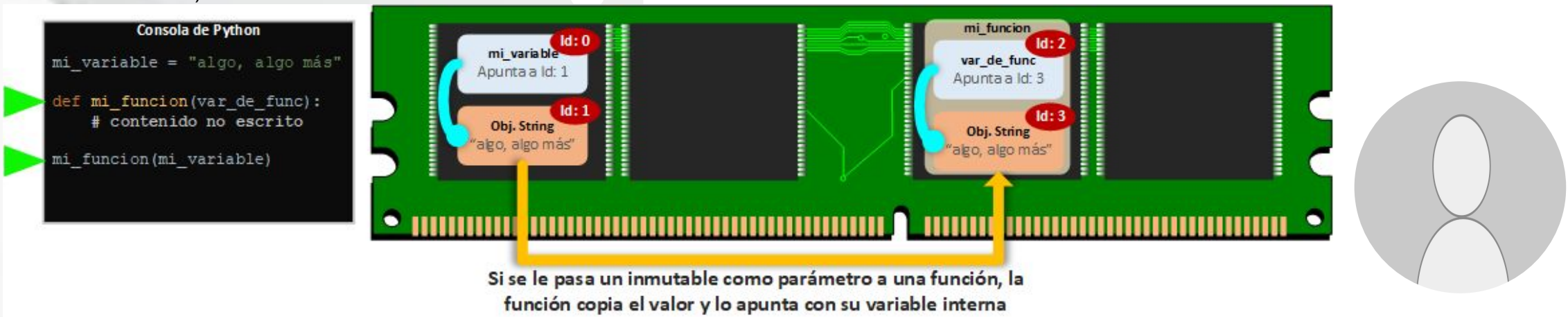
Cuando declaramos una variable (en la siguiente imagen “mi_variable”) en Python (o cualquier otro lenguaje de programación) se crea en memoria una variable en una dirección de memoria (en la imagen se guardará en la dirección de memoria “id: 0”). Si luego a continuación le asignamos (la variable es un cursor o puntero que “apuntará a la dirección de memoria de su valor”) un valor mutable, que para este ejemplo será un texto (String, en la imagen “algo, algo más”) y se creará en otra posición de memoria (en la imagen en la dirección de memoria “id: 1”); entonces la variable anterior apuntará al objeto mutable que contiene su valor.



