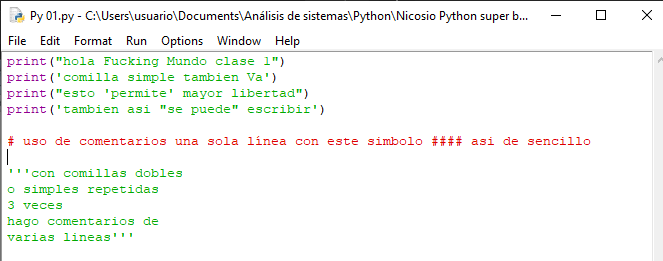
Python básico para gente de C#, C++ y Java

<https://www.youtube.com/playlist?list=PLM-p96nOrGcZmWE-320vKw-e6TJsle7rC>

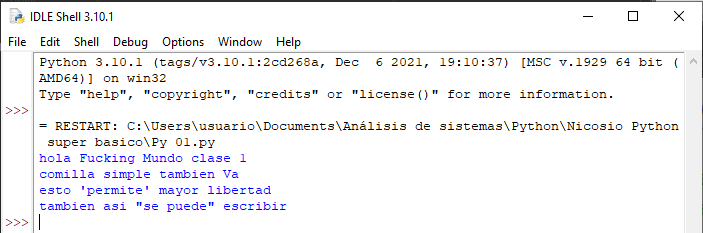
Este curso lo desarrolla en Python 2.X hay diferencias de sintaxis con las versiones 3.X, pero es una buena ayuda para los que tenemos pocos o nulos conocimientos.

# print y comentarios - 2 - Python básico para gente de C#, C++ y Java

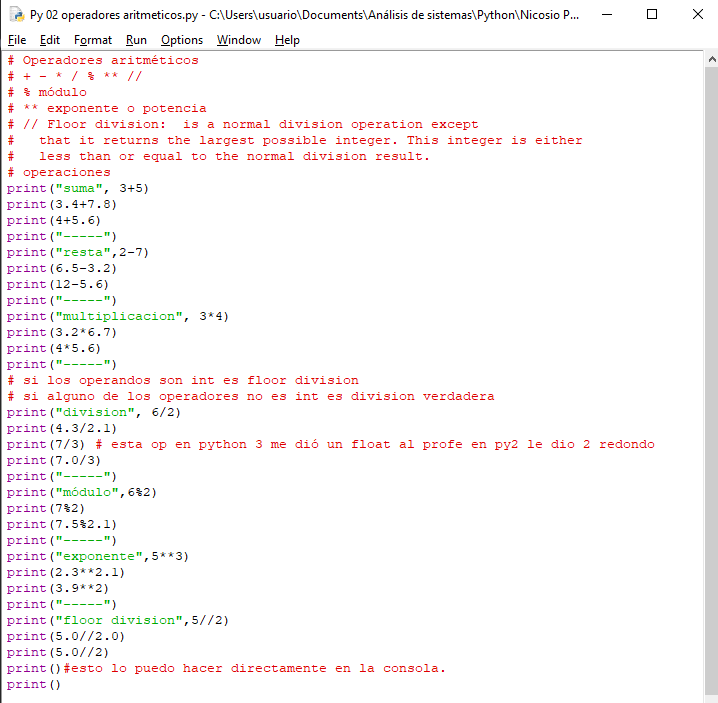
Sentencias con cadenas de texto (String):



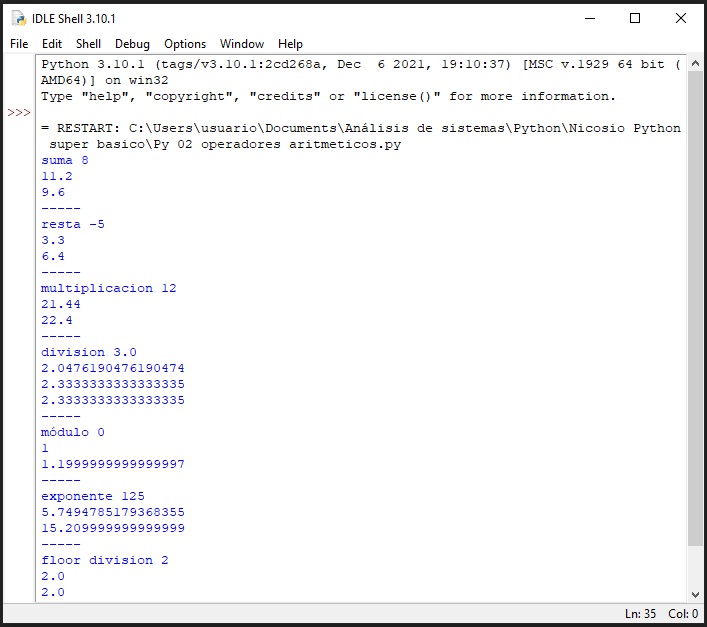
Resultado/salida:



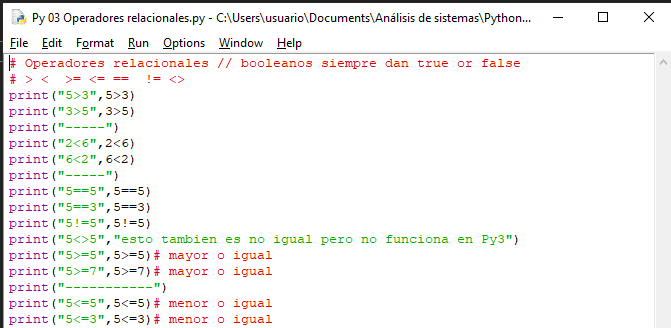
# Operadores aritméticos - 3 - Python básico



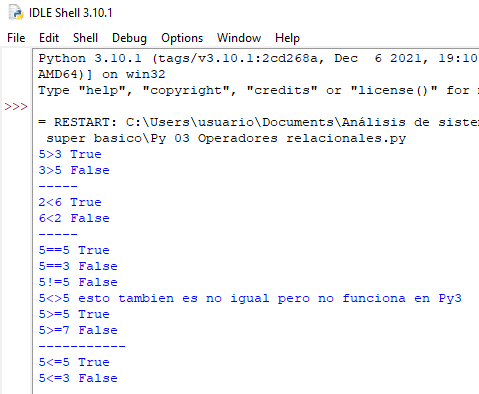
Salida:



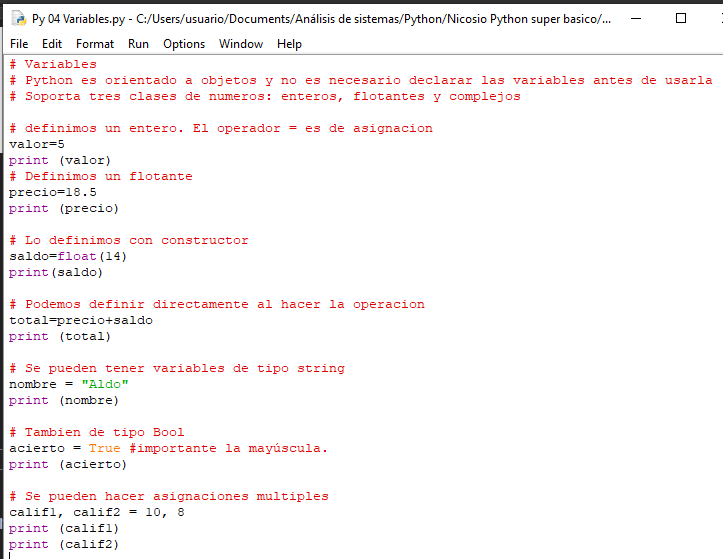
# Operadores relacionales - 4 - Python básico



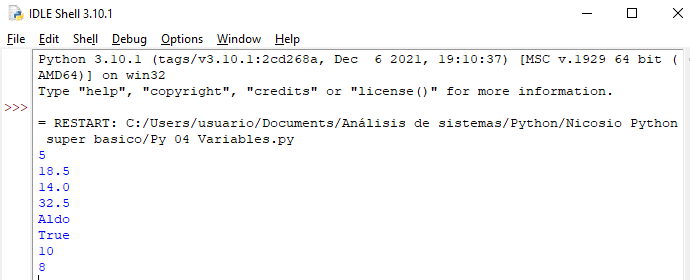
Salida:



# Variables - 5 - Python básico

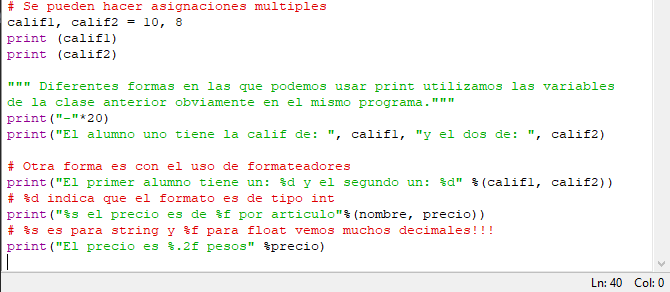


Salida:

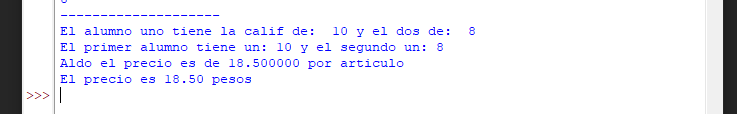


# Print con variables - 6 - Python básico

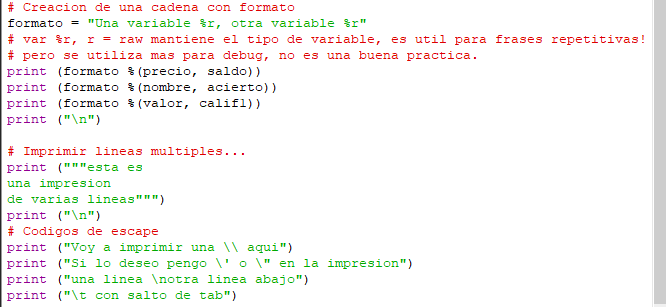
**“-”\*20 // %d // %s // %f // %.2f**



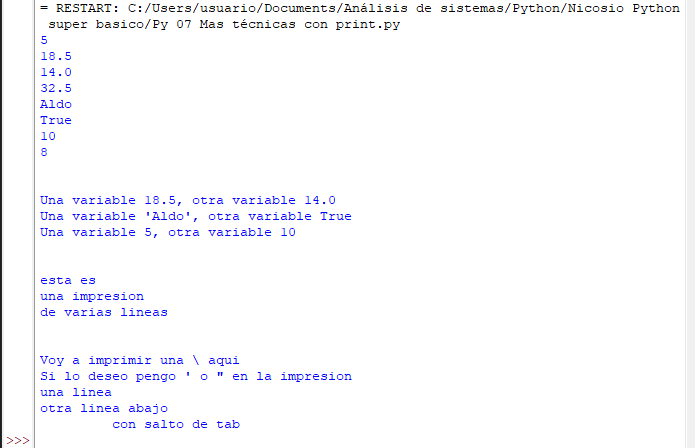
Salida:



# Más técnicas con print -7- Python básico



Salida:



# Input y raw\_input - 8 - Python básico.

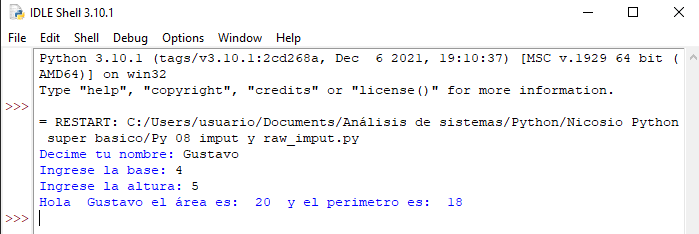
Al realizar los ejercicios note que raw\_input no existe como tal en las versiones 3.X así que me quede solo con la primera parte.

También utilicé la función type para saber qué tipo de dato almacena la variable.

**Muy importante Convertir o Parsear las variables ingresadas.**



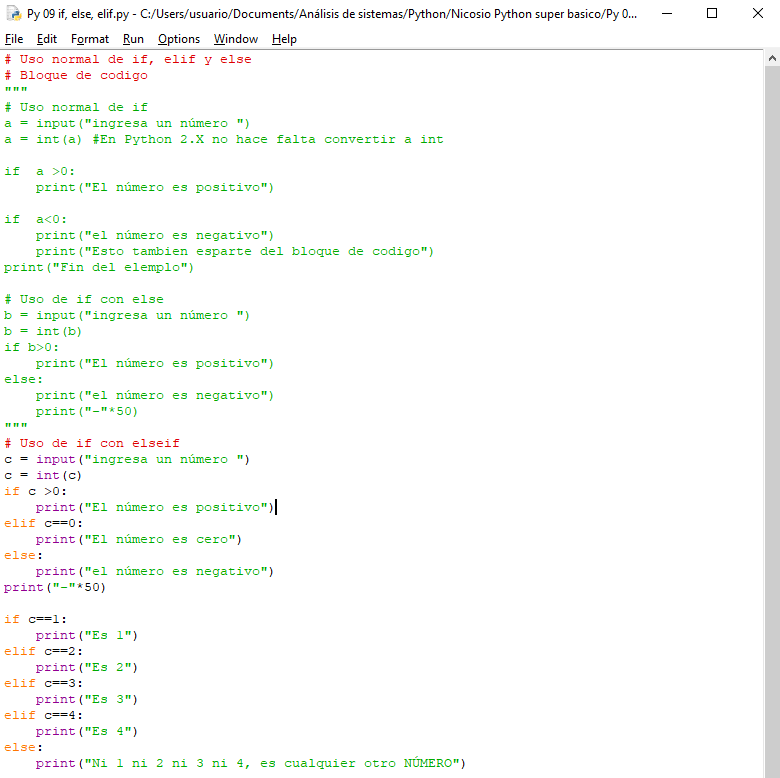
Salida:



# if, else, elif -9 - Python básico

**¡¡En este caso es importante recordar que Python es secuencial y va a ejecutar las sentencias en orden!!**

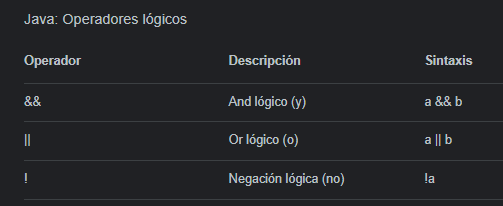
**En el ejemplo va a ir evaluando en orden una a una las condiciones, por eso va a arrojar resultados de los dos bloques, el superior que evalúa si es positivo, negativo o igual a cero y seguidamente si es igual a 1, 2, 3, 4, o ninguno de esos.** [**https://www.youtube.com/watch?v=CrlIv1U1rTI&list=PLM-p96nOrGcZmWE-320vKw-e6TJsle7rC&index=9**](https://www.youtube.com/watch?v=CrlIv1U1rTI&list=PLM-p96nOrGcZmWE-320vKw-e6TJsle7rC&index=9)

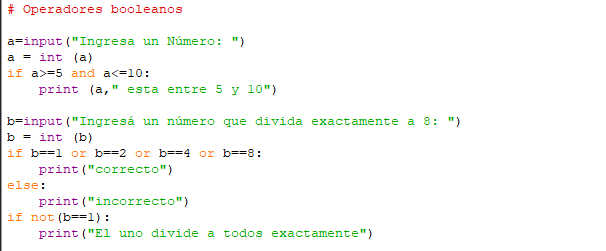
****

**No voy a subir la salida porque son muchas opciones distintas, pero junto con este resumen están los Scripts de cada clase.**

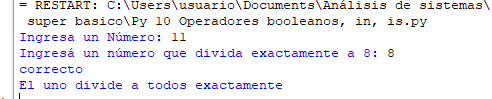
# Operadores booleanos, in, is - 10 - Python básico

**Un detalle importante en la sintaxis es que and, or y not , se escriben no se usan los símbolos & , || o ! como son utilizados en otros lenguajes como Java, por ejemplo. ¡¡Ver los comentarios!!**

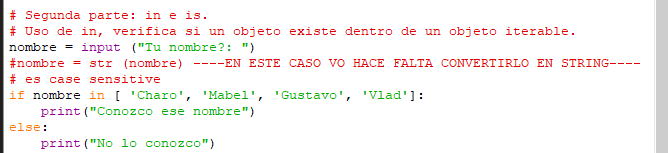




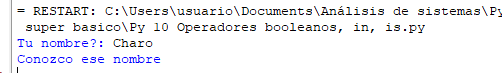
Salida:



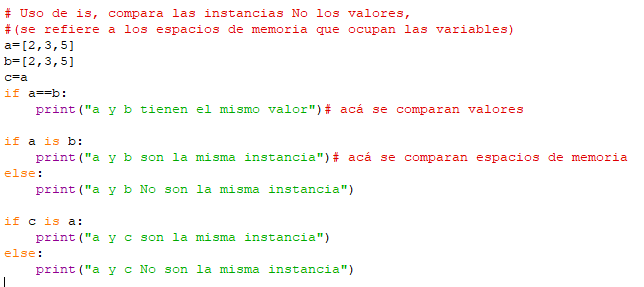
2.



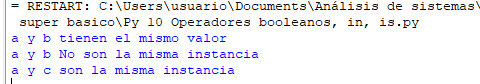
Salida:



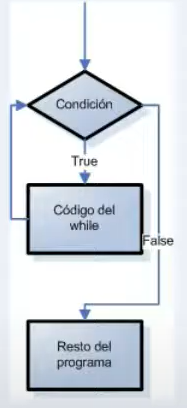
3.



