COMPREHENSIONS, FUNCIONES Y MANEJO DE ERRORES

TABLA DE CONTENIDO

[SETS 4](#_Toc129948012)

[MODIFICAR UN CONJUNTO 6](#_Toc129948013)

[AGREGAR 6](#_Toc129948014)

[MODIFICAR 6](#_Toc129948015)

[REMOVER 7](#_Toc129948016)

[OPERACIONES CON CONJUNTOS 8](#_Toc129948017)

[Unión 8](#_Toc129948018)

[Intersección 9](#_Toc129948019)

[diferencia 9](#_Toc129948020)

[Diferencia simétrica 10](#_Toc129948021)

[RETO: UNIÓN DE PAÍSES Y ELIMINAR LOS REPETIDOS 11](#_Toc129948022)

[LIST COMPREHENSION 11](#_Toc129948023)

[DICTIONARY COMPREHENSION 14](#_Toc129948024)

[DICTIONARY COMPREHENSION: CONDITION 16](#_Toc129948025)

[LIST VS TUPLES VS SETS 17](#_Toc129948026)

[FUNCIONES 17](#_Toc129948027)

[FUNCIONES RETURN 19](#_Toc129948028)

[EL SCOPE 20](#_Toc129948029)

[REFACTOR GAME 23](#_Toc129948030)

[FUNCIONES ANONIMAS LAMBDA 25](#_Toc129948031)

[HIGHER ORDER FUNCTION: UNA FUNCION DENTRO DE OTRA FUNCION 26](#_Toc129948032)

[MAP 27](#_Toc129948033)

[MAP CON DICCIONARIOS 29](#_Toc129948034)

[RETO MAP CON INMUTABILIDAD 31](#_Toc129948035)

[FILTER 32](#_Toc129948036)

[REDUCE 33](#_Toc129948037)

[MODULOS 34](#_Toc129948038)

[functools 34](#_Toc129948039)

[sys 34](#_Toc129948040)

[Re 35](#_Toc129948041)

[Time 35](#_Toc129948042)

[Collections 36](#_Toc129948043)

[MIS PROPIOS MODULOS 37](#_Toc129948044)

[MODULOS COMO SCRIPTS: \_\_NAME\_\_Y\_\_MAIN\_\_ 39](#_Toc129948045)

[PAQUETES 41](#_Toc129948046)

[ITERABLES 45](#_Toc129948047)

[ERRORES EN PYTHON 47](#_Toc129948048)

[MANEJO DE EXCEPCIONES 49](#_Toc129948049)

[LEER UN ARCHIVO DE TEXTO 52](#_Toc129948050)

[ESCRIBIR UN ARCHIVO 54](#_Toc129948051)

[LEER UN CSV 55](#_Toc129948052)

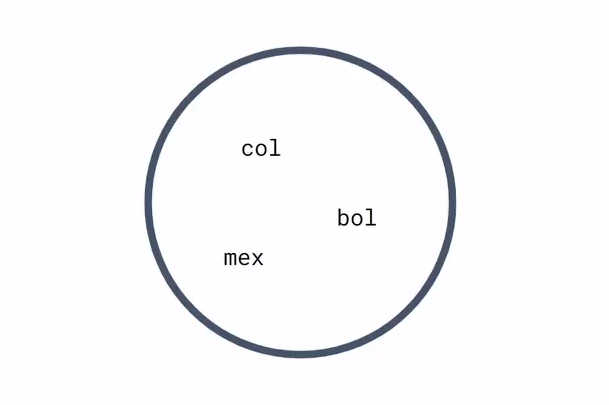
[CREANDO UNA GRAFICA 57](#_Toc129948053)

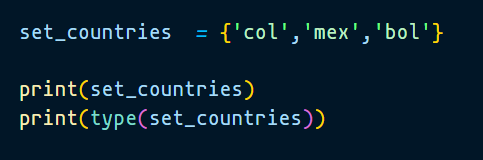
# 

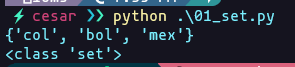
# SETS

(Conjuntos)