python

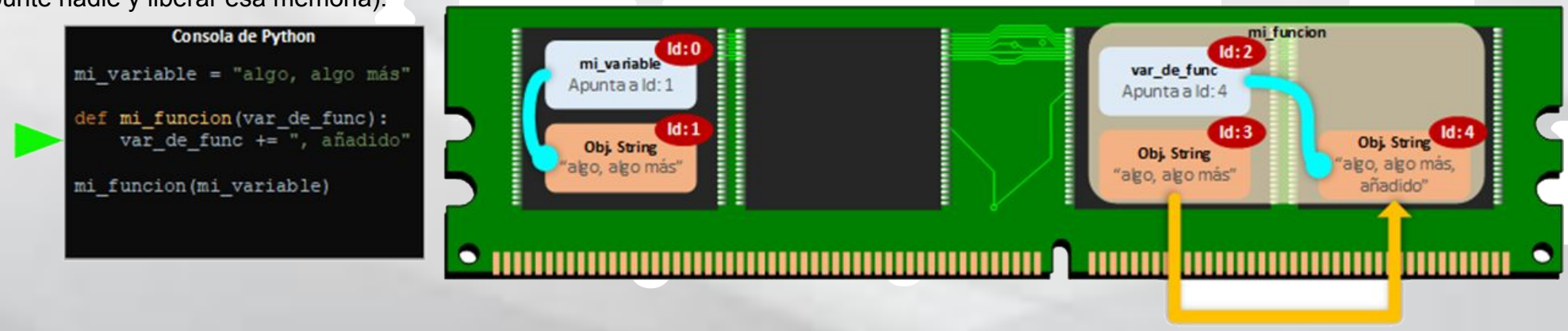
Inmutables

Aunque podría hacer un ejemplo simple, aquí te voy a enseñar dos casos de la inmutabilidad (aunque en el fondo es lo mismo, pero aclara muchos conceptos este primer caso). El primer caso de inmutabilidad, y el que creo que es más importante porque sule ser el que más confunde, es cuando se pasa como parámetro el valor (de una variable) inmutable a una función. Cuando se pasa el valor inmutable como parámetro a una función (en la imagen a la variable “var_de_func” que está en la dirección de memoria “Id:2”), inmediatamente dicho valor se copia a otra dirección de la memoria (en la imagen “Id: 3”); por lo que podrás deducir que si se modifica dentro de la función habrá que extraerlo (normalemnte con el “return” dentro de la función) para poder utilizarlo fuera de la función. Por tanto, lo que pase dentro de la función se quedará dentro de la función (como dice el dicho “lo que pase en las Vegas se queda en las Vegas” 😛) y no afectará a variables fuera de la función (salvo que se devuelva con “return”).



Si se le pasa un inmutable como parámetro a una función, la función copia el valor y lo apunta con su variable interna

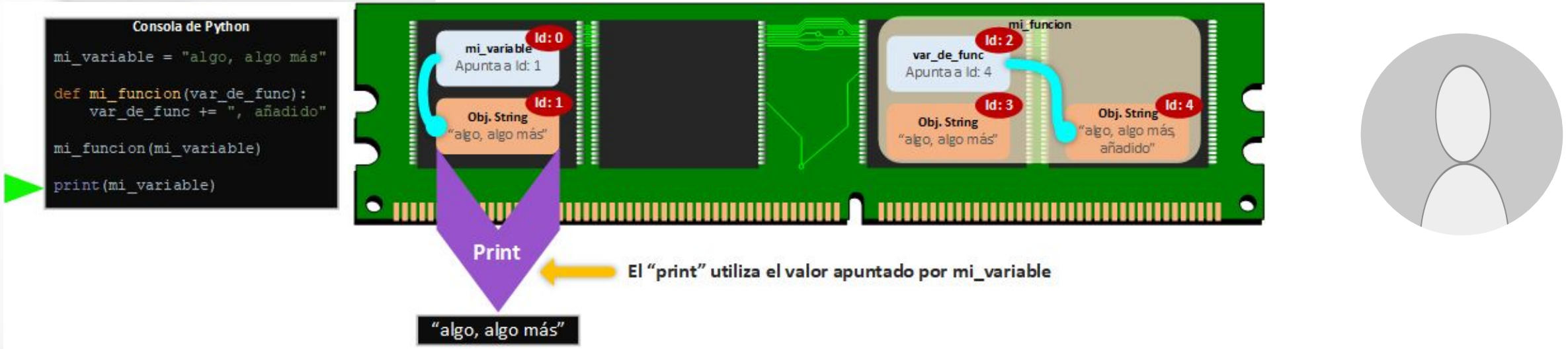
El segundo caso de inmutabilidad no es tan apreciable y pasa más desapercibido. Esto es al modificarse un objeto de un valor inmutable, lo que hace realmente es copiarse y guardarse la modificación propiamente en otra dirección de memoria (en la imagen “Id: 4”), haciendo que la variable apunte a este nuevo objeto y borrando el objeto que no es apuntado (normalmente el borrado lo van a hacer los recolectores de basura o en inglés “garbage collector”, que se encargan de buscar objetos a los que no apunte nadie y liberar esa memoria).