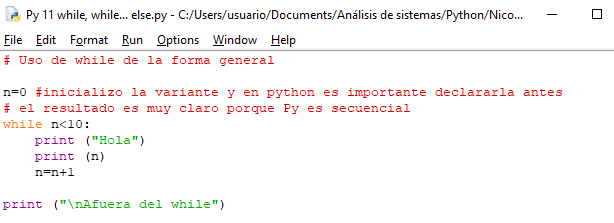
Salida:



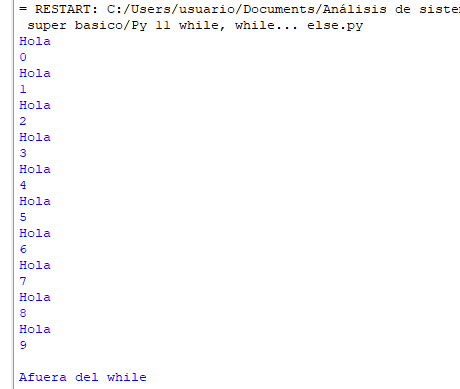
# while, while... else - 11 - Python básico



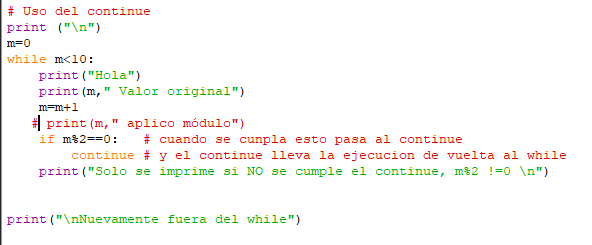
Ciclo while.



Salida:

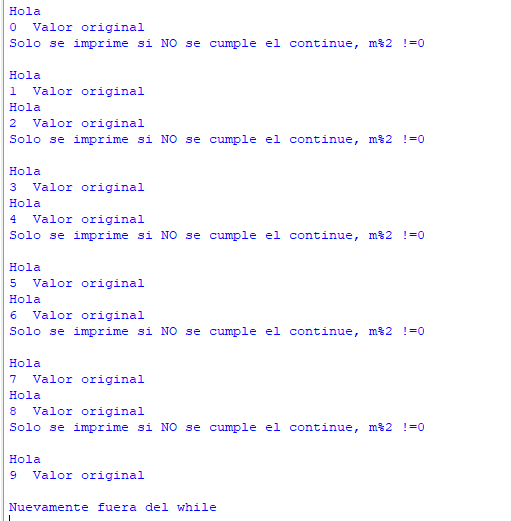


Al ejercicio con “continue” lo hice a prueba de tontos porque no lo entendía bien.

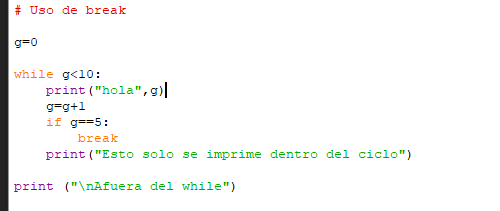


Se ve algo confuso, la línea comentada “aplico modulo” es para mostrar porque se aplica o no el continue imprimiendo en pantalla según si es par o impar.

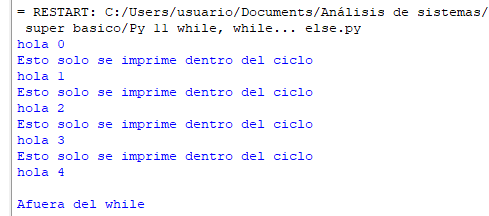
Salida:

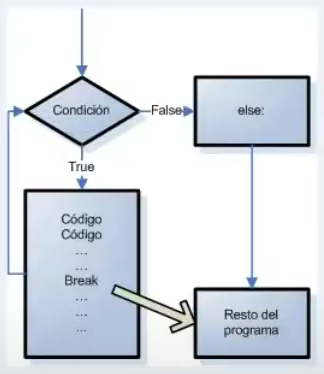


Uso de break

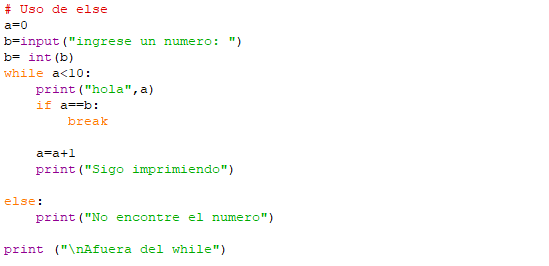


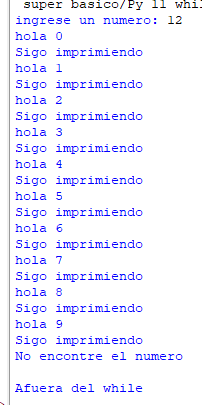
Salida: break corta el ciclo donde sea que se encuentre y sigue con la siguiente instrucción.

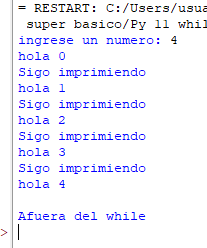


Else dentro de un ciclo while.

Si se agotan las condiciones true del while pasa al *else* y luego continua al resto del programa, si se utiliza un *break* el *else* no se ejecuta.

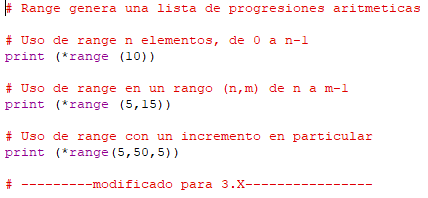


Salida: en este ejemplo si el número ingresado es menor a 10 lo encuentra y corta el ciclo, si es mayor llega hasta el else y luego sale del ciclo.



# range y for- 12 - Python básico

el video nº 12 es exclusivo de Python 2.X busque la manera de representarlo en 3.X y quedo similar.

# Ejemplos de for con range - 13 - Python básico

Es muy distinta la sintaxis en Py 2.X y 3.X voy a ir directamente a 3.X

<https://www.youtube.com/watch?v=5KO7exSdlp0> (en Pycharm)

<https://www.discoduroderoer.es/range-en-python-3/#comment-10733> (en CMD)

Voy a utilizar 2 ejemplos uno en CMD y otro en el IDLE Shell 3.10.1 Porque tienen sintaxis distintas.

Range en Python 3 es como un tipo de dato que nos genera un rango de números de un inicio a un final en formato de lista.

Es muy útil cuando queremos recorrer por ejemplo un simple rango de números y no tener que crearlo con un array como en otros lenguajes.

Tiene 3 parámetros:

* **El primero**, indica el inicio del rango incluyendo el mismo, si es un solo parámetro, se considera el final y empieza desde 0 y va 1 a 1.
* **El segundo**, indica el final del rango no incluyendo el mismo, por defecto va de uno en uno.
* **El tercero**, indica el paso del rango, es decir, si vamos de 2 en 2, 3 en 3, etc. No puede ser cero.

Usamos list() para que podamos ver los elementos.

Veamos algunos ejemplos:

* Generar un rango entre 0 y 10.

rango_1

* Generar un número entre 5 y 10

rango_2

* Generar un rango entre 0 y 10 pero dando pasos de dos en dos.

rango 3

* Generar un rango negativo.

rango 4

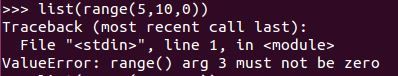
Se genera una lista vacía ya que empieza en 0 y va de 1 en 1, por eso está vacía.

* Generar un rango entre -5 y 5

rango 5

Si le damos un final si funciona si usamos un negativo.

* Generar un rango entre 5 y 10 con paso 0



El avance no puede ser cero.

* Generar un rango de 10 a 0.

rango 7

Podemos ir de delante a atrás, pero es obligatorio el -1

* Generar un rango de 0 a 10 y de 15 a 20, incluidos el 10 y 20

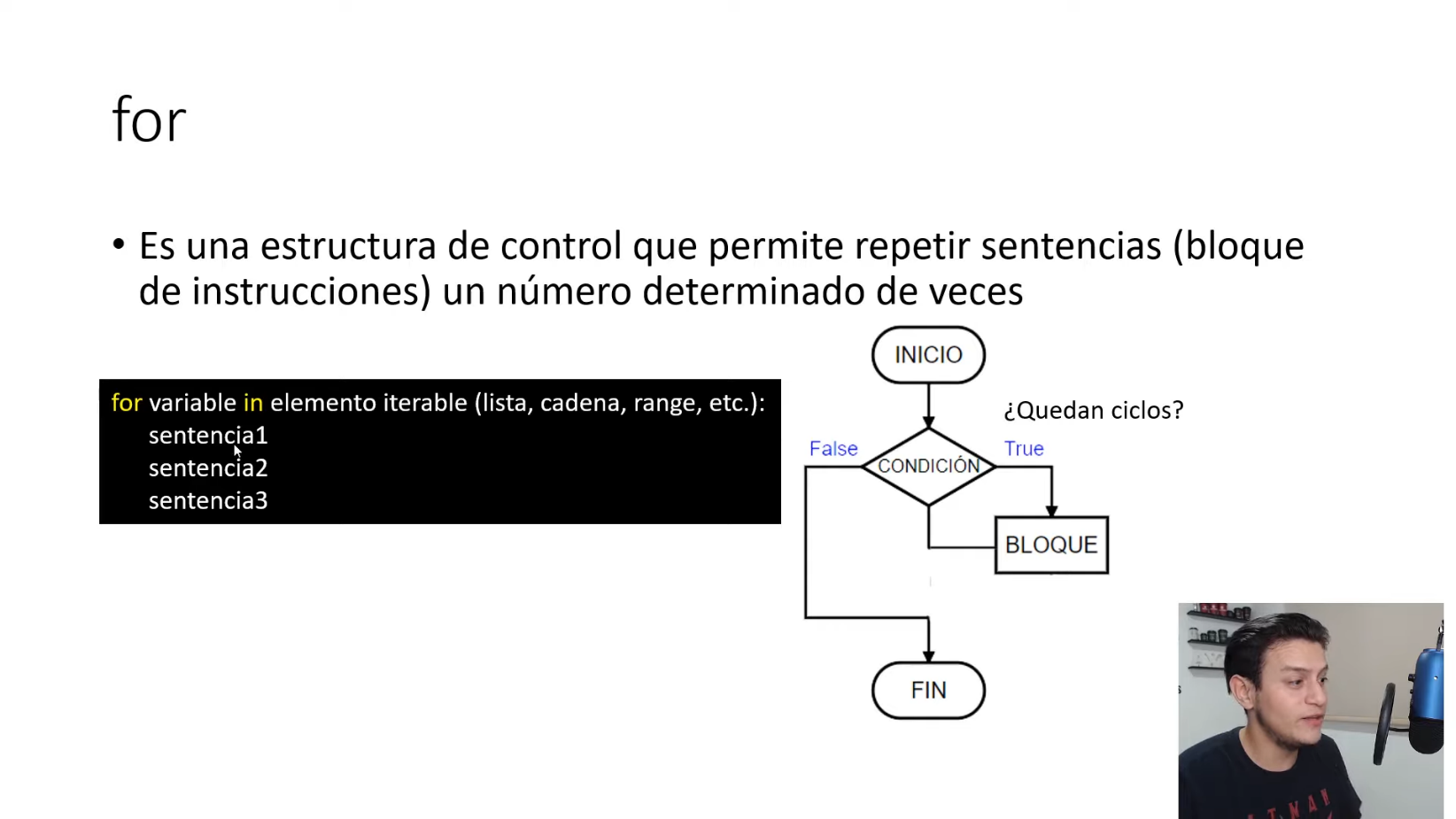
rango 8

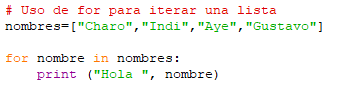
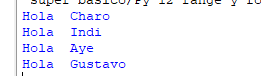
Se puede sumar rangos para hacerlo uno. **Importante**: aquí **list()** es muy importante ya que sino nos dará un error.

* Generar un rango desde 0 hasta la longitud de la cadena “Hola mundo”

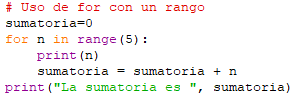
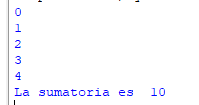
rango 9

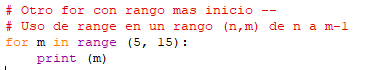
Aquí debemos usar la función len() que nos permite saber la longitud de una cadena, lista o tupla.

**For como iterador**

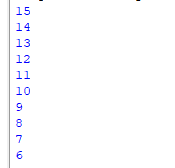
** Salida:**

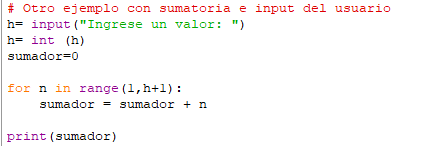
**En el caso de números los ejemplos más básicos son los mismos que en el ejemplo de CMD mencionado algunas líneas arriba.**

** Salida: **

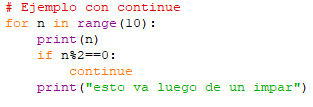
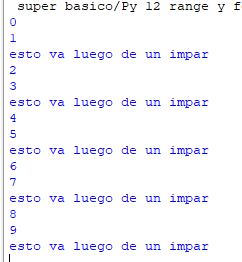
** Salida: **

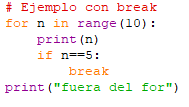
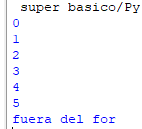
**El siguiente ejemplo puede ser positivo o negativo.**

** Salida: **

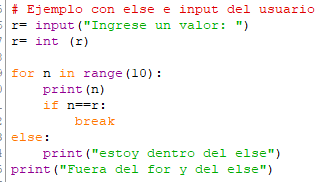
** Salida: **

**De la misma manera que en la clase anterior se pueden utilizar el continue y break**

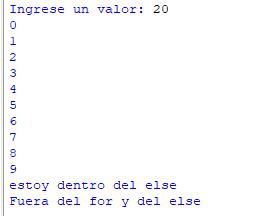
** **

** **

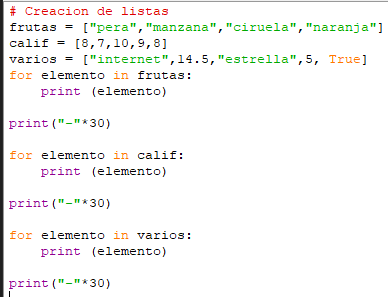
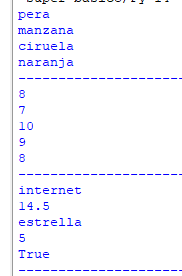
**Por último, un ejemplo con else:**

** con salida en break **

**Ahora como no pasa por break por ser un valor mayor al rango, utiliza el else:**

****

# Listas I - 14 - Python básico

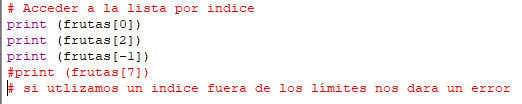
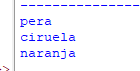
Las listas son estructuras de datos tiene capacidades polimórficas, la forma más fácil de acceder a las listas es a través del for para así poder iterar los distintos elementos.

También las puedo imprimir sin iterar:

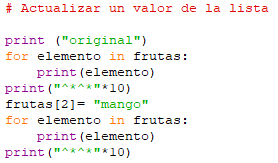
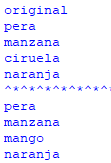
 

Para acceder por elemento debemos seleccionar la lista seguido del índice del elemento.

Los índices negativos son exclusivos de Python y vamos a poder ir en reversa del final hacia atrás,

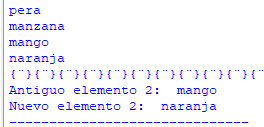
Actualizar un valor de la lista

Borrado de elementos

Para ello utilizo el comando del de la siguiente manera:

Del + nombre lista + índice del elemento.

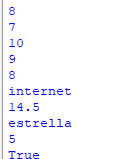
 

# Listas II - 15 - Python básico

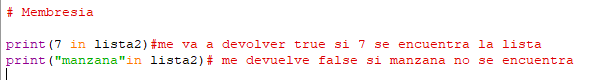


Los siguientes métodos son bastante sencillos así que solo subo las capturas.



Membresía:





Repetición:

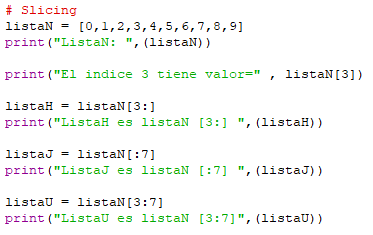


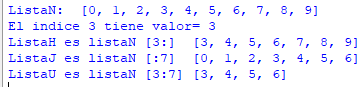


**Slicing:** en este caso siempre cuenta los índices de izquierda a derecha, en el 1er ejemplo toma del índice 3 hasta el final de la lista y lo recorta.

En el segundo caso toma hasta el índice 7 pero en Python el límite superior no se incluye.

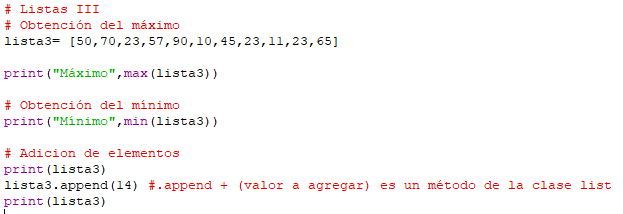
Por último, toma valores entre 3 y 7 siendo incluido el primero y no el último.

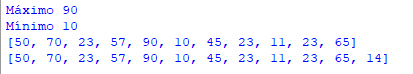




# Listas III - 16 - Python básico

Mas métodos y funciones

****

****

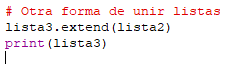
Conteo

** **

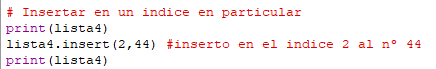
Encontrar el índice mas bajo del elemento repetido o donde aparece por primera vez

** **

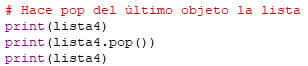
Otra forma de unir listas

** **

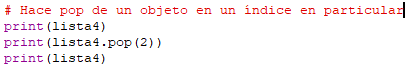
Insertar en un índice en particular

Pop elimina en este caso por no poner ningún parámetro al último elemento de la lista

pop de un objeto en un índice en particular seleccionando el índice.

Hacer la reversa de una lista

Ordenar una lista

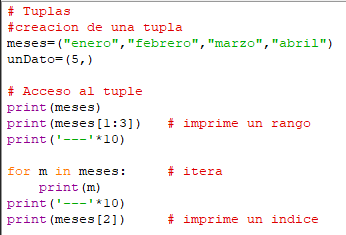
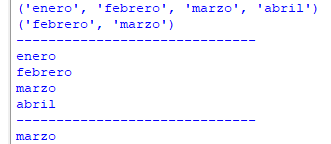
 

Tuple - 17 - Python básico

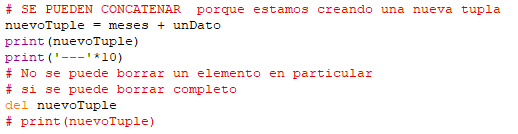
Las tuplas sin también secuencias de objetos, pero inmutables. Se utilizan () en lugar de [] . si no se usan se toma por default el tuple.