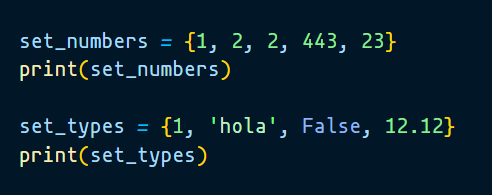
Agrupa elementos que tienen algo en común. No tienen un orden especifico, tampoco se permiten duplicados.





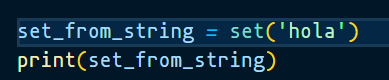


Solo muestra un elemento de elementos repetidos



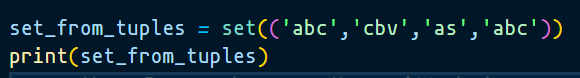


Se puede definir conjuntos a partir de otras estructuras de datos. Si los caracteres se repitieran entonces solo tendría uno en cuenta.





Aquí transformamos una tupla en un conjunto. Luego lo imprimes, note que no repite abc



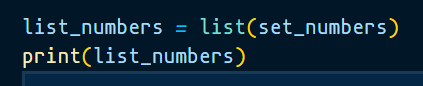


Trabajando con una lista, podemos transformarlo en conjunto. Otra vez, no repetirá los elementos. Devolverá un diccionario, luego también se puede transformar en lista





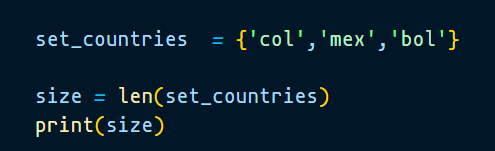
Se puede transformar un conjunto e una lista





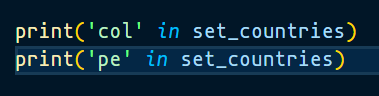
# MODIFICAR UN CONJUNTO

Podemos ver que tenemos el conjunto set\_countries y tiene un tamaño de 3





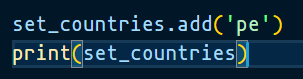
Podemos preguntar si Colombia o Perú están dentro del conjunto set\_countries





## AGREGAR

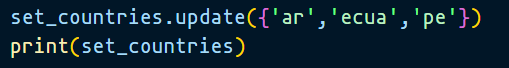
Se puede agregar ‘pe’ a set\_countries. Si se intentara ingresar “pe” por segunda vez solo se mostraría una





## MODIFICAR

Podemos agregar un conjunto a otro conjunto y el resultado será la unión de ambos en un solo conjunto sin repetir elementos.





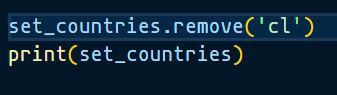
## REMOVER

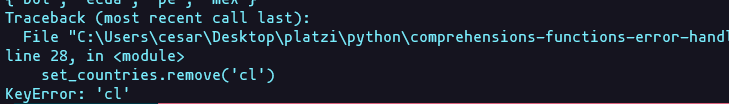
Removemos Colombia y argentinay como resultado nos quedan menos paises



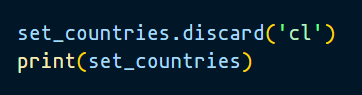


Si quisiera remover un elemento que no existe, mostrara error



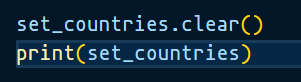


Se puede usar para este fin el método discard. Que no mandara error simplemente ignorara si es que no existe el elemento a la hora de remover