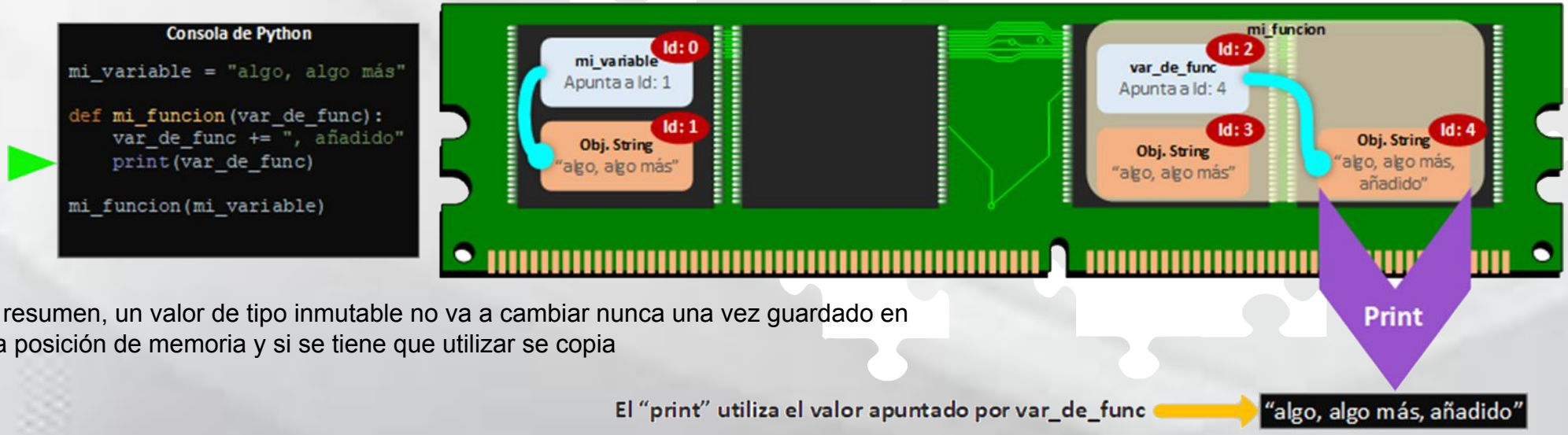
Se ejecuta la función y modifica el valor de las variables a las que apunta

Inmutables

De este modo, si por ejemplo al final mostrar por pantalla el valor guardado de “mi_variable” nos devolverá lo que está guardado en la posición de memoria a la que apunta (que es lo que había al principio, pues no se ha tocado):