Los elementos se separan por comas, incluso si es un solo elemento, son más rápidas que las listas, si la información no va a cambiar es la mejor opción, se pueden usar como llaves para los diccionarios.

Creación y acceso:

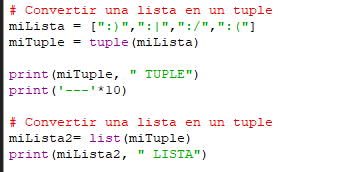
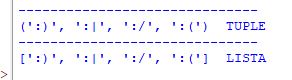
 

Concatenación y borrado:

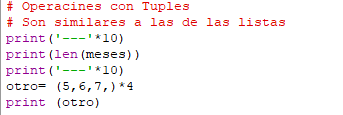




Las conversiones son mediante constructores tanto tuple como list.

Operaciones con tuplas son muy similares a las de las listas

Verifico si la tupla contiene un valor, me responde un booleano:

Asignación múltiple:





Otra creación:

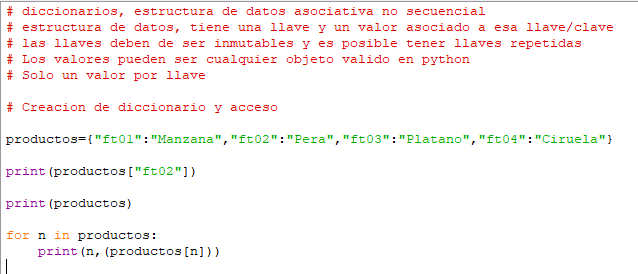




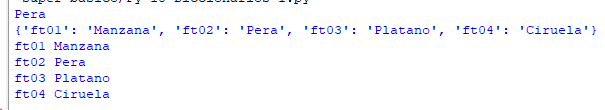
Máximos y mínimos:

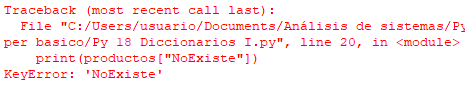
# Diccionario I - 18 - Python básico



Notar que no es una estructura de datos secuencial, lo importante son las claves asociadas y no su orden.

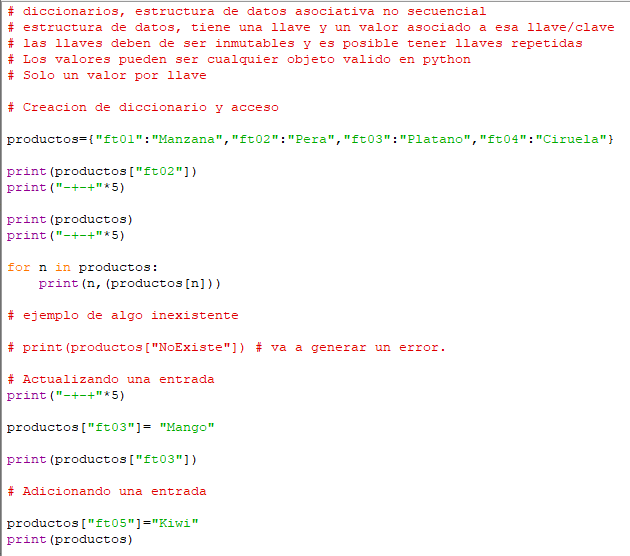


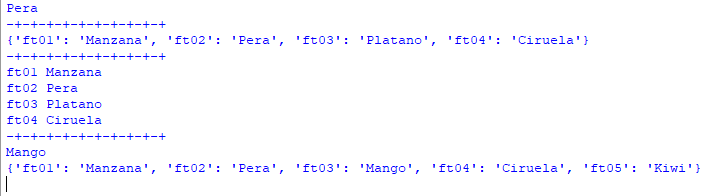
ejemplo de algo inexistente

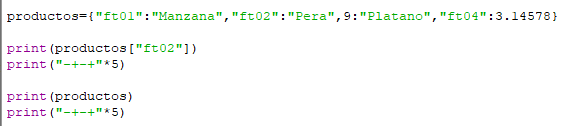
Simplemente da un error al no existir como clave/llave o valor.

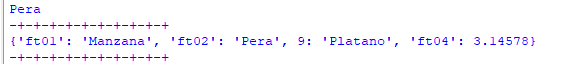
# Diccionario II - 19 - Python básico





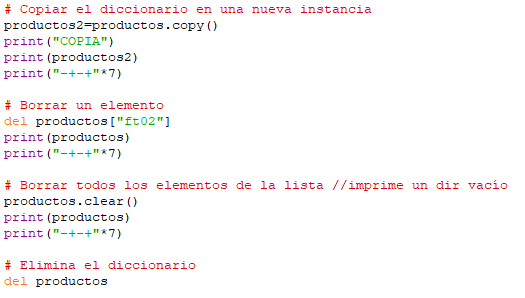
En el siguiente ejemplo voy a cambiar una clave/llave por un número y también un valor por un float.

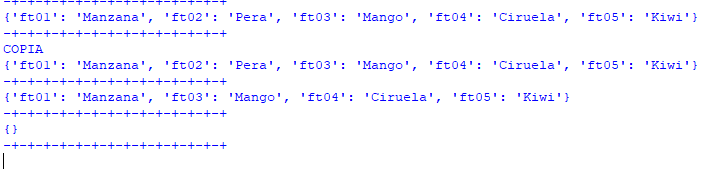




A continuación:

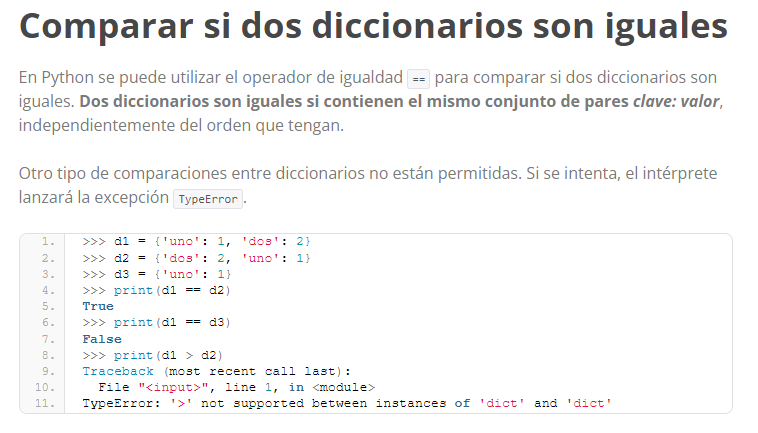
* copia de diccionario
* Borrado de un elemento (solo si existen si no dará un error)
* Borrado de todos los elementos de la lista (sin eliminar el diccionario) (utilizando el método clear)
* Eliminación del diccionario. (Si trato de imprimir dará un error)

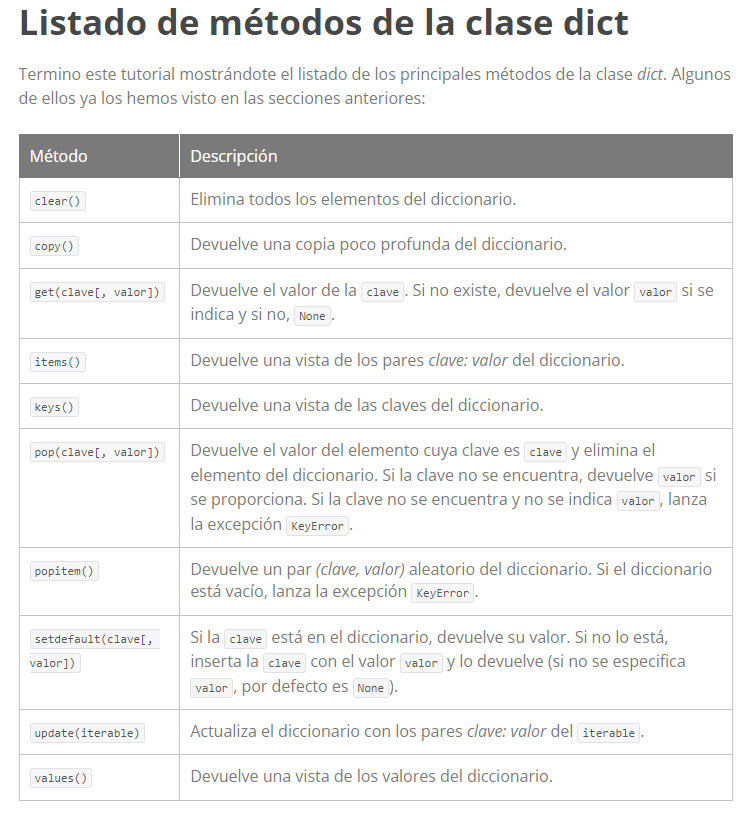


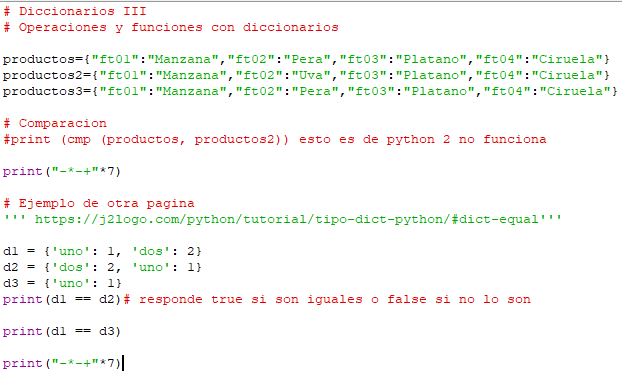


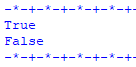
# Diccionario III - 20 - Python básico

<https://j2logo.com/python/tutorial/tipo-dict-python/#dict-equal> (explicación distinta a los videos muy completo y actualizado a Python 3)



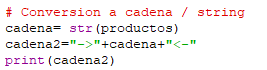






Longitud:  

Conversión a string: en el ejemplo primero de transforma a string/cadena, luego solo para ilustrar en la cadena 2 se concatena con símbolos para mostrar que opera como string.



Otro método de P 2.X



## Objetos vista de un diccionario

La clase dict implementa tres métodos muy particulares, dado que devuelven un tipo de dato, iterable, conocido como objetos vista. Estos objetos ofrecen una vista de las claves y valores contenidos en el diccionario y si el diccionario se modifica, dichos objetos se actualizan al instante.

Los métodos son los siguientes:

* keys(): Devuelve una vista de las claves del diccionario.
* values(): Devuelve una vista de los valores del diccionario.
* items(): Devuelve una vista de pares (clave, valor) del diccionario.

Imprimir las llaves:

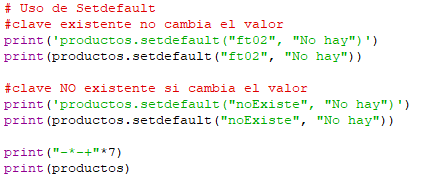
Imprimir los valores:

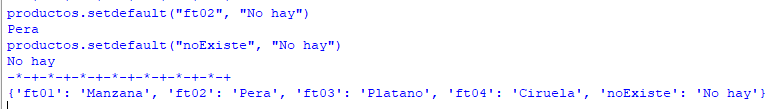
 

Imprimir los items:

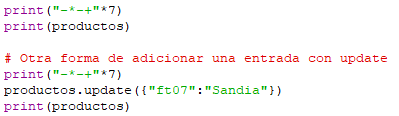


Uso de Set default: setdefault (clave[, valor]) Si la clave está en el diccionario, devuelve su valor. Si no lo está, inserta la clave con el valor valor y lo devuelve (si no se especifica valor, por defecto es None).





Al final como no existía "noExiste", "No hay" los agrego al diccionario.





# Set y frozenset - 21 - Python básico

El tipo set en Python es la clase utilizada por el lenguaje para representar los conjuntos. Un conjunto es una colección desordenada de elementos únicos, es decir, que no se repiten.

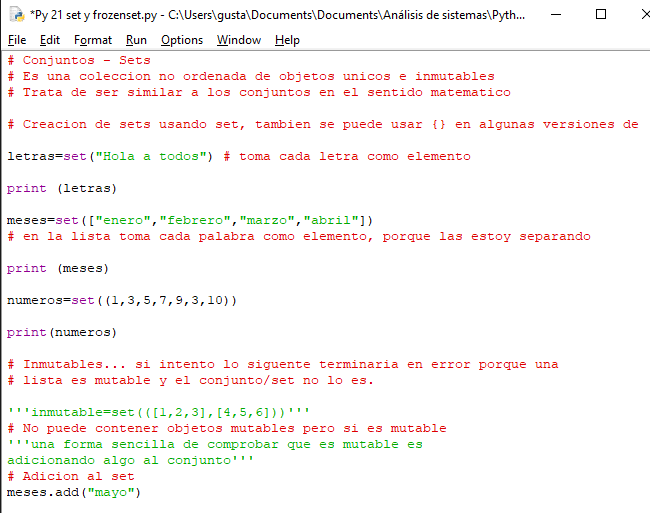
Estas características hacen que los principales usos de esta clase sean conocer si un elemento pertenece o no a una colección y eliminar duplicados de un tipo secuencial (*[list](https://j2logo.com/python/tutorial/tipo-list-python/)*, *[tuple](https://j2logo.com/python/tutorial/tipo-tuple-python/)* o *[str](https://j2logo.com/python/tutorial/tipo-str-python/)*).

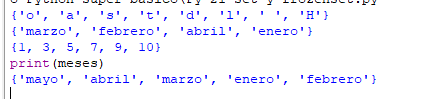
Además, esta clase también implementa las típicas operaciones matemáticas sobre conjuntos: *unión*, *intersección*, *diferencia*, …

Para crear un conjunto, basta con encerrar una serie de elementos entre llaves {}, o bien usar el constructor de la clase set() y pasarle como argumento un objeto *iterable* (como una *lista*, una *tupla*, una *cadena* …).

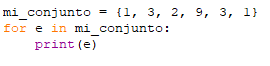
**IMPORTANTE:**{} NO crea un conjunto vacío, sino un *diccionario* vacío. Usa set() si quieres crear un conjunto sin elementos.

https://j2logo.com/python/tutorial/tipo-set-python/

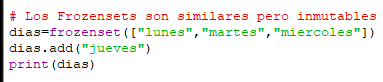


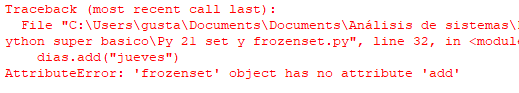


Se puede acceder y/o recorrer todos los elementos de un conjunto usando un bucle for:

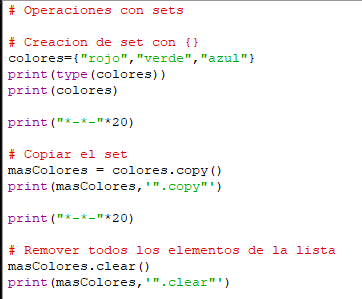
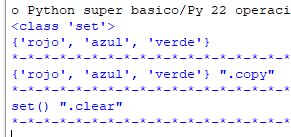
 

Con la utilización de frozenset el conjunto se vuelve inmutable, pero conserva las demás características.



 No tiene atributo add.

# Operaciones con set I- 22 - Python básico

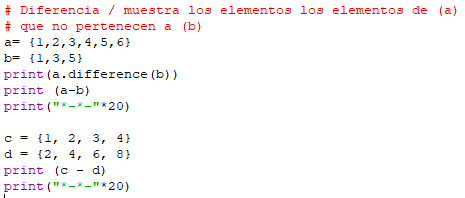
Clear va a eliminar solo el contenido NO al conjunto/set

Si quiero borrar el conjunto lo hago con del (del masColores)

**Diferencia de conjuntos en Python**

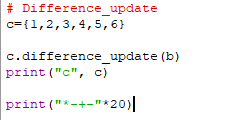
La diferencia entre dos conjuntos A y B es el conjunto A \ B que contiene todos los elementos de A que no pertenecen a B.

En Python se utiliza el operador (-) o método (.difference)  para realizar la diferencia de dos o más conjuntos.

En este caso el conjunto (a) no se ve afectado, en el siguiente caso si será modificado.

Difference\_update va a modificar también al conjunto (a) en el caso anterior vemos la impresión de (a) y (c) y son diferentes. Va a guardar en el conjunto los resultados de esa diferencia

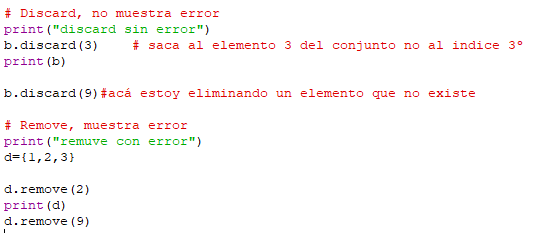
**Eliminar un elemento de un conjunto en Python**

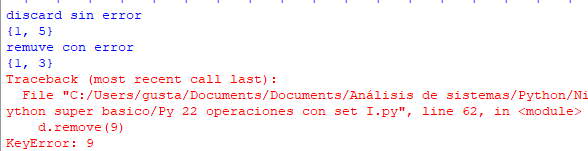
La clase *set* ofrece cuatro métodos para eliminar elementos de un conjunto. Son: discard(), remove(), pop() y clear(). A continuación te explico qué hace cada uno de ellos.

discard(elemento) y remove(elemento) eliminan elemento del conjunto. La única diferencia es que si elemento no existe, discard() no hace nada mientras que remove() lanza la excepción KeyError.

pop() es un tanto peculiar. Este método devuelve un elemento aleatorio del conjunto y lo elimina del mismo. Si el conjunto está vacío, lanza la excepción KeyError.

Finalmente, clear() elimina todos los elementos contenidos en el conjunto.