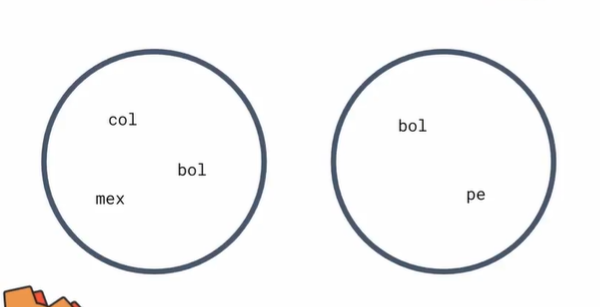


Puedes limpiar un conjunto con clear()



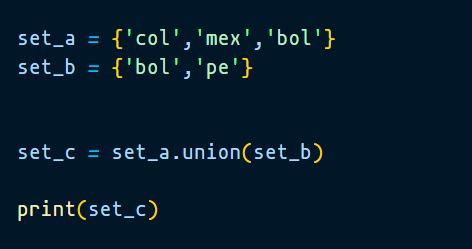


# OPERACIONES CON CONJUNTOS

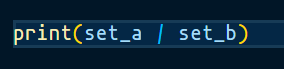


## Unión



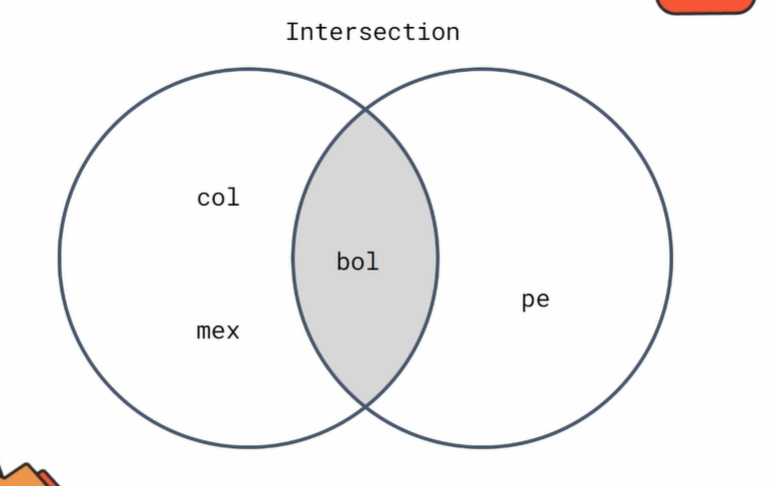


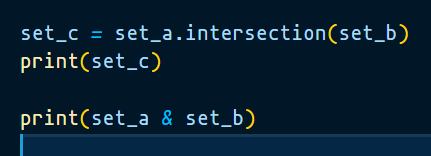






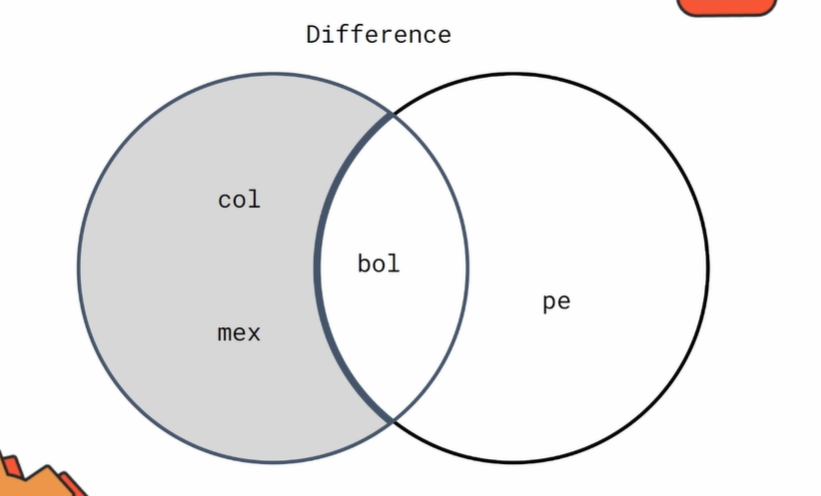
## Intersección

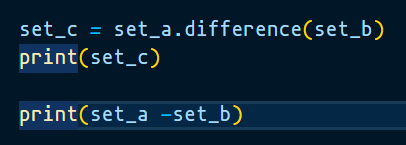






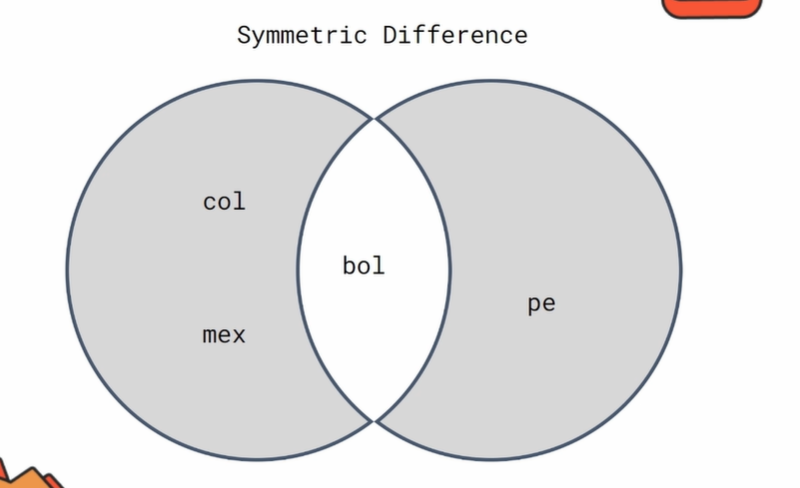
## diferencia

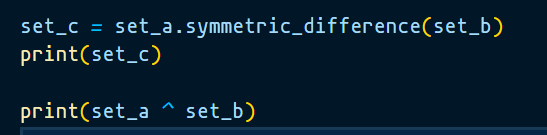