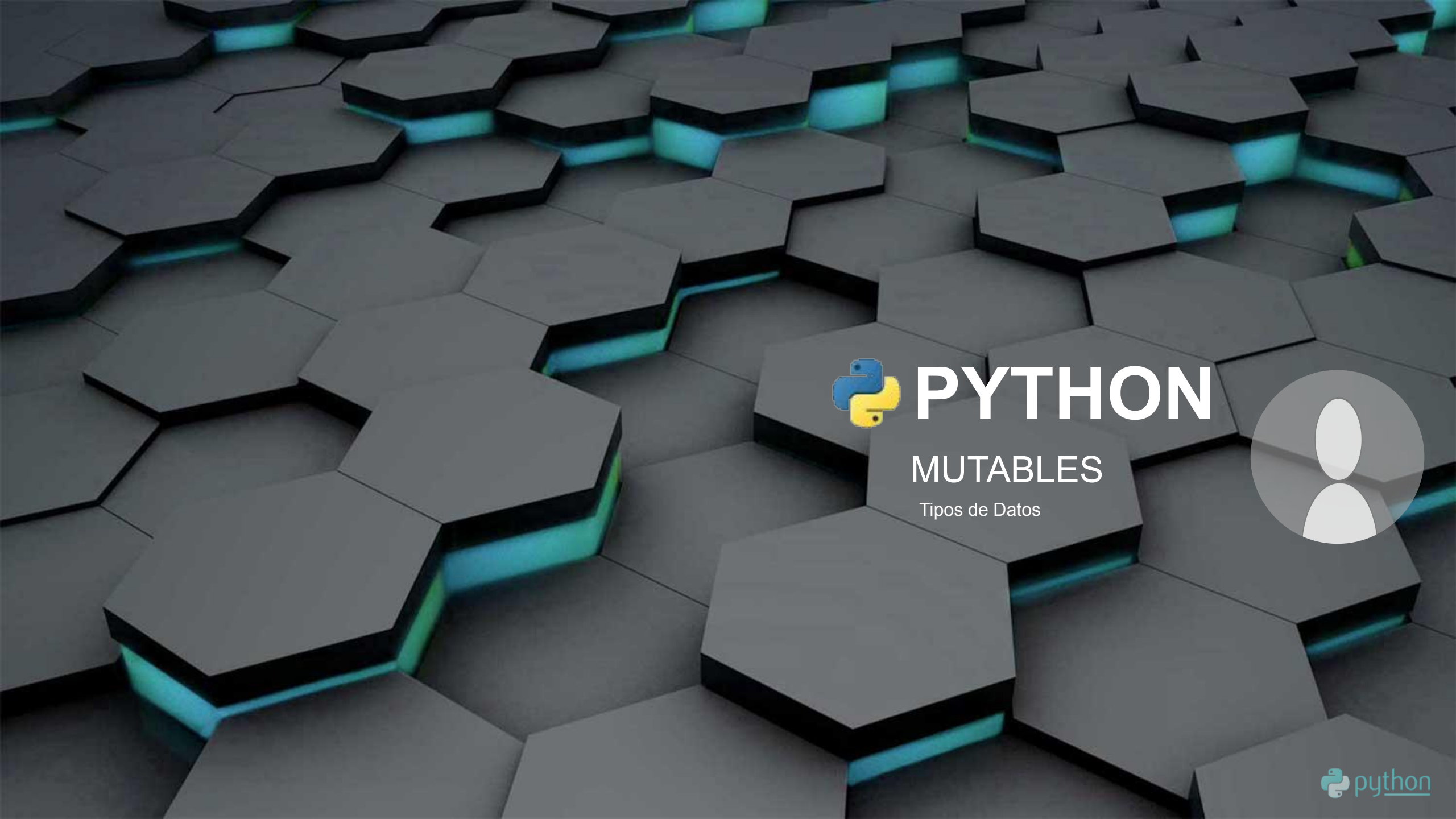


O si mostrásemos por pantalla lo que hay dentro de la función nos imprimiría a donde apunta la variable “var_de_func”:



En resumen, un valor de tipo inmutable no va a cambiar nunca una vez guardado en una posición de memoria y si se tiene que utilizar se copia

python



PYTHON

MUTABLES

Tipos de Datos

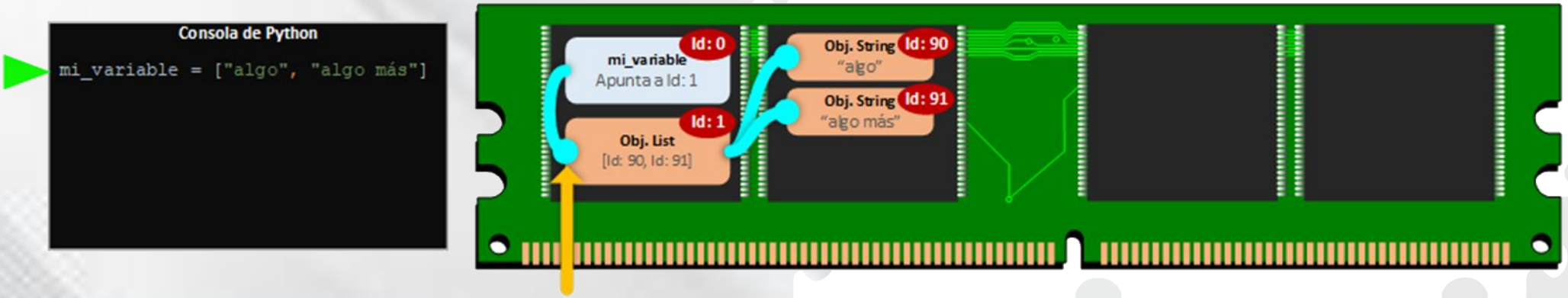
python

Mutables

Los mutables son objetos que, una vez creados, su estado puede cambiar en el futuro.

Los mutables son los más “complejos” de utilizar en programación (suelen ser las estructuras de datos como: dict, list, etc) y no solo porque son más complejos porque son estructuras que tienen cosas, sino que suelen liar con el tema de los punteros y, paradójicamente, son los que menos perjudican a la memoria (se escriben una sola vez y se reutilizan siempre). Hay que decir que los mutables están diseñados así a proposito, porque copiar una estructura de datos entera (aunque se puede) tardaría mucho e implicaría utilizar mucha memoria para seguramente no aprovechar la copia (no es lo mismo copiar un objeto String que copiar una lista con millones de objetos String, para luego no haber necesitado la copia; llega a ser un derroche de procesador y de memoria).

Como hicimos antes con Imutables, ahora veremos un ejemplo de Mutable, concretamente un listado de valores String. Se guardará en un espacio de memoria (en la imagen “Id: 0”) la variable “mi_variable” que apunta a otra posición de memoria (en la imagen “Id: 1”) donde estará nuestro objeto Inmutable de tipo listado. Y para empezar inicializaremos el listado con dos Strings (en la imagen apunta a “Id: 90” y “Id: 91”).