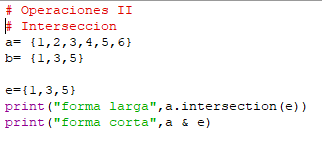
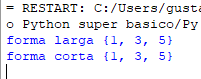


# Operaciones con set II- 23 - Python básico

**Intersección de conjuntos en Python**

La intersección de dos conjuntos A y B es el conjunto A ∩ B que contiene todos los elementos comunes de A y B.

En Python se utiliza el operador & o “intersection” para realizar la intersección de dos o más conjuntos.

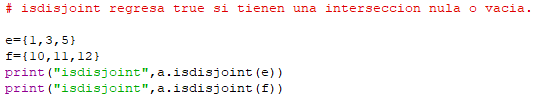
 

**Conjuntos disjuntos en Python**

Dos conjuntos A y B son disjuntos si no tienen elementos en común, es decir, la intersección de A y B es el conjunto vacío.

En Python se utiliza el método isdisjoint() de la clase set para comprobar si un conjunto es disjunto de otro.

Isdisjoint (iterable) Devuelve True si dos conjuntos son disjuntos.

**Inclusión de conjuntos en Python**

Dado un conjunto A, subcolección (issubset o subconjunto) del conjunto B o igual a este, sus elementos son un subconjunto de B. Es decir, A es un subconjunto de B y B es un superconjunto de A.

En Python se utiliza el operador <= para comprobar si un conjunto A es subconjunto de B y el operador >= para comprobar si un conjunto A es superconjunto de B.

>>> a = {1, 2}

>>> b = {1, 2, 3, 4}

>>> a <= b

True

>>> a >= b

False

>>> b >= a

True

>>> a = {1, 2}

>>> b = {1, 2}

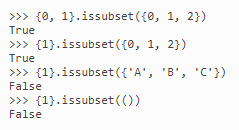
>>> a < b # Ojo al operador < sin el =

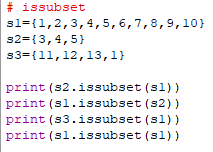
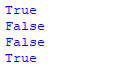
False

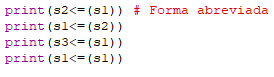
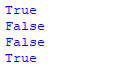
>>> a <= b

True

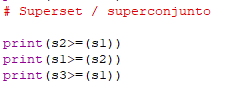
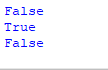
**Issubset** Devuelve un valor booleano que indica si el conjunto está contenido en el conjunto especificado o es iterable.

 <https://Python-reference.readthedocs.io/en/latest/docs/sets/issubset.html>

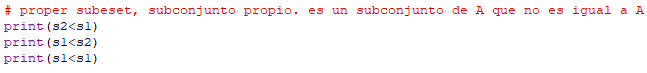
 

  Forma abreviada

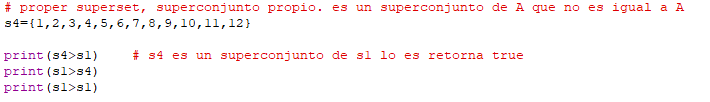
Superconjunto

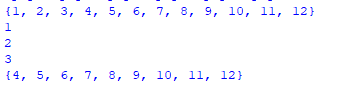
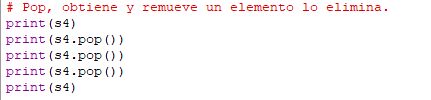
Subconjunto propio / proper subset. **Es un subconjunto de A que no es igual a A**

Superconjunto propio / proper superset. Es un superconjunto de A que no es igual a A

**Pop**, obtiene y remueve un elemento lo elimina.



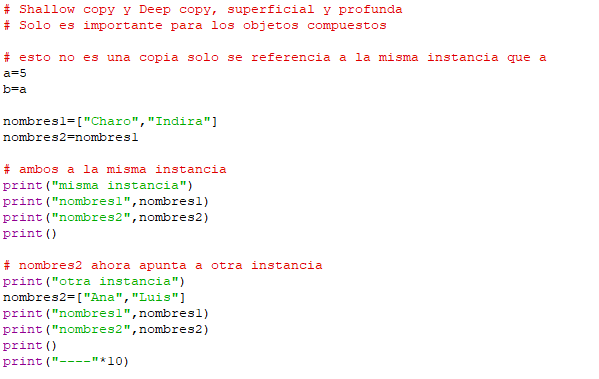
# Shallow y deep copy - 24 - Python básico

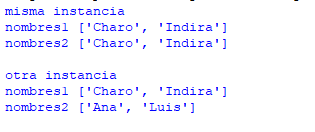
Shallow copy y Deep copy, superficial y profunda.

Las declaraciones de asignación en Python no copian objetos, crean enlaces entre un objetivo y un objeto. Para las colecciones que son mutables o contienen elementos mutables, a veces se necesita una copia para poder cambiar una copia sin cambiar la otra. Este módulo proporciona operaciones genéricas de copia superficial y profunda.

La diferencia entre copia superficial y profunda solo es relevante para objetos compuestos (objetos que contienen otros objetos, como listas o instancias de clase):

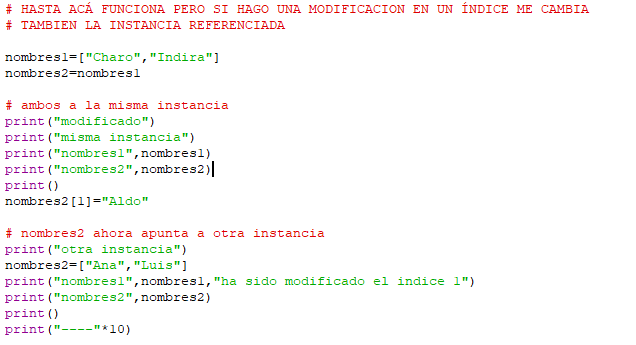
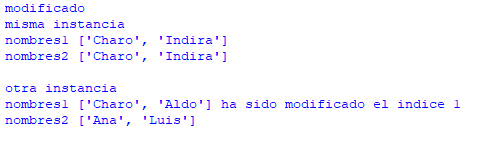
* Una copia superficial construye un nuevo objeto compuesto y luego (en la medida de lo posible) inserta referencias a los objetos que se encuentran en el original.
* Una copia profunda construye un nuevo objeto compuesto y luego, recursivamente, inserta copias de los objetos encontrados en el original.



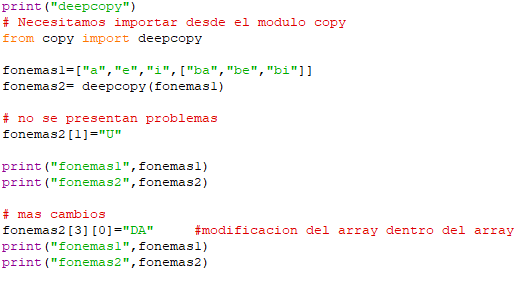


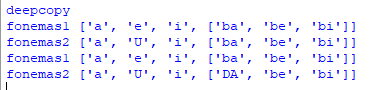
Hasta acá funciona pero si hago una modificación en un índice me cambia también la instancia referenciada.

En el ejemplo siguiente la modificación la hago en el índice 1 de la variable 2 (nombres2) pero lo modifica también en nombres1

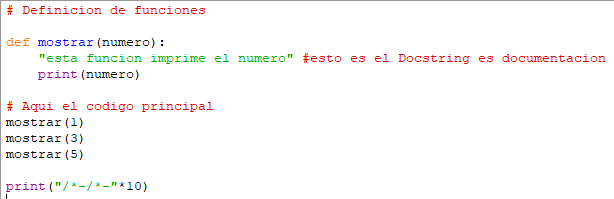


Ejemplo de deep copy

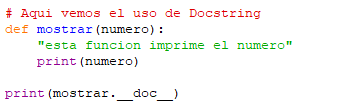




# Funciones I - 25 - Python básico

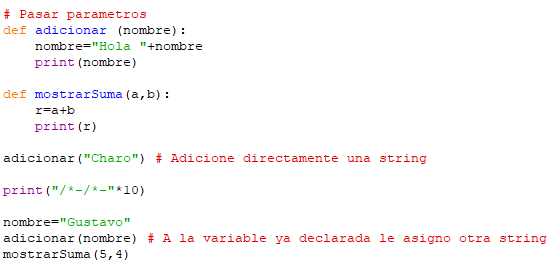
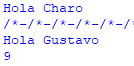


En este ejemplo definí la función con nombre mostrar y le pase como parámetro luego la variable **número**, para invocarla use el nombre de la función y entre paréntesis asigne números como parámetros.

Con la sintaxis: (**nombre de función** + **.** + *dos guiones bajos* + **doc** + *dos guiones bajos*)

Imprimo la documentación de la función.

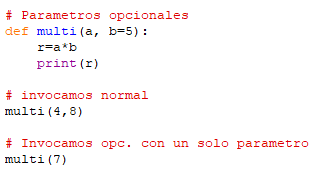
 

<https://www.youtube.com/watch?v=RAMt3ks4fLk&list=PLM-p96nOrGcZmWE-320vKw-e6TJsle7rC&index=25>

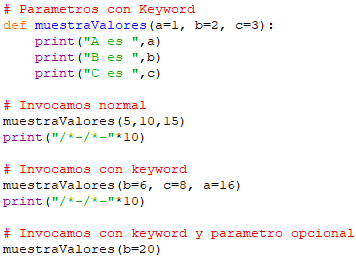
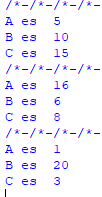
Minuto 6.

En el primer ejemplo invoco la función y le paso una string como parámetro, pasa por dentro de la func. Y arroja “hola Charo”.

En el segundo ejemplo también invoco la función con su parámetro predeterminado, pero a ese le doy un valor de string a su variable.

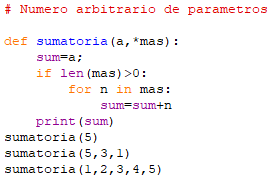
En este ejemplo al parámetro b le asigno un valor de 5, en la invocación normal invoco los dos valores en la siguiente solo asigno un valor que queda en a y b toma el valor opcional que le asigne en la función.

En la invocación normal y con keyword asigne todos los valores y fueron impresos luego en la última utilizo solo un parámetro y toman lugar los opcionales. (si dejaba los param. Vacíos se imprimían solo los opcionales)

Otro detalle que al utilizar las keyword no importó el orden.

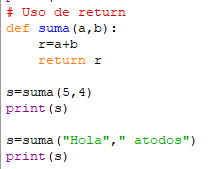
# Funciones II - 26 - Python básico

Con este ejemplo puedo tener una cantidad **n** de parámetros.

1. Declaro la función sumatoria y establezco que como mínimo necesito el parámetro **a**, y en este caso el **\*** funciona como un comodín para indicar que en la variable **mas** voy a tener una cantidad arbitraria de parámetros.
2. Al haber un solo parámetro va a caer en **a**, los siguientes sean cuantos sean van a caer en **\*mas**, luego entran a la función donde voy a iterar con un bucle for y los va sumando, por ultimo imprimo el total acumulado en la variable sum.

Uso de return

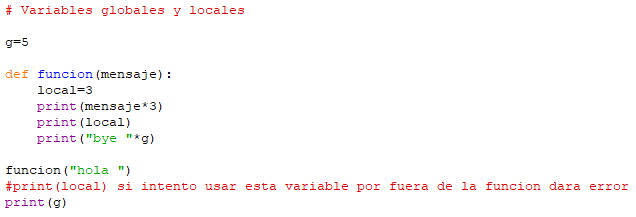
 

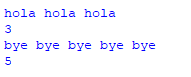
En este caso con return voy a cortar la ejecución de la función y obtener el retorno de la operación dentro de la misma. Abajo cuando uso el código al invocar la función le asigno a la variable **s** el resultado de suma con los parámetros que también le di.

En el segundo ejemplo hago uso del ***polimorfismo*** y sumo dos strings.

Variables globales y locales

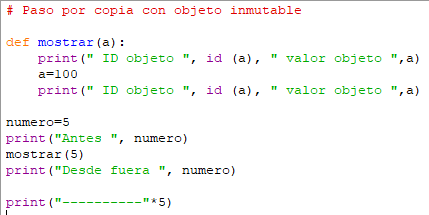
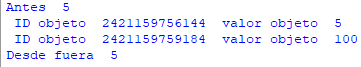
Las globales serán conocidas por todo el programa y las locales solo por la función a la que pertenecen.





# Paso de parámetros - 27 - Python básico

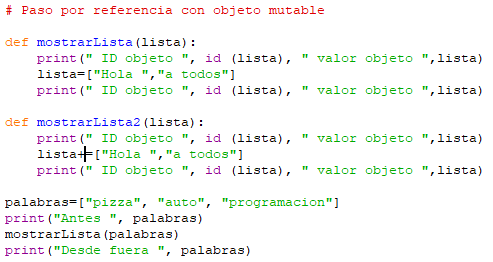
Paso por copia con objeto inmutable

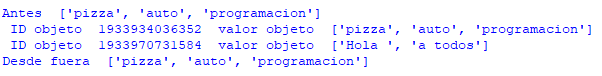
 

Al utilizar el método id nos demuestra si es una copia o la misma instancia, al pasar el 5 por dentro de la función vemos que se creó una copia referenciada en otra instancia. (Los enteros en Python son inmutables)

Paso por referencia con objeto mutable

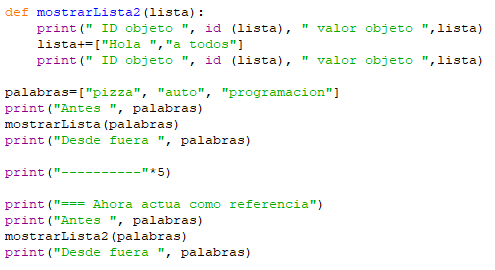
En el primer ejemplo voy a usar la función mostrarLista (solamente), similar al caso anterior imprimo modifico y vuelvo a imprimir, y por fuera 1, imprimo la lista. 2, invoco a la función, 3, vuelvo a imprimir luego de pasar la lista por la función.

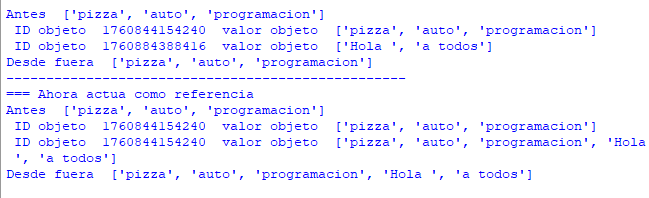




En este caso paso como si fuera una copia no por referencia

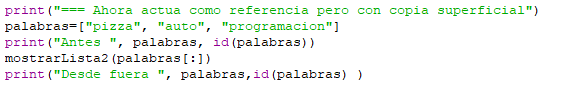
Ejemplo 2 mostrarLista2

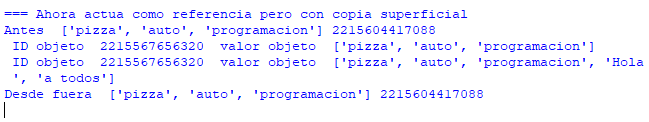




En el ejemplo 2 se ve como la concatenación ocurre en la misma instancia.

Ahora actúa como referencia, pero con copia superficial





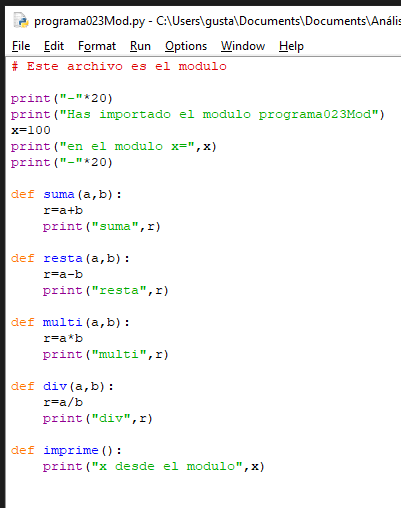
# Módulos - 28 - Python básico

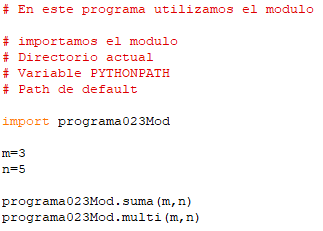
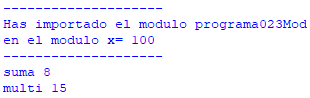
Básicamente los módulos nos permiten la reutilización de código. Los módulos y paquetes en Python son la forma de organizar los scripts y programas a medida que estos crecen en número de líneas de código. Por otro lado, un espacio de nombres define los límites en que se puede utilizar un nombre o identificador determinado. <https://j2logo.com/python/tutorial/espacios-de-nombres-modulos-y-paquetes/#python-namespaces>

En el ejemplo creamos un módulo con cálculos básicos para luego importarlo las veces que sea necesario desde el programa principal. **programa023Mod** será el nombre el módulo que vamos a llamar.

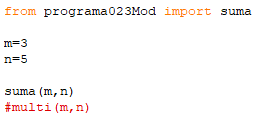
Para importar Python va a seguir una serie de reglas, primero lo va a buscar en la carpeta/directorio donde lo estamos ejecutando, si no lo encuentra ahí los va a buscar en PYTHONPATH y por último en path de default.

* La sintaxis es con la palabra reservada import seguido del nombre del módulo
* También podemos importar una función en particular, sintaxis: from **programa023Mod** import suma



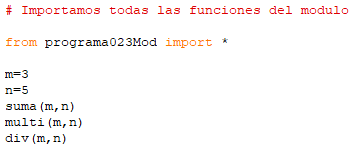
 

Importo el módulo y llamo a las funciones.

Importo solo la función suma, si uso multi en este caso dará error. Y si quiero importar más de una función las separo con comas.

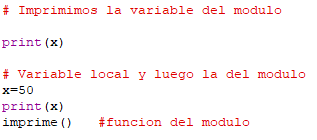
Importo todas las funciones

# Módulo y paquete - 29 - Python básico

Con los paquetes podemos agrupar lógicamente varios módulos, en esta lección veremos cómo crear paquetes en Python.

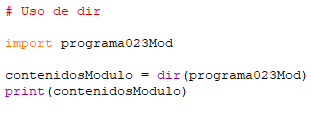
Variable local y luego la del modulo

El resultado muestra que coexisten las dos variables la local y la del módulo sin que haya conflicto. ¡Muy importante definir bien el ámbito de las variables utilizadas!