## Diferencia simétrica



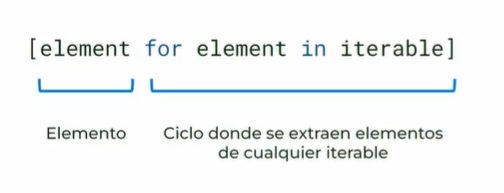


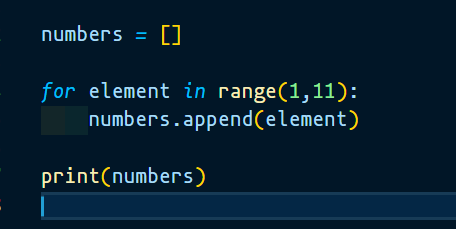


# RETO: UNIÓN DE PAÍSES Y ELIMINAR LOS REPETIDOS

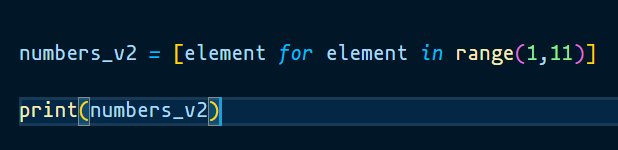


# LIST COMPREHENSION







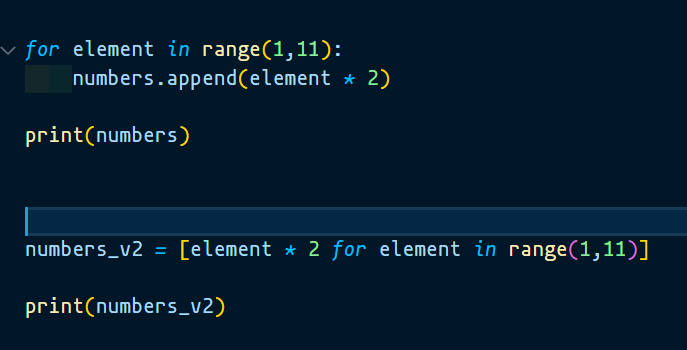




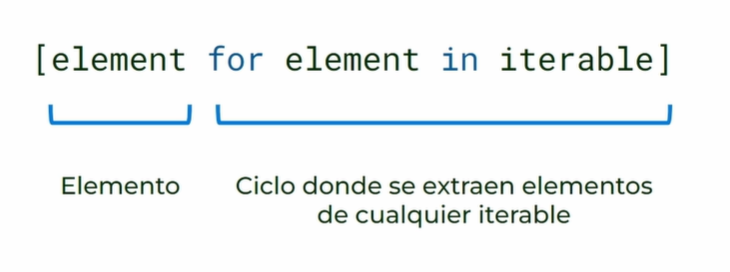
En el ejemplo queremos tomar elementos en un rango del 1 al 11 (1,2,3 … 10)

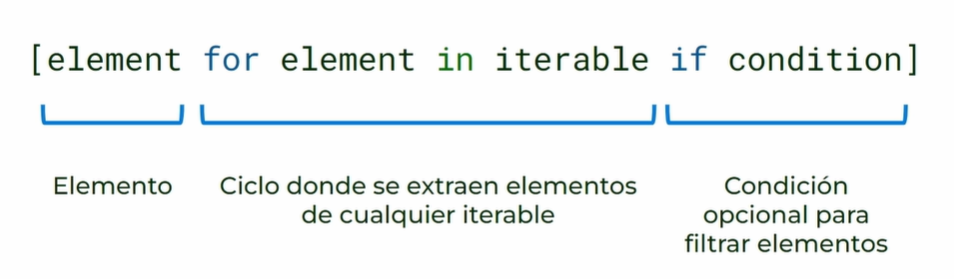