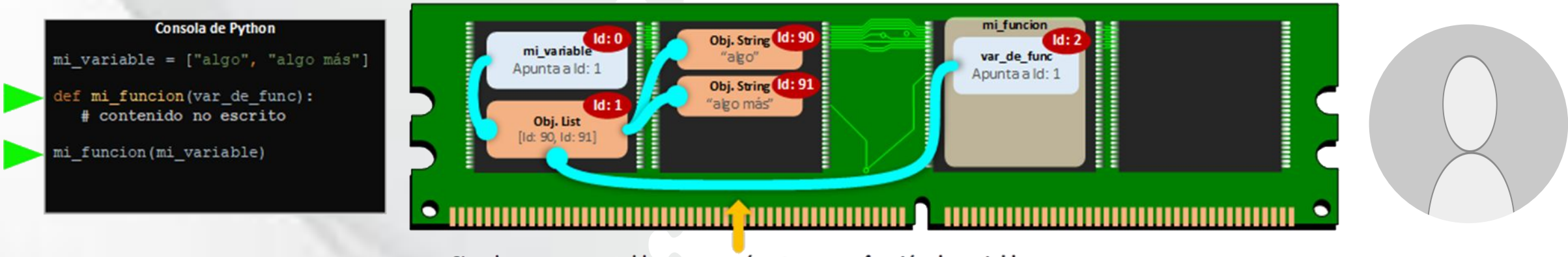


mi_variable apunta a un objeto de tipo List que contiene un listado con dos elementos: [“algo”, “algo más”]

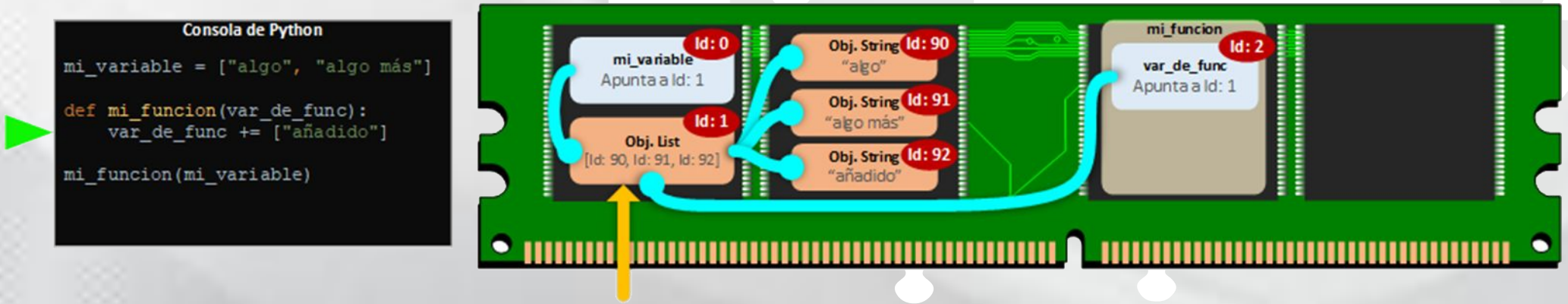
Nota: el listado guarda en un Array con apuntadores a los objetos Inmutables String en partes diferentes de la memoria

Mutables

Cuando se pasa el valor mutable como parámetro a una función (en la imagen a la variable “var_de_func” que está en la dirección de memoria “Id:2”), la variable apuntará al objeto mutable (en la imagen “Id: 1”), por tanto NO se copia. Aquí ya se pueden deducir algunos problemas (cuando se modifique el objeto dentro de la función se modificará fuera) y ventajas (no ocupa más memoria y la “copia” de un mutable gigante es inmediata).



Así, cuando se modifique el objeto mutable desde dentro de la función, se modificará para todos (dentro y fuera de la función). En este ejemplo añadiremos un nuevo valor String al listado.