Uso de **dir**

Cree una variable para almacenar la lista de funciones del módulo





## Nombres y espacios de nombres en Python (NAMESPACE)

### Nombres

Lo primero que debes tener claro antes de profundizar en los conceptos es qué es un nombre.

**En Python todo es un objeto**. El número 2 es un objeto, el texto ‘Hola mundo’ es un objeto, las funciones son objetos, … Pues bien, **un nombre o identificador es la forma que existe en Python de referenciar a un objeto concreto. Equivaldría al concepto de variable**. En definitiva, una variable no es más que el nombre con el que nos referimos a un objeto que existe en memoria.

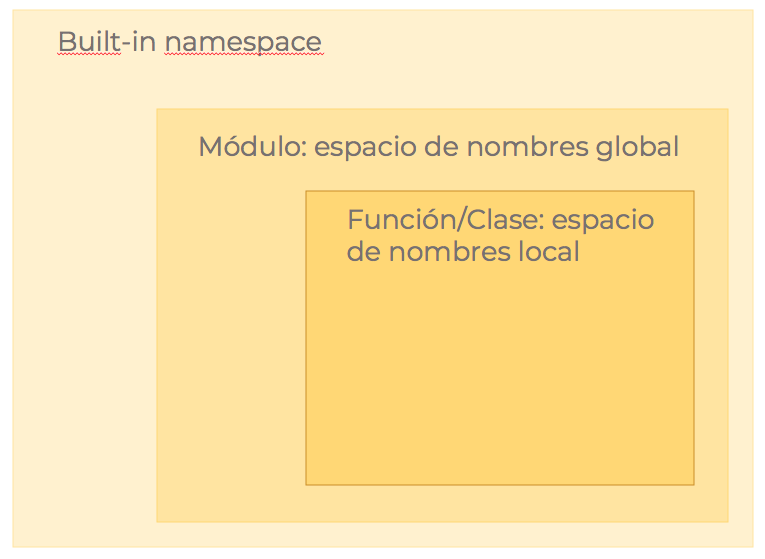
### Espacios de nombres (NAMESPACE)

Una vez aclarado este término, paso a explicarte qué son los espacios de nombres en Python. **Un espacio de nombres es una colección aislada de nombres (o identificadores) que referencian a objetos.** Como veremos a continuación, en un mismo script o programa Python pueden coexistir varios espacios de nombres a la vez.

Cuando accedemos a un intérprete de Python o ejecutamos un programa, **todos los identificadores que define el lenguaje son añadidos a un espacio de nombres al que es posible acceder desde cualquier punto de un script**. Es por esto que las funciones como print() o len() están siempre accesibles. Este espacio de nombres es conocido como espacio de nombres incorporado (o built-in namespace)

Además, **cada módulo en Python crea su propio espacio de nombres global**. Como te decía, los espacios de nombres están aislados. Esa es la razón por la que en diferentes módulos se pueden usar los mismos nombres y estos no interfieren entre sí.

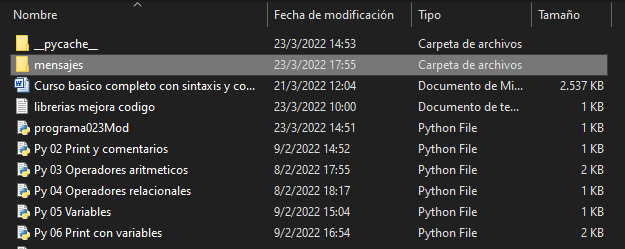
A su vez, en un módulo existen funciones y clases. **Cuando se invoca a una función se crea un espacio de nombres local asociado a dicha función que contiene todos los nombres definidos dentro de la misma** (sucede algo similar para las clases).



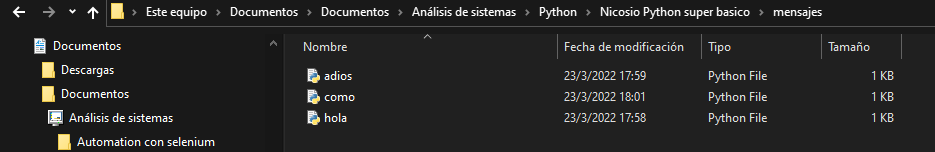
Importar un paquete

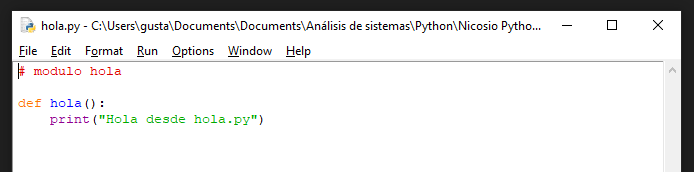
Un paquete va a ser una *colección* de módulos organizados de una manera lógica esto será dentro de una *carpeta/directorio*, al crear el paquete debemos crear también un **archivo** que dirá cuales cosas vamos a importar…

En mi carpeta de este curso creo una llamada mensajes y dentro voy a crear varios módulos



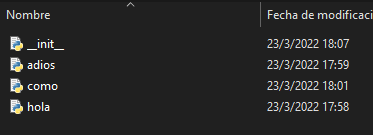
También creo los módulos: hola, como y adiós dentro de mensajes:



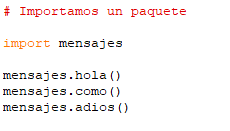


Para que funcione como paquete debo crear el archivo \_\_init\_\_ ( \_ \_ init \_ \_ )

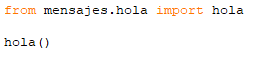
* lo creo como cualquier programa…

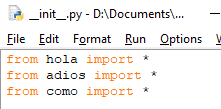
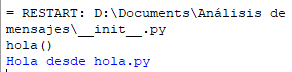
**Esto falló, pero lo dejo como ejemplo.**



Esto último me falló y solo con la siguiente sintaxis pude resolver. El archivo \_\_init\_\_ debió quedar vacío porque tampoco funcionaba.

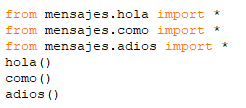


Algo que me confunde bastante, si desde dentro del archivo init corro con f5 e invoco las funciones estas corren bien. Si lo hago desde otro programa intentando importar no corren.

Encontré en otro video la siguiente manera con el init aun vacío y corrió también.

<https://www.youtube.com/watch?v=sgcTujbQhmA>

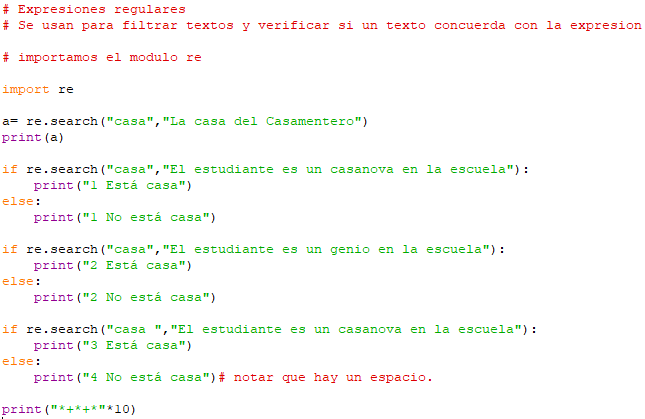
 

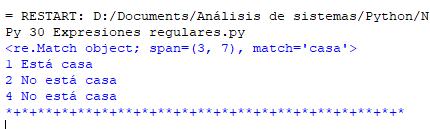
Otra buena explicación:

<https://www.youtube.com/watch?v=rwUVwxh5N74>

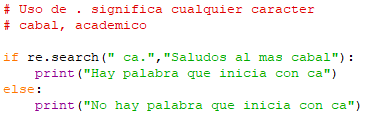
# Expresiones regulares - 30 - Python básico

Función search.

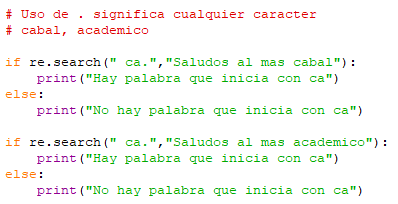




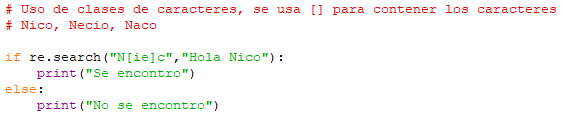
Uso de (.) como comodín.

En este ejemplo va a buscar palabras que inician con ca y el (.) va a representar cualquier otra letra a continuación.

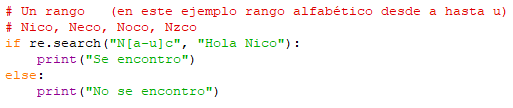
En el segundo caso no hay espacio antes de ca por eso no lo valida.

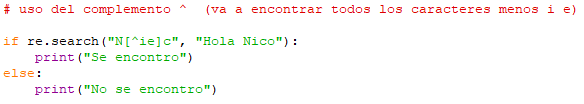
 

En este caso la búsqueda se entiende de la siguiente manera: “N[ie]c” la palabra buscada comenzara con N mayúscula luego [ie] quiere decir que puede contener ‘i’ o ‘e’ y luego la c.

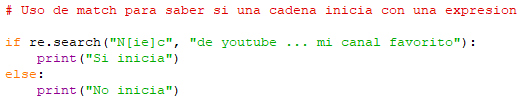
Si cambiara la N o la c y no contuviera ‘i’ o ‘e’ el resultado sería NO se encontró.

# Expresiones regulares II - 31 - Python básico

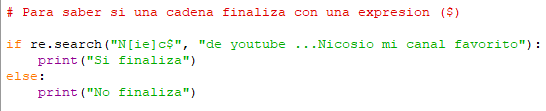
 cualquier letra pasando la u queda fuera de la búsqueda.

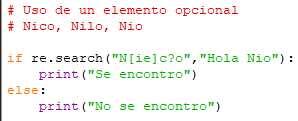
Con el símbolo “ **^** ” le digo que busque todo menos los valores a continuación, **Muy útil para eliminar.**

La expresión **match** solamente busca al inicio de la expresión.

Con el símbolo “ $ “ voy a buscar si una frase termina con las letras consultadas.



Los resultados serian con:

Nico = se encontró

Nilo = no se encontró

Nio = se encontró

En este caso con el signo “ ¿ “voy a decir que ese carácter puede estar o no.

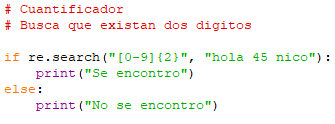
La expresión entera seria, tiene que tener una **N**, luego puede ser una **i** o una **e**, puede tener o no una **c** y por último tiene que tener una **o**.

Los resultados serian con:

Hola 45 Nico **=** se encontró

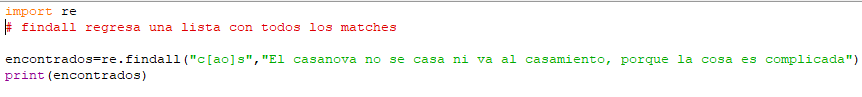
Si cambio a “N [0-9] {2}” **=** no se encontró porque va a pedir que el numero tenga antes una N

Si cambio a “[0-9] {3}” **=** no se encontró porque faltaría un numero



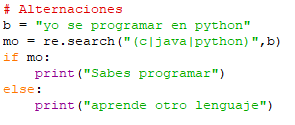
En esta búsqueda le digo que en el rango de **[0 a 9]** deben haber **{2}** cifras, [] = rango, {} = cuantificador.

# Expresiones regulares y cadenas - 32 - Python básico (no hay video 33 en el canal)

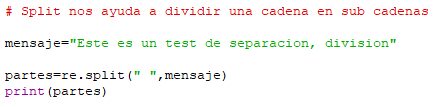




Nos regresa una lista de todas las veces que se encuentre un match en el texto indicado.

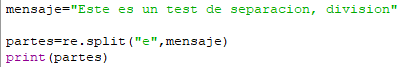
 

En las alternaciones mediante el símbolo | que equivale al operador lógico OR le doy una lista de palabras que pueden validar la búsqueda, luego como 2do parámetro agrego la variable b a la cual ya le había asignado una string.



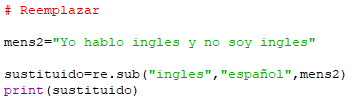


Por los parámetros brindados va a crear una separación cada vez que encuentre un espacio.



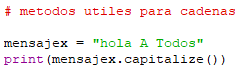
 para mejor ilustración cree una separación con la letra “e”.

Reemplazos

En este ejemplo en el primer parámetro digo cual es la expresión que quiero reemplazar, en el 2do con cual lo reemplaza y el 3ero en que cadena hago la operación.

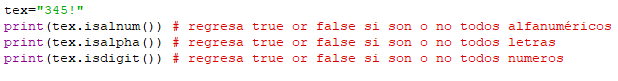
Métodos útiles

  en este caso el 1er parámetro (15) es el tamaño de la nueva cadena y el 2do es el carácter que se va a utilizar para rellenar a izquierda y derecha.

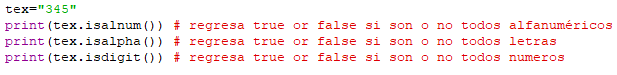
Simplemente me indica cuantas veces se repite el texto parámetro dentro de la cadena.

Isalnum significa alfanumérico

isalpha significa solo letras (no caracteres especiales)

Isdigit solo números

# Operador Lambda - 34 - Python básico

**Qué son las funciones lambda en Python** [**https://j2logo.com/python/funciones-lambda-en-python/**](https://j2logo.com/python/funciones-lambda-en-python/)

En Python, las *funciones lambda* son también conocidas como *funciones anónimas* porque se definen sin un nombre.

Una función en Python se define con la palabra reservada def. Sin embargo, una *función anónima* se define con la palabra reservada lambda.

* Son funciones que pueden definir cualquier número de parámetros, pero una única expresión. Esta expresión es evaluada y devuelta.
* Se pueden usar en cualquier lugar en el que una función sea requerida.
* Estas funciones están restringidas al uso de una sola expresión.
* Se suelen usar en combinación con otras funciones, generalmente como argumentos de otra función.
* *Si solo se van a usar una vez pueden ser convenientes, si se vuelven repetitivas claramente es mejor crear una función.*

Ejemplo:

cuadrado = lambda x: x \*\* 2

En el ejemplo anterior, x es el parámetro y x \*\* 2 la expresión que se evalúa y se devuelve.

Como ves, la función no tiene nombre y toda la definición devuelve una función que se asigna al identificador cuadrado.

En el siguiente ejemplo se aprecian las similitudes y diferencias de usar una función anónima y una función normal:

>>> def cuadrado(x):

... return x \*\* 2

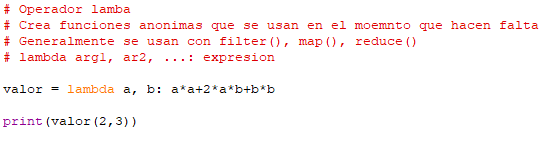
>>> cuad = lambda x: x \*\* 2

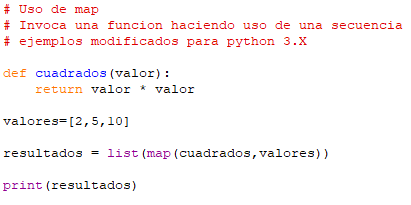
>>> print(cuadrado(3))

9

>>> print(cuad(5))

25

### map()

La función map() en Python aplica una función a cada uno de los elementos de una lista.

map(una\_funcion, una\_lista)

Imagina que tienes una lista de enteros y quieres obtener una nueva lista con el cuadrado de cada uno de ellos.

Seguramente, lo primero que se te ha ocurrido es algo similar a lo siguiente:

enteros = [1, 2, 4, 7]

cuadrados = []

for e in enteros:

cuadrados.append(e \*\* 2)

print(cuadrados)

[1, 4, 16, 49]

Sin embargo, podemos usar una función anónima en combinación con map() para obtener el mismo resultado de una manera mucho más simple:

enteros = [1, 2, 4, 7]

cuadrados = list(map(lambda x : x \*\* 2, enteros))

print(cuadrados)

[1, 4, 16, 49]

La cosa se vuelve todavía más interesante cuando, en lugar de una lista de valores, pasamos como segundo parámetro una lista de funciones:

enteros = [1, 2, 4, 7]

def cuadrado(x):

return x \*\* 2

def cubo(x):

return x \*\* 3

funciones = [cuadrado, cubo]

for e in enteros:

valores = list(map(lambda x : x(e), funciones))

print(valores)

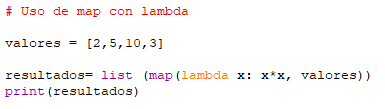
[1, 1]

[4, 8]

[16, 64]

[49, 343]

MAP agregando lambda

### filter()

La función filter() filtra una lista de elementos para los que una función devuelve True.

filter(una\_funcion, una\_lista)

Imagina que quieres filtrar una lista de números para obtener solo los valores pares.

De nuevo, una primera aproximación podría ser como la siguiente:

valores = [1, 2, 3, 4, 5, 6, 7, 8, 9]

pares = []

for valor in valores:

if valor % 2 == 0:

pares.append(valor)

print(pares)

[2, 4, 6, 8]

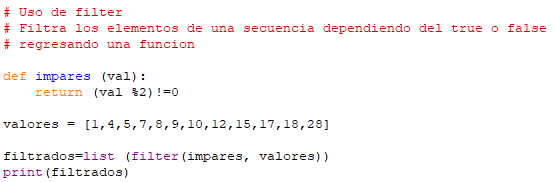
No obstante, podemos usar la función filter() y una función lambda para obtener el mismo resultado con una sola línea de código:

valores = [1, 2, 3, 4, 5, 6, 7, 8, 9]

pares = list(filter(lambda x : x % 2 == 0, valores))

print(pares)

[2, 4, 6, 8]

### reduce()

La última función de esta serie que vamos a ver es la función reduce(). Esta función se utiliza principalmente para llevar a cabo un cálculo acumulativo sobre una lista de valores y devolver el resultado.

La función reduce() está incluida en el módulo functools.

En este caso, la función que se pasa como primer parámetro recibe dos argumentos.

Imagina que quieres obtener el resultado de sumar todos los elementos de una lista.