Luego multiplicar cada resultado de cada elemento y multiplicarlo x 2 y se agrega al arreglo numbers.

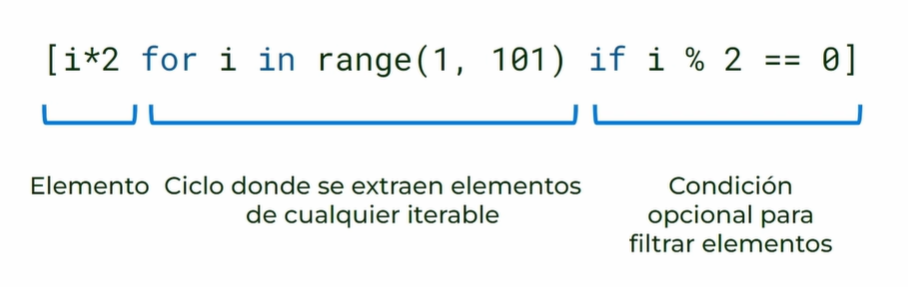
Se puede hacer de manera resumida multiplicando los elementos x 2 venidos de el rango entre 1 y 11 (1 al 10)







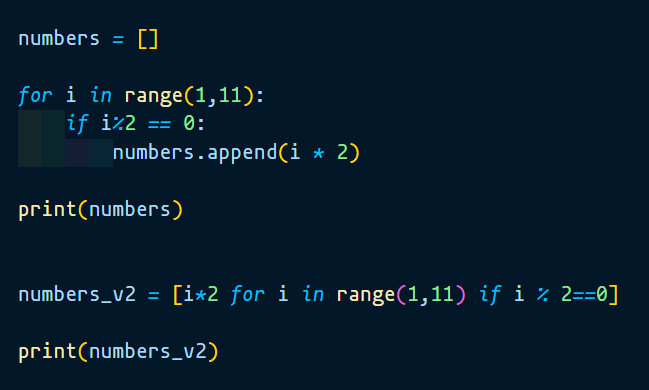




Existen 2 formas de escribir lo Sgte:

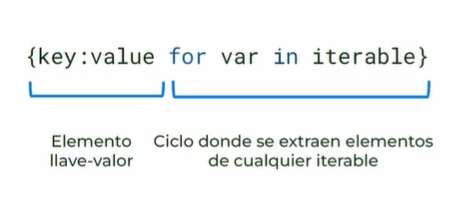
Quiero repasar un rango de valores desde el 1 al 10 y los valores pares multiplicarlos por 2y agregarlos al arreglo numbers.