Mutables

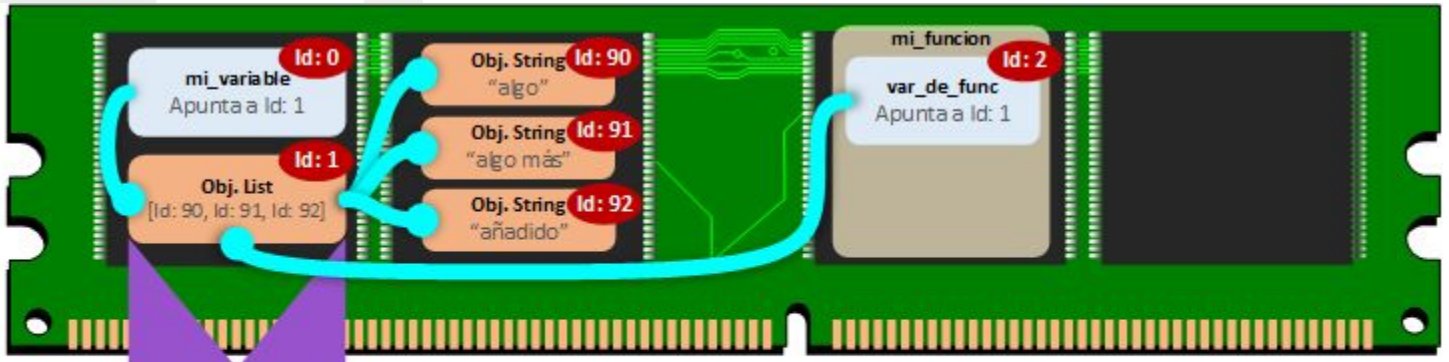
Si imprimimos mi_variable desde fuera de la función comprobaremos que efectivamente “alguien” ha modificado nuestro objeto mutable.

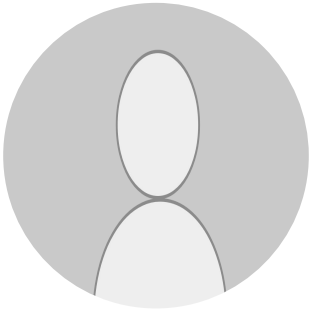
```
Consola de Python
mi_variable = ["algo", "algo más"]

def mi_funcion(var_de_func):
    var_de_func += ["añadido"]

mi_funcion(mi_variable)

print(mi_variable)
```





Print

El “print” utiliza el valor apuntado por mi_variable (que ha sido modificado por la función)

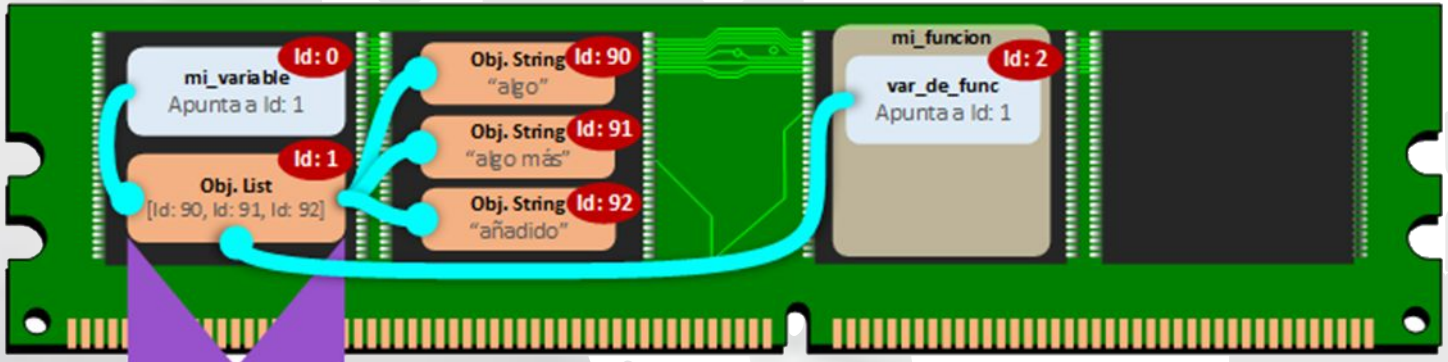
[“algo”, “algo más”, “añadido”]

Por otro lado, si imprimimos var_de_func dentro de la función, como apunta al mismo objeto, imprimirá lo mismo.

```
Consola de Python
mi_variable = ["algo", "algo más"]

def mi_funcion(var_de_func):
    var_de_func += ["añadido"]
    print(var_de_func)

mi_funcion(mi_variable)
```



Print

El “print” utiliza el valor apuntado por var_de_func (que es el mismo al que apunta mi_variable)

[“algo”, “algo más”, “añadido”]

En resumen, **un valor de tipo mutable se puede cambiar en un futuro una vez guardado en una posición de memoria y si se tiene que utilizar se utiliza el mismo objeto siempre apuntado desde diferentes variables**.