Como en las veces anteriores, la suma la puedes calcular de la siguiente manera:

valores = [2, 4, 6, 5, 4]

suma = 0

for el in valores:

suma += el

print(suma)

21

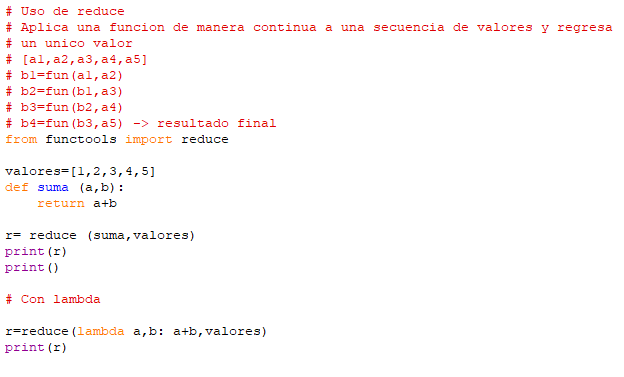
Pero también usando la función reduce() en combinación con una función lambda:

from functools import reduce

suma = reduce(lambda x, y: x + y, valores)

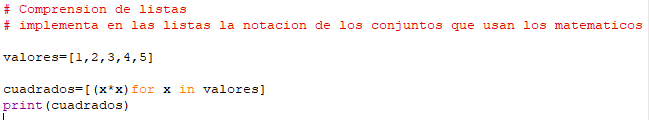
print(suma)

21



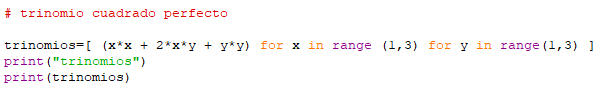
Lo que se ve en los comentarios es la explicación de como va sumando y almacenando ese resultado hasta el final de las sumas, b1 es el resultado de a1 + a2, luego b2 va a almacenar el resultado de b1 + a3 ….

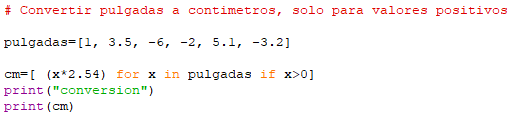
# Comprensión - 35 - Python básico



 Cuadrados va a ser igual al conjunto en el que x\*x es ejecutado con cada valor de la lista valores.

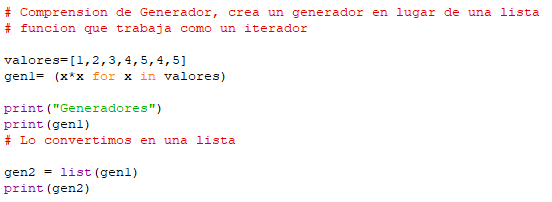
Calculo del cuadrado del trinomio perfecto para X e Y, en el rango de (1,3) que seria 1 y 2 porque en Python el ultimo valor no es incluido. **(utilización de rangos en el cálculo)**

  los resultados son simplemente las combinaciones de 1 y 2 reemplazando a x e y.



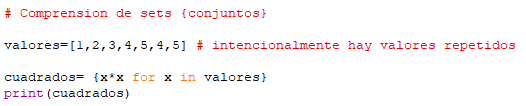
 en este ejemplo voy a convertir todas las x en la lista pulgadas siempre y cuando sean mayores a 0. **(utilización de condiciones en el cálculo).**

Generadores, en este curso todavía no los vimos, pero básicamente son iteradores. **(los paréntesis me dicen que va a ser un generador)**





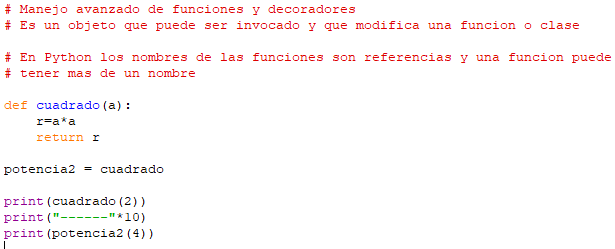
Conjuntos/sets **(las llaves {} me indican que el resultado va ser un conjunto)**



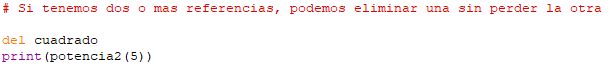


# Manejo avanzado de funciones - 36 - Python básico

Algo diferente entre Python y otros lenguajes es que las funciones aquí son objetos, el nombre de la función hace referencia a una instancia (objeto función)

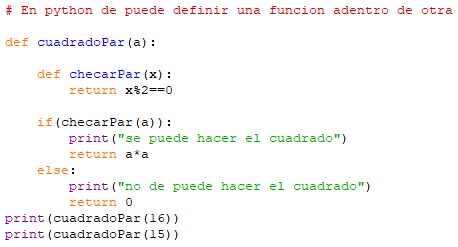
 

Tanto la función como la variable potencia2 hacen referencia a la misma instancia.

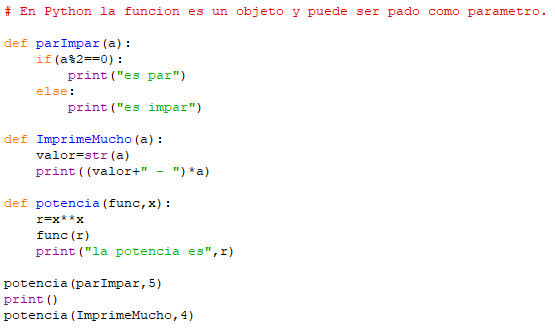
 

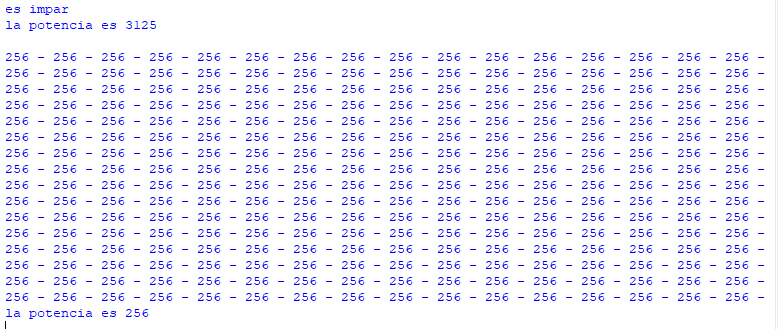
Borre la función, pero como en Python las instancias no se borran solo dejan de ser referenciadas potencia2 funciona normalmente.

Función dentro de otra.

Función como parámetro



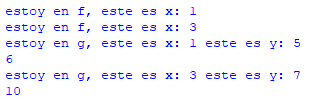
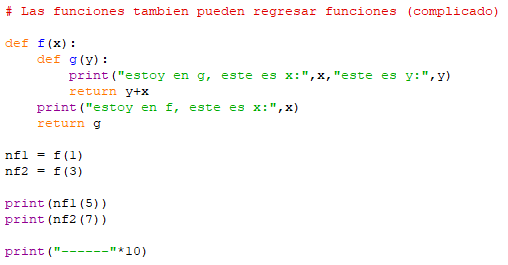


La primera función solo dice si es par o impar.

La segunda imprime al número pasado como parámetro la cantidad de veces que valga ese número, un 5 cinco veces etc.

La tercera es la función a la que le voy a pasar otra función como parámetro, más un valor al cual voy a trabajar dentro de la función, luego en la otra línea invoco la función que pase como parámetro y le doy el valor resultado (r) como su parámetro para operar…

Las funciones pueden regresar funciones… (difícil)



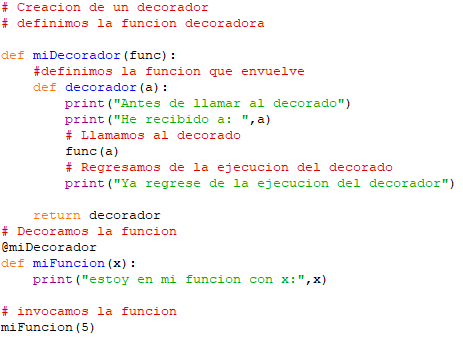
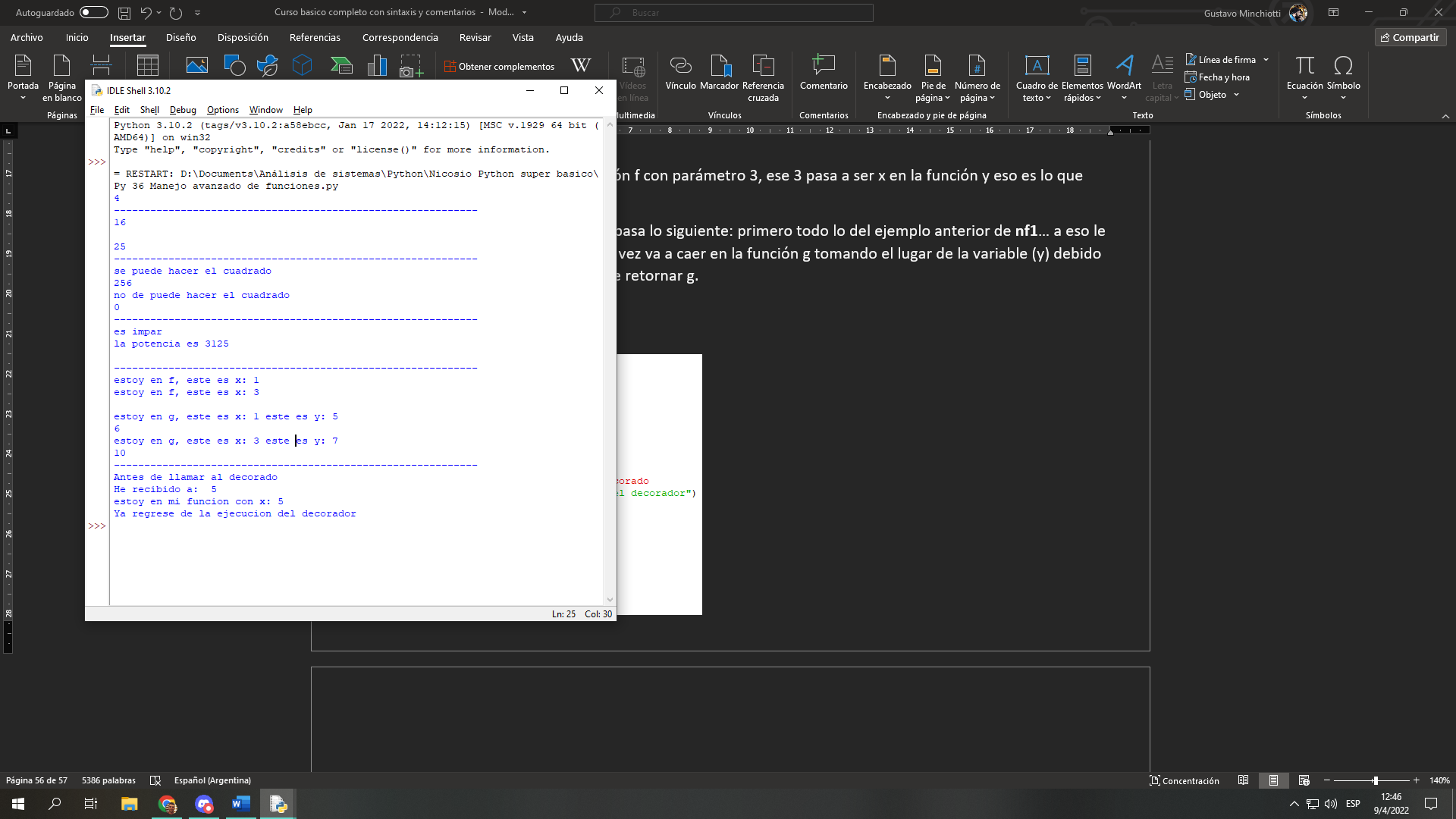
A nf1 le pase como parámetro la función f con parámetro 1, ese 1 pasa a ser x en la función y eso es lo que imprime.

A nf2 le pase como parámetro la función f con parámetro 3, ese 3 pasa a ser x en la función y eso es lo que imprime.

En los ejemplos de abajo print(nf1(5)) pasa lo siguiente: primero todo lo del ejemplo anterior de **nf1**… a eso le paso un nuevo parámetro (5) que esta vez va a caer en la función g tomando el lugar de la variable (y) debido a todo eso va a estar en condiciones de retornar g.

VER @staticmethod decorador <https://www.delftstack.com/es/howto/python/python-static-method/>

Creación de un decorador

**3**

**6**

**5**

**44**

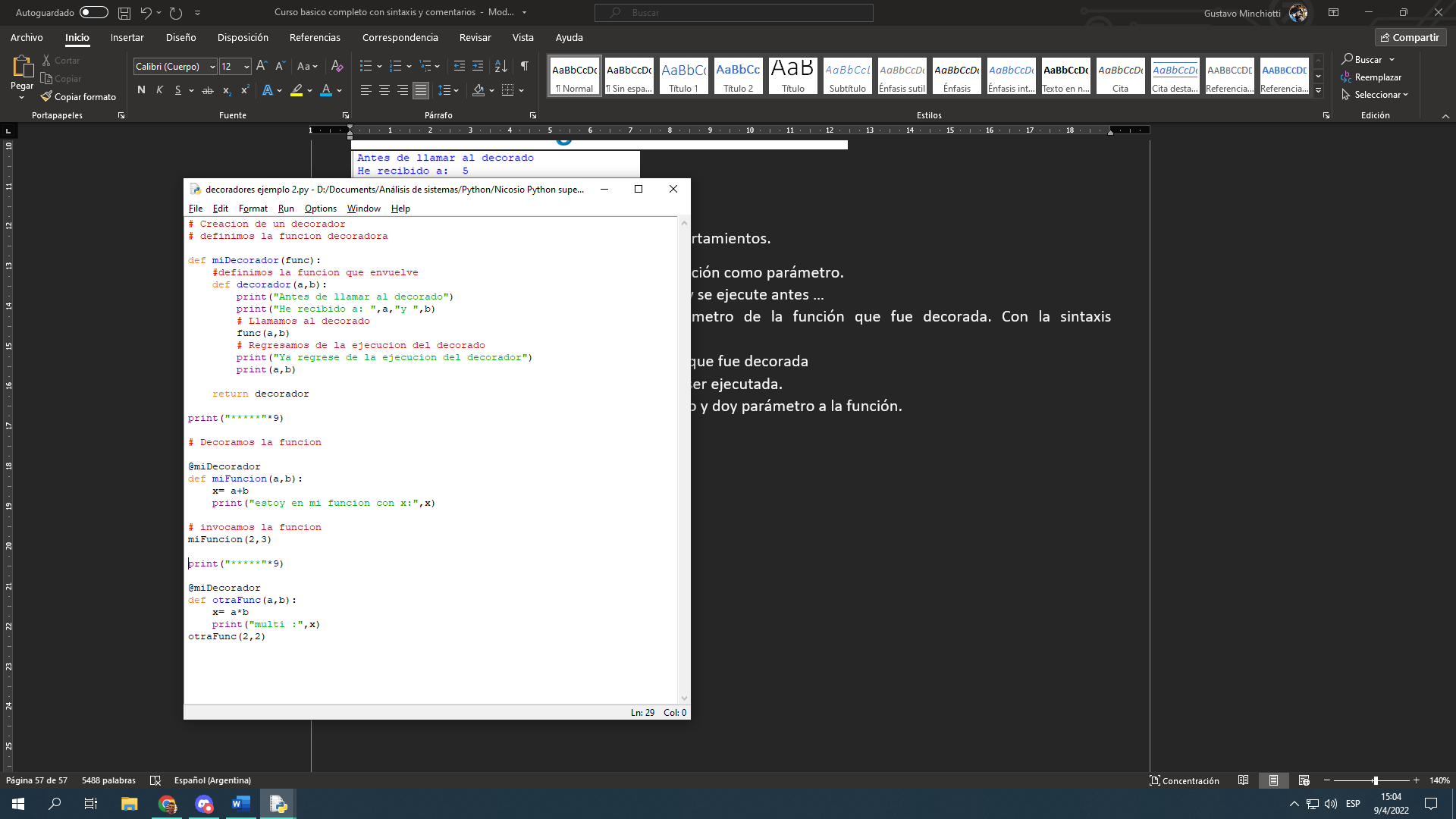
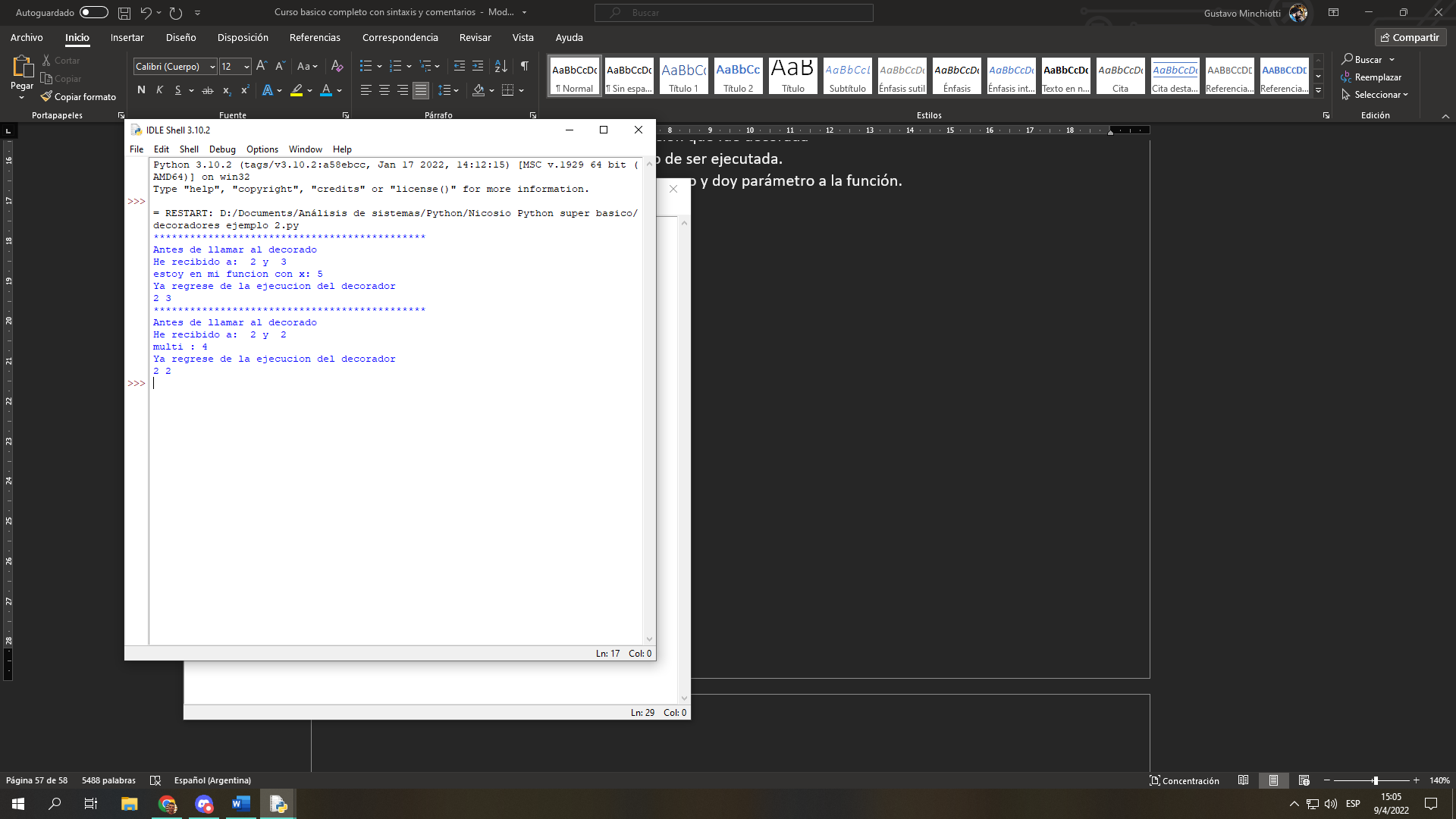
**2**

**1**

Al decorar a una función le agrego nuevos comportamientos.