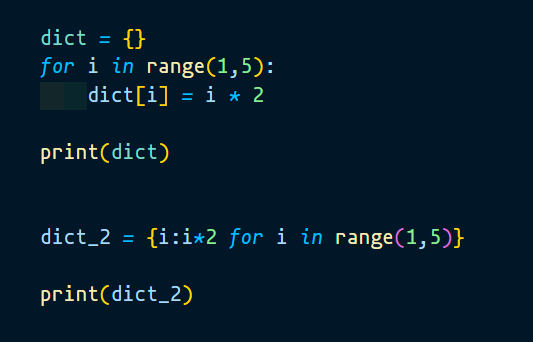
numbers\_v2 contiene un arreglo que internamente toma un rango de números entre 1 al 10 el cual se multiplica por 2 si es que el resultado de su residuo es 0



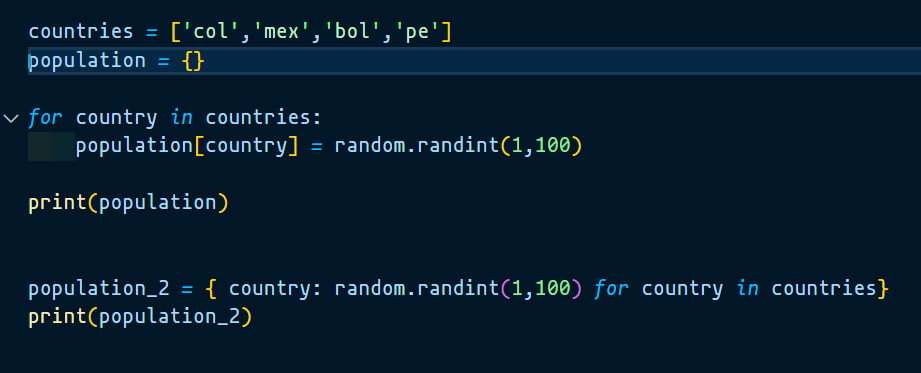


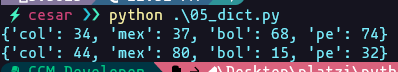
# DICTIONARY COMPREHENSION









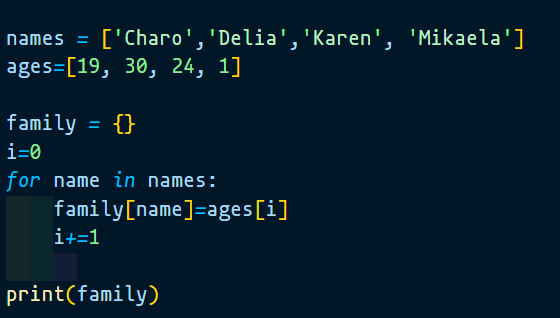


Nos piden crear un diccionario donde aparezca un nombre relacionado a su respectiva edad (key : value).

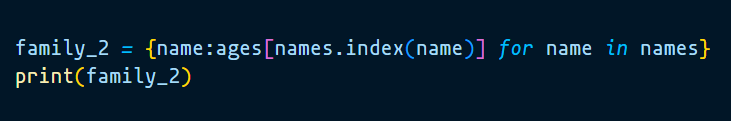
De forma tradicional podemos crear un diccionario family y una variable i que valga 0

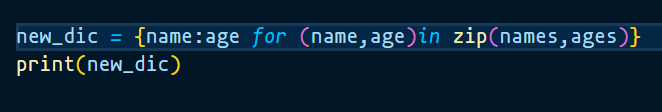
Que recorra cada nombre a travez de un for