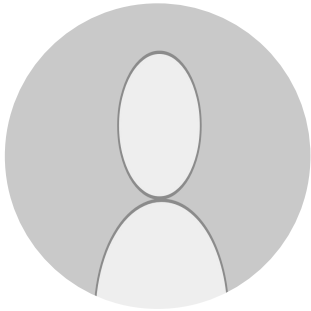
PYTHON

INMUTABLES & MUTABLES

Tipos de Datos

python

Tipos en Python clasificados en Inmutables y Mutables



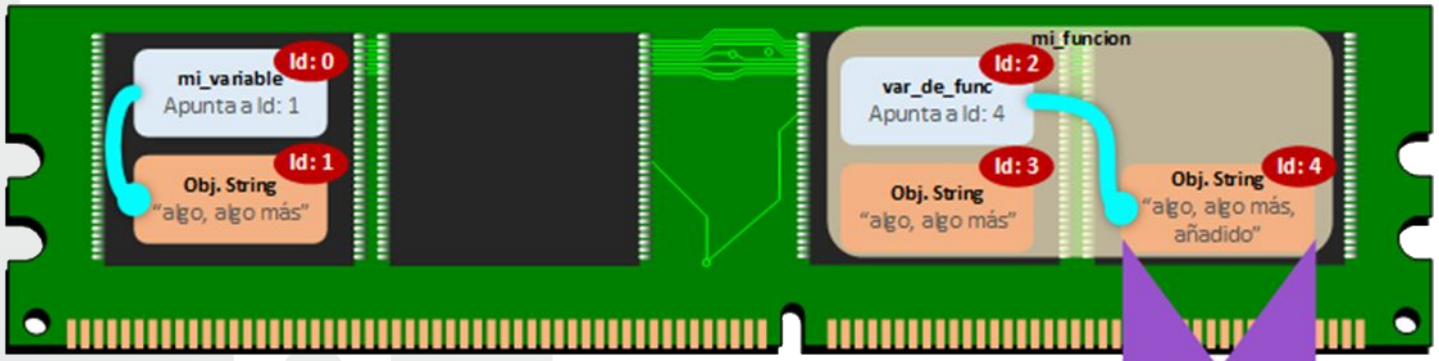
Inmutables

```
Consola de Python

mi_variable = "algo, algo más"

def mi_funcion(var_de_func):
    var_de_func += ", añadido"
    print(var_de_func)

mi_funcion(mi_variable)
```



El "print" utiliza el valor apuntado por var_de_func → "algo, algo más, añadido"

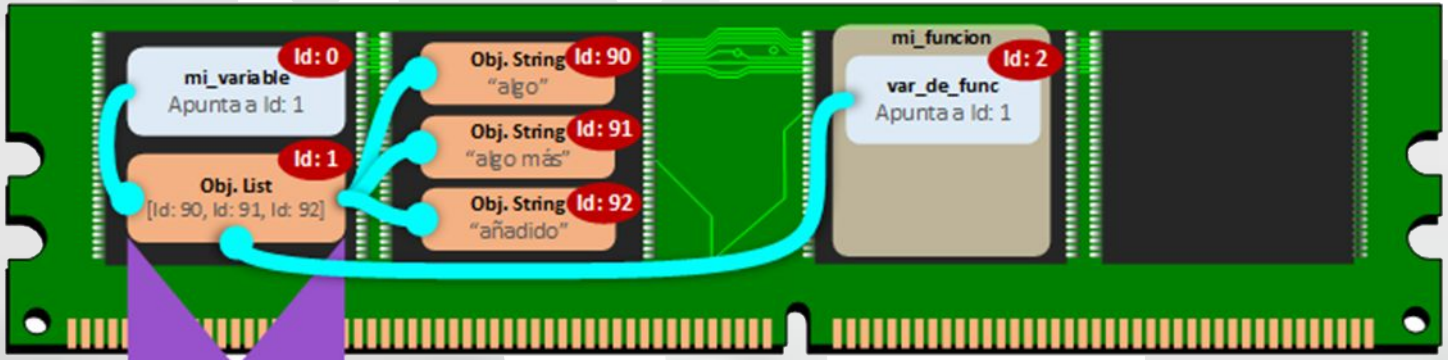
Mutables

```
Consola de Python

mi_variable = ["algo", "algo más"]

def mi_funcion(var_de_func):
    var_de_func += ["añadido"]
    print(var_de_func)

mi_funcion(mi_variable)
```