1. Creo el decorador que va a recibir una función como parámetro.
2. Puedo crear una función que la envuelva y se ejecute antes …
3. Llamo al *decorado* que recibe el parámetro de la función que fue decorada. Con la sintaxis @miDecorador
4. Dentro del decorador **invoco** a la función que fue decorada
5. Regreso de la función decorado luego de ser ejecutada.
6. En el programa para que corra todo invoco y doy parámetro a la función.

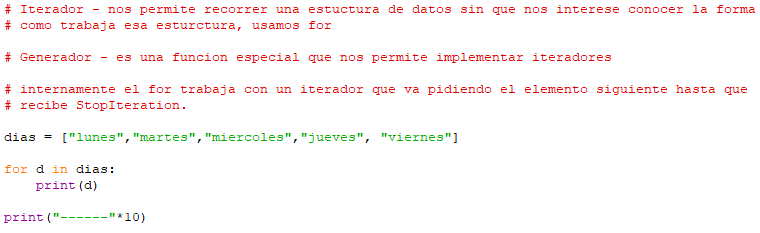
Otro ejemplo usando 2 veces el mismo decorado.

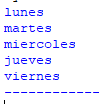
 

# Iterador - 37 - Python básico

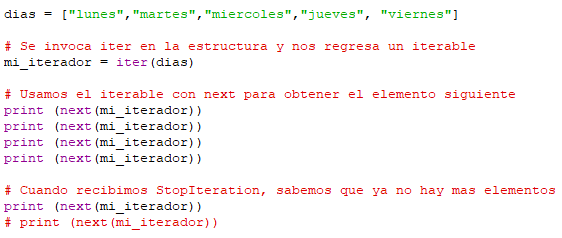
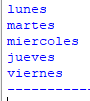
Sin entrar en detalle los iteradores son patrones de diseño y tienen métodos que los hacen funcionar, ejemplo método reset *que nos regresa al inicio de la estructura de datos*, el método has *next nos dice si queda algún elemento por leer.*

**For** en Python funciona como iterador

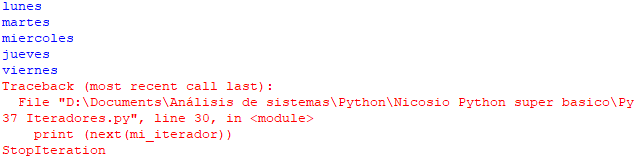




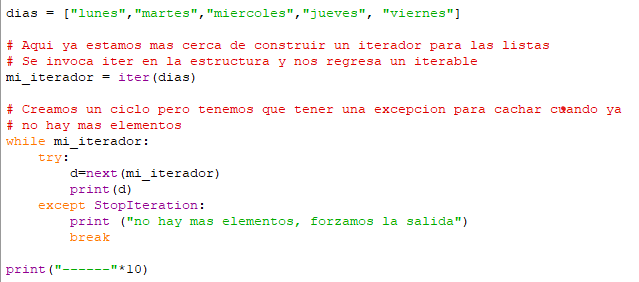
En el siguiente ejemplo para graficar iteramos a mano (lo hacemos nosotros)

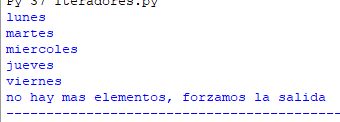
 

La variable mi\_iterador va a recibir mediante la invocación de la función **iter** en la estructura **lista [] días** nos va a regresar un objeto iterable, cada vez que use netx voy a leer el elemento actual y avanzo al siguiente, todo esto hasta llegar al final y recibir StopIteration que me rompe el programa.



En el próximo ejemplo estamos mas cerca de construir un iterador que funciones a través de un ciclo **While** y con la función **try** (que veremos en la lección 40 de manejo de excepciones) en el nivel de **except** capturo la excepción y llevo a cabo código.





Conceptualmente este código es similar al que existe dentro del bucle **for.**

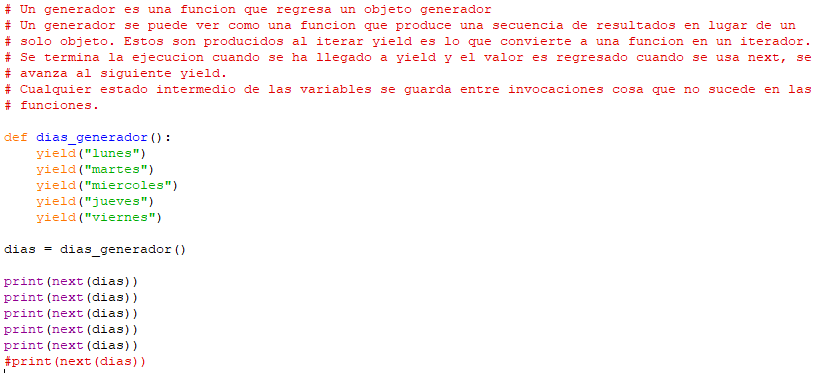
# Generadores - 38 - Python básico

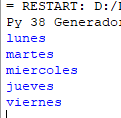
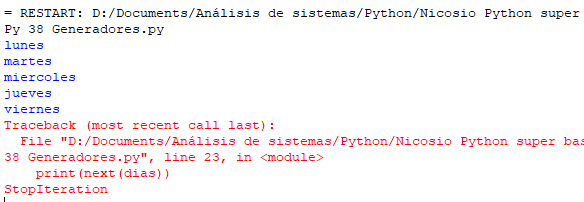
Un generador es una función que regresa un objeto generador

Un generador se puede ver como una función que produce una secuencia de resultados en lugar de un solo objeto. Estos son producidos al iterar yield es lo que convierte a una función en un iterador.

Se termina la ejecución cuando se ha llegado a yield y el valor es regresado cuando se usa next, se avanza al siguiente yield.

Cualquier estado intermedio de las variables se guarda entre invocaciones cosa que no sucede en las funciones.



Si retiro el comentario me da un erro con Stop Iteration.

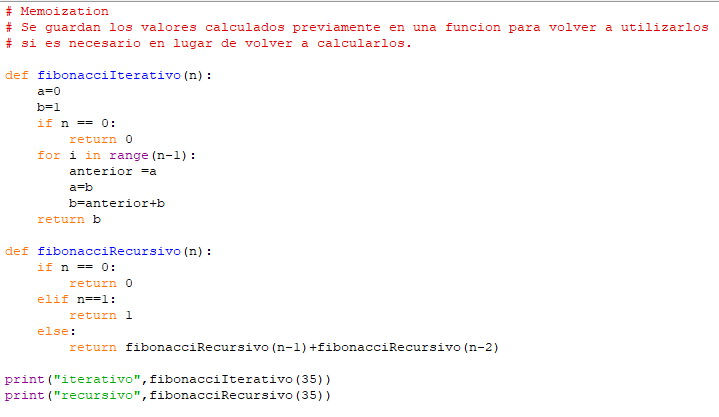
El break o corte del programa esta fuera de la función en este caso.

El caso siguiente no funciona en Python 3.X y no quise perder tiempo buscando documentación.

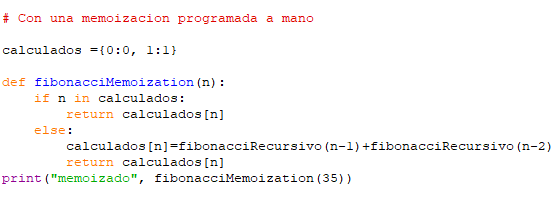
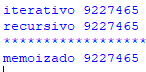
# Memoización - 38 - Python básico

Básicamente hace que se desperdicie menos memoria y tiempo de ejecución con cálculos.

**Tema difícil para mí en este momento.**

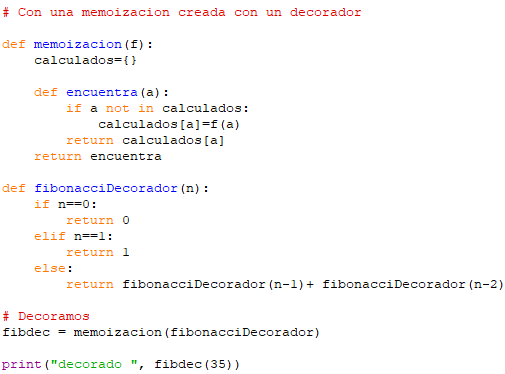


 Lo que se observa en el video es que el recursivo toma mucho mas tiempo.

En este caso la versión Memoization tardo menos que recursivo.

<https://www.youtube.com/watch?v=La6nd40Ab3o&list=PLM-p96nOrGcZmWE-320vKw-e6TJsle7rC&index=38>

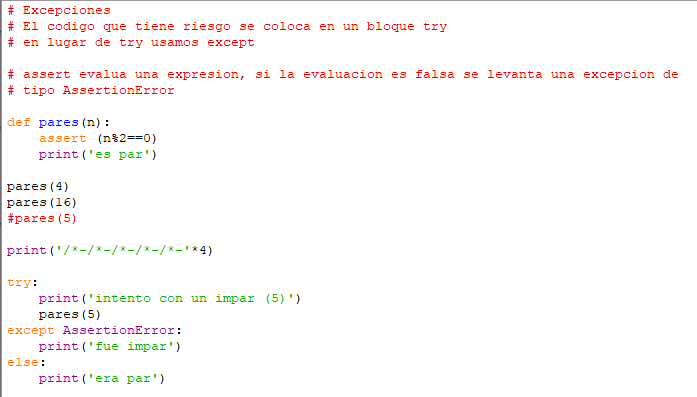
 

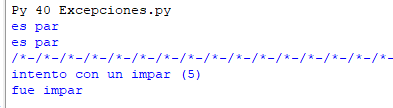
# Excepciones - 39 - Python básico

<https://controlautomaticoeducacion.com/python-desde-cero/manejo-de-errores-en-python/>

mejor explicado que en Nicosio

Assert va a evaluar una expresión



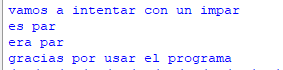


En **try** se coloca el código que puede generar el problema, al generarlo pasamos a **except** el cual necesita que le indiquemos el error o excepción que nos interesa capturar, debido a que en la función estamos usando **assert** su excepción es la que nos interesa capturar. Si el try no arroja error pasa al **else** sino se ejecuta la excepción

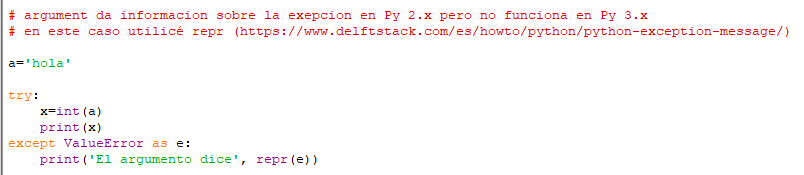
Si quisiera capturar cualquier tipo de excepción o todas utilizo solo el **except**.

Con el uso de **finally** vamos a ejecutar un bloque de código **haya o no** ocurrido la excepción





Al igual que en otros lenguajes la información puede tener argumentos, con la palabra reservada **Argument** podemos visualizar el mismo en Py 2.X pero en 3.X lo solucione con **repr.** De esta pagina <https://www.delftstack.com/es/howto/python/python-exception-message/>





múltiples excepciones y levantar una excepción usando **rise**

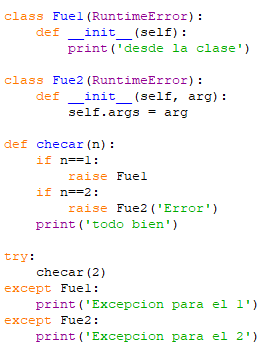
Las clases son donde creo el error (no lo explico bien todavía)

Con la función **checar** establezco las condiciones para los parámetros 1 y 2, si ingreso 1 se levanta a través de **rise** la clase Fue1 y lo mismo con 2.

Luego en el bloque **try** le paso la función creada con algún parámetro: 1, 2 o cualquier otro

Si entra en **checar** con el param 1 levanta el primer **rise** e imprime ‘excepción para el 1’ lo mismo con 2 aunque ese tiene también un argumento creado que no explico.

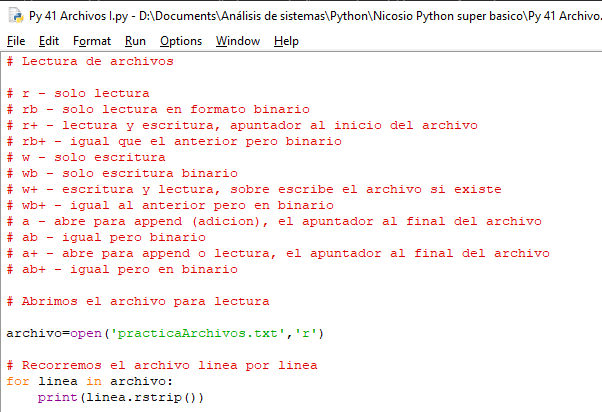
Por último, si se ingresa cualquier otro valor imprime ‘todo bien’

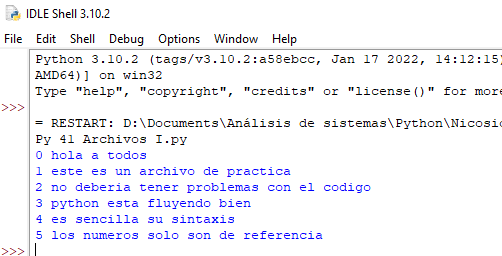


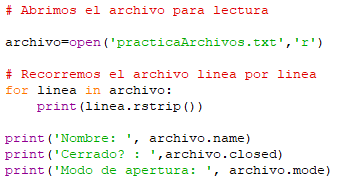
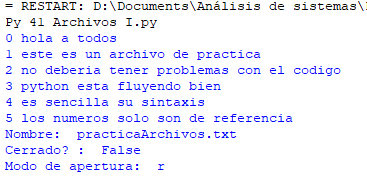
  

# Archivos I - 41 - Python básico

Cree un archivo de texto en la misma carpeta en la que ejecuto el programa .Py

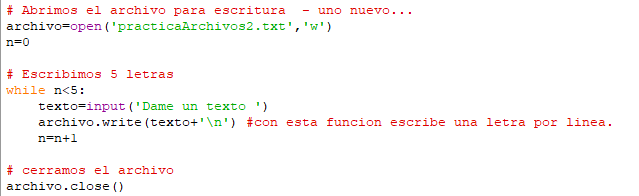


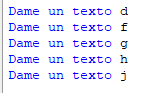
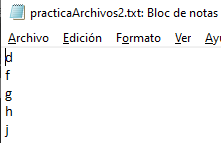


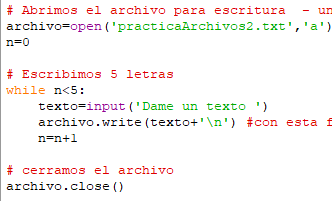
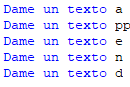
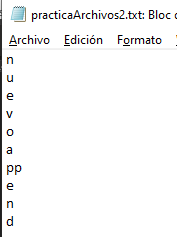


The raw\_input() function reads a line from input (i.e. the user) and returns a string by stripping a trailing newline. This page shows some common and useful raw\_input() examples for new users. Please note that raw\_input() was renamed to **input()** in Python version 3.



  Funcionó perfecto!!!

 **Si el archivo no existe crea uno con el nombre y lo escribe, en caso de Si existir escribe sobre ese.**

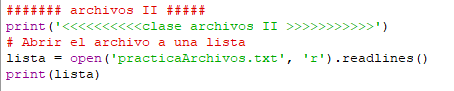
  

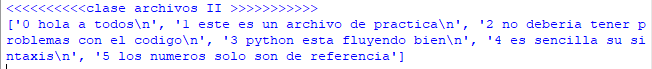
Cambiando la w por a escribe al final del archivo agregando líneas.

**Importante** si el archivo esta **abierto** no muestra los cambios hasta que lo cierre y vuelva a abrir.

# Archivos II - 42 - Python básico

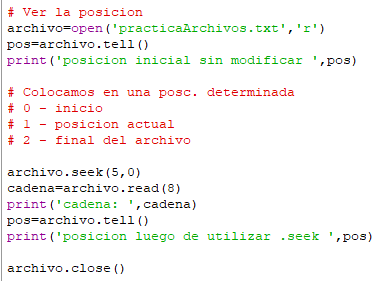
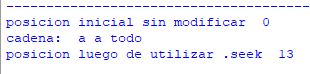
En ocasiones es conveniente colocar las líneas de un archivo de texto dentro de una lista, **serialización**,





A cada línea del txt lo convierte en un elemento.