En cada recorrido que cree la clave valor usando el nombre y el valor de age y a final sumarle 1 a I

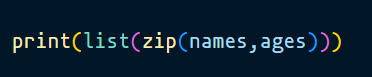


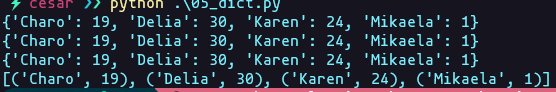
Esta forma es mas corta, esta envuelta en corchetes



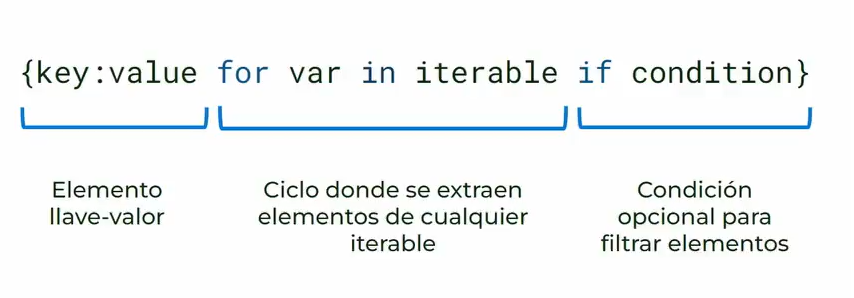


Zip permite unir listas. Para poder verla en una lista como tal se debe aplicar list(). El resultado es un a lista con tuplas internas con la estructura name, age

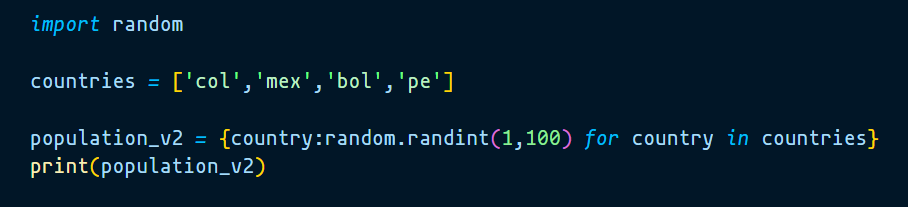


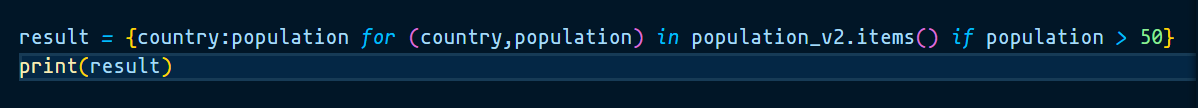


# DICTIONARY COMPREHENSION: CONDITION

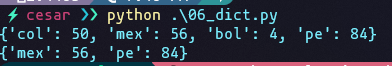


Creamos un diccionario que contenga data country population con un valor random y como country se toma desde la lista countries

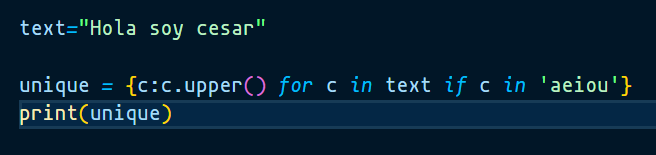




Primero se genera un diccionario llamado population\_v2 y sobre ese diccionario se genera otro que contenga datos de acuerdo a la condición population > 50



Tomamos un texto, tomamos cada letra y si cumple que es a,e,i,o,u entonces es considerada.