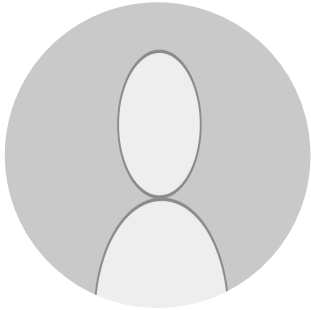


El "print" utiliza el valor apuntado por var_de_func
(que es el mismo al que apunta mi_variable)

["algo", "algo más", "añadido"]

python

Tipos en Python clasificados en Inmutables y Mutables



Inmutables:

- **Strings** (Texto)
- **Int** (Número entero)
- **Float** (Número de coma flotante)
- **Decimal** (Número decimal)
- **Complex** (Complejo)
- **Bool** (Booleano)
- **Tuple** (Tupla): Actúa como una lista inmutable siempre que sus elementos sean Resumibles (“Hashables”).
- **Frozenset** (Conjunto congelado): Actúa como un conjunto inmutable siempre que sus elementos sean Resumibles (“Hashables”)
- **Bytes**
- **Range** (Rango)
- **None** (Nulo)

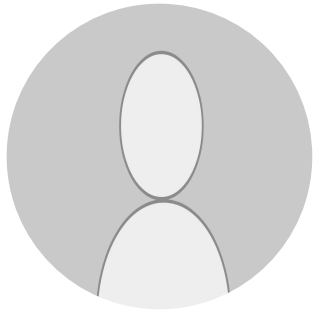
Mutables:

- (Listado): Admite cualquier tipo de valores
- (Diccionario : Admite solo claves Resumibles (“Hashables”) y valores de cualquier tipo
- **Set** (Conjunto): Admite solo valores Resumibles (“Hashables”)
- **Bytearray** (array de bits): Entre otros usos, se puede utilizar como un String mutable.
- **MemoryView** (Vista de la memoria): Referencia a objetos
- **Clase definida** por el programador. Más información en ela

rtículo sobre Hashable.
Más información de cada tipo en la documentación oficial en: https://docs.python.org/3/library/stdtypes.html

python

Objetos mutables e inmutables en Python



Que un tipo sea inmutable significa que no puede ser cambiado luego de haber sido creado.

Veamos un ejemplo para que nos quede mas claro.

```
>>> apellido = 'Gonzales'
>>> apellido
'Gonzales'
>>> apellido[0] = 'g'
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'str' object does not support item assignment
>>> apellido
'Gonzales’
```

En este ejemplo hemos asignado la cadena 'Gonzales' a la variable apellido. Luego estamos intentando cambiar la primera letra de la cadena, y por supuesto, nos tira un error.

El error se traduciría como: "El objeto 'str' no soporte la asignación de elementos"

Y esto se da porque estamos intentando cambiar el valor de un objeto inmutable, ya que las cadenas en Python son inmutables.

Lo que si podríamos hacer es asignarle un nuevo valor a la variable que no vendría a ser lo mismo que cambiar la cadena.

```
>>> apellido = 'g' + apellido[1:]
>>> apellido
'gonzales’
```

Según este ejemplo pareciera que estamos cambiando la cadena, pero en realidad lo que estamos haciendo es asignándole un nuevo valor, ya que si te das cuenta, le estamos pasando el signo = y el nuevo valor que queremos que tenga esa variable.

Entre otras cosas, la inmutabilidad (inmutability) se puede utilizar para garantizar que un objeto permanezca constante en todo el programa; los valores de los objetos mutables se pueden cambiar en cualquier momento y lugar (y si lo espera o no).

Language - Agnostic

Language-agnostic programing

La programación o secuencias de comandos independientes del lenguaje es un paradigma de desarrollo de software en el que se elige un lenguaje en particular debido a su idoneidad para una tarea en particular, y no simplemente por el conjunto de habilidades disponibles dentro de un equipo de desarrollo.

Modern Portfolio Designed

THANK YOU

Let`s Python Code…

python