Ver posición del puntero en el archivo y utilizar .seek para buscar un punto especifico

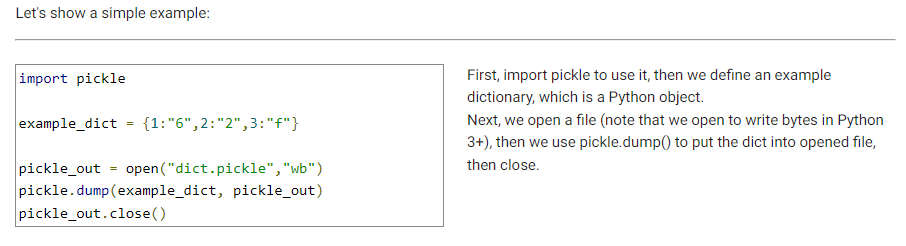
 

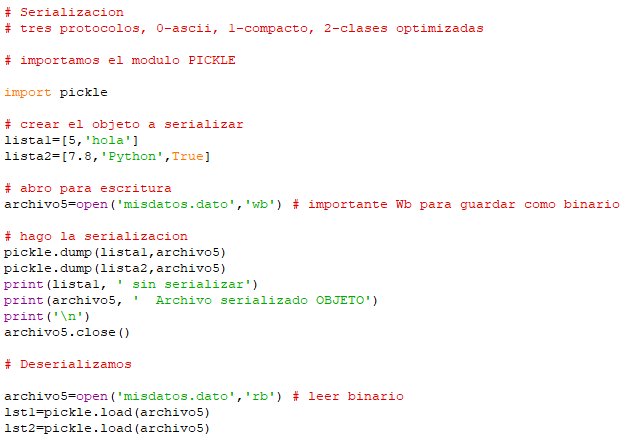
 (de la pos **0** cuento **5** posiciones sería la última ‘a’ de ‘hola’, y de ahí leo **8** pos más hasta la ultima ‘o’ de ‘todos’.)

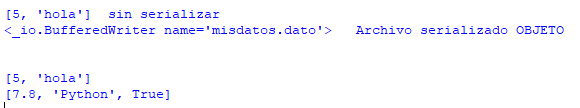
Con la función **seek** le paso el primer parámetro que posición quero leer y el segundo es a partir de donde comienzo a hacerlo.

Luego en este caso en cadena le pido que lea 8 bits por último con **tell()** me muestra la nueva posición del puntero.

**Serialización importante en py 3.X se usa wb**

****





# Clases - 43 - Python básico

The \_\_init\_\_ function is called when an object is created. used to initialize the object.

The self parameter is used to refer to the object itself.

used to access the attributes and methods of the class.

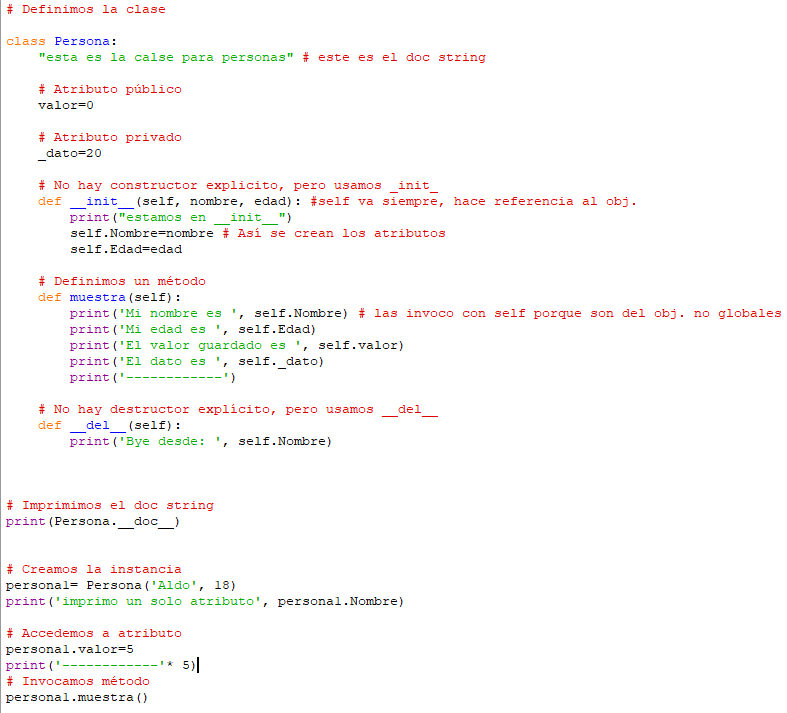
The \_\_init\_\_ function is called automatically when the object is created

:param nombre: This is the first parameter of the \_\_init\_\_ method. It is a string that

represents the name of the person

:param edad: This is the parameter that is passed to the \_\_init\_\_ method

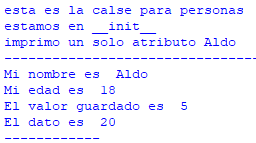
The \_\_del\_\_() method is called when the instance is about to be destroyed

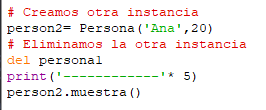
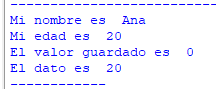


# Invocamos método

# Calling the method `muestra` on the object `persona1`

persona1.muestra()



# Clases II - 44 - Python básico

La **variable de clase** es compartida por todas las instancias de la clase, si una instancia hace un cambio a esa variable todas las demás instancias van a ver reflejado ese cambio.



Idéntico a la clase anterior, se agregó una variable de clase. Esta se declara y se invoca distinto “**;**” “**Persona.cantidad**”

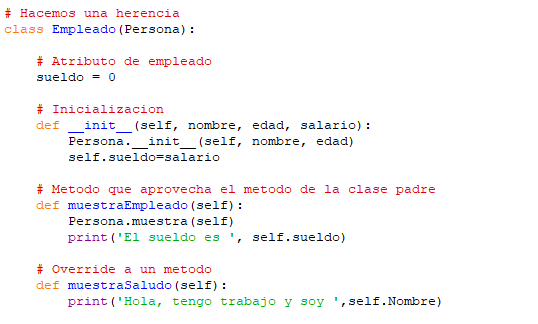
El método ponCantidad va a modificar a la variante de clase.

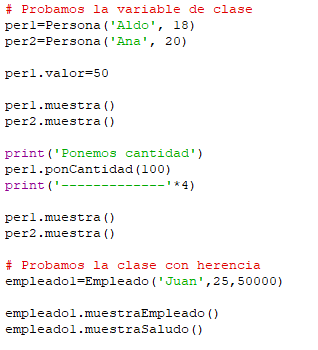
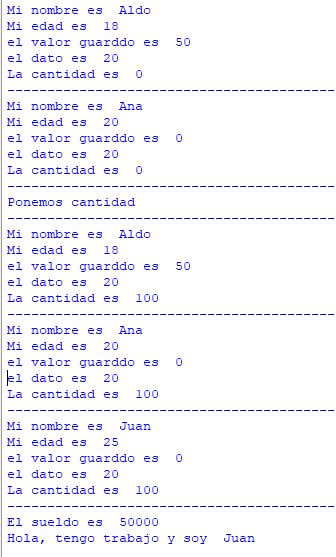
En la clase empleado utilizamos la **HERENCIA**, de la clase persona más los atributos propios, en el inicializador le paso los paramatros padre mas los de hija, luego cuando los inicializo lo hago:

Persona.\_\_init\_\_(self, nombre, edad)”padre”

Self.sueldo=salario “hija”

En el override simplemente sobrescribo un método de la clase padre

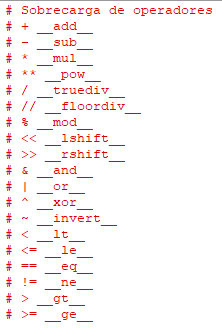


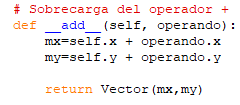
 

# Sobrecarga de operadores - 45 - Python básico

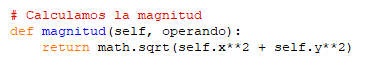
La sobrecarga de operadores es una técnica muy poderosa de la PO nos permite utilizar los operadores con nuestras propias clases, en Python cada operador tiene un método y eso lo hace más fácil que en otros lenguajes.

Al sobrecargar hago mucho mas corta la sintaxis del lado del cliente.

**

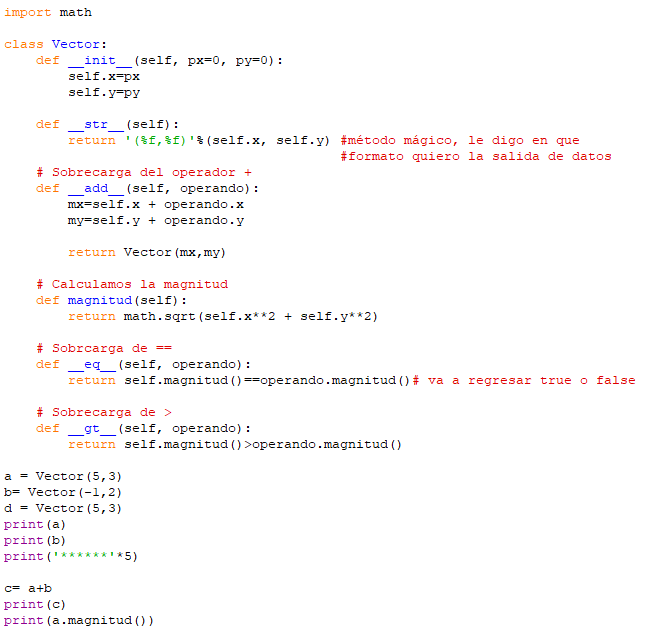
En este caso creo una regla para sumar vectores y quiero como resultado un vector. 

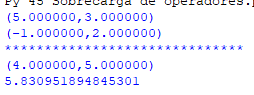
El profesor decide comparar los vectores a través de su magnitud ( <https://www.fhybea.com/magnitud-vector.html> )



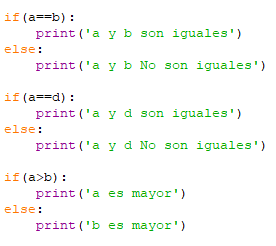
Al momento de imprimir los vectores lo hace con la sobrecarga de **str** y por eso el formato es el deseado.

Cuando hago la suma en c = a+b entra en la sobrecarga de **+** siendo **a self** y **b operando.**



 …

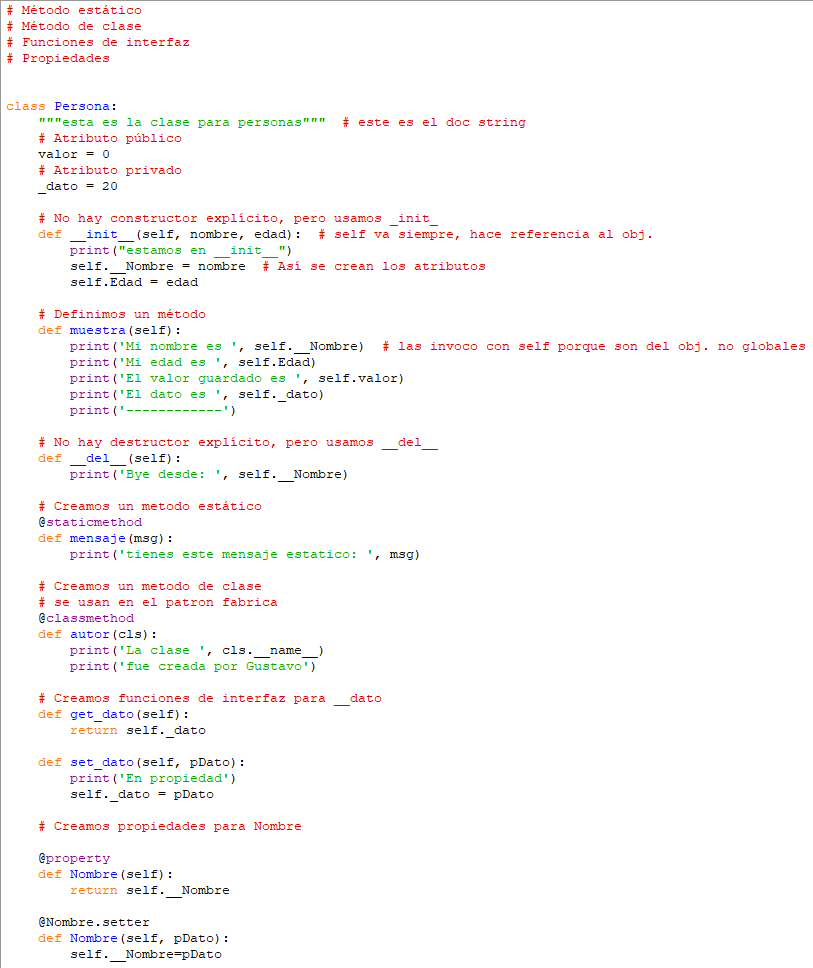
Ahora comparo con la sobrecarga de estos operadores:

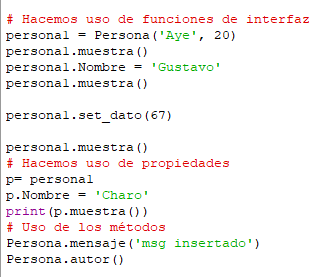
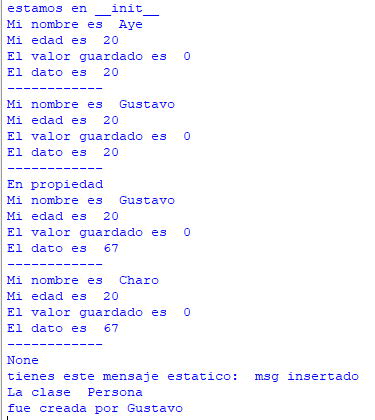
 

# Clases III - 46 - Python básico

En este tema tuve problemas con el decorador @property luego de buscar en varias fuentes entendí que el atributo a decorar debía ser privado y con eso funcionó.

Esta clase es bastante larga en contenido o al menos me llevo bastante entenderla asi que la voy a pegar completa y luego desglosar.



# Atributos dinámicos - 47 - Python básico

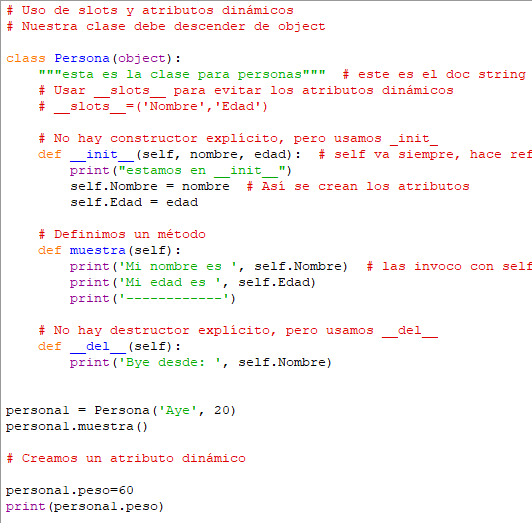
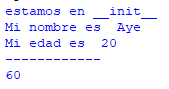
<https://docs.hektorprofe.net/python/programacion-orientada-a-objetos/atributos-y-metodos/>

Los atributos dinámicos tienen por su facilidad de ser creados algo muy bueno, pero también algo malo y es que por un error de tipeo podemos crear uno sin desearlo y no nos va a dar un error.

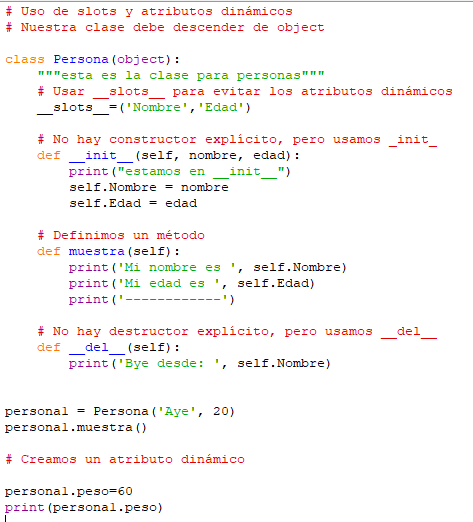
**Podemos elegir usar o no atributos dinámicos**

Cuando definimos una clase dentro de esta existe una estructura de datos que se llama **SLOTS** dentro de esa estructura se van colocando los diferentes atributos que le estamos dando a la clase y sus valores correspondientes. Hay un mecanismo en el que podemos indicar cuales son los **slots** que queremos utilizar de manera que si por error creamos un atributo dinámico nos marcara un error.

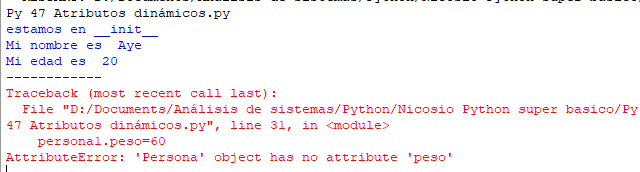
Para poder utilizar los slots la clase debe descender de (object) y los definimos de esta manera: \_\_slots\_\_=('Nombre','Edad')

Ejemplo con atributos dinámicos antes de poner en uso los Slots



Ejemplo con uso de Slots bloquean a los atributos dinámicos y arroja error



# Metaclases - 48 - Python básico

Tema avanzado…

En Python todo es un *objeto*, si tenemos un *objeto* cualquiera este es una instancia de alguna *clase,* pero esta a su vez es una instancia de una *metaclase*.

Si modifico a la metaclase toda su herencia es modificada: metaclase >clase>instancia.

En el ejemplo el método duplicador esta fuera de las clases, pero lo vamos a poder aplicar a cualquier clase que posea un atributo llamado *a* o *b.*

Este ultimo tema funciona distinto en Python 3.X no lo resolví aun, pero debo pasar a otro tema para avanzar, no voy a guardar la captura, pero si el código en la carpeta correspondiente.