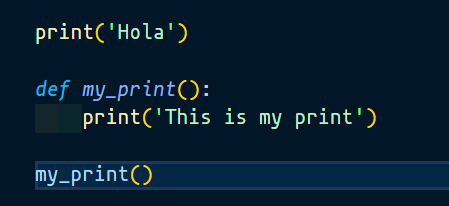


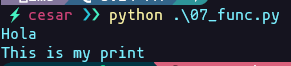
# LIST VS TUPLES VS SETS

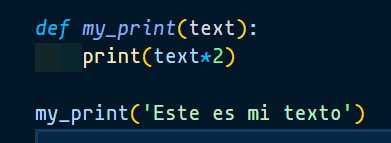


# FUNCIONES

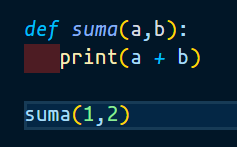
Dentro tiene definido el comportamiento.











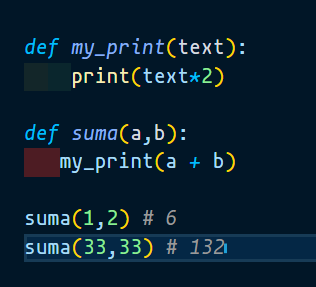
Se puede llamar una función dentro de otra función

En el caso de la suma 1 y 2 llegara a la función suma y sumara ambos números dando como resultado 3

El 3 pasara como argumento para la función my\_print

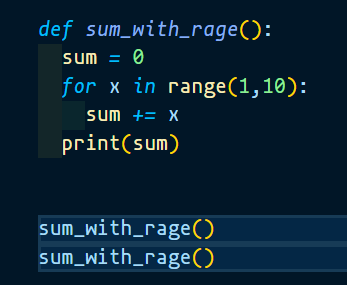
Ahora el parámetro text vale 3 y se multiplicara por 2

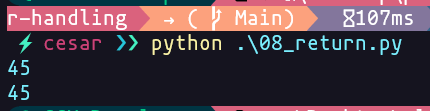
El resultado final es 6



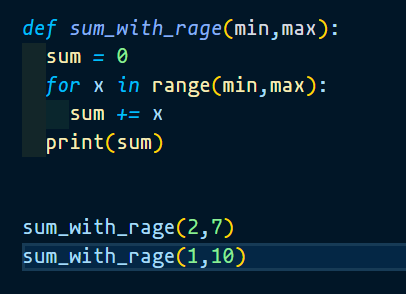
# FUNCIONES RETURN

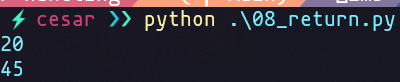
Se puede llamar todas la veces que necesitemos





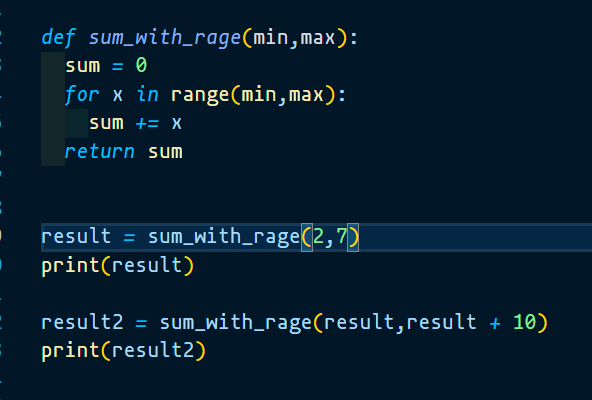
También se puede agregar parámetros de esta forma podemos reutilizarlo de manera mas personalizada.

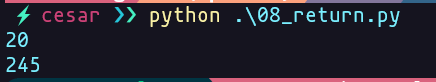




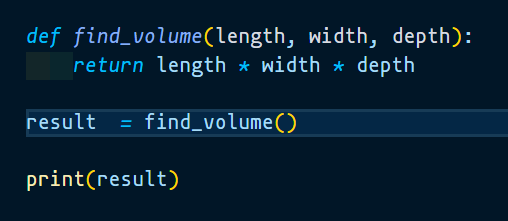
Podemos retornar el resultado de una función utilizando return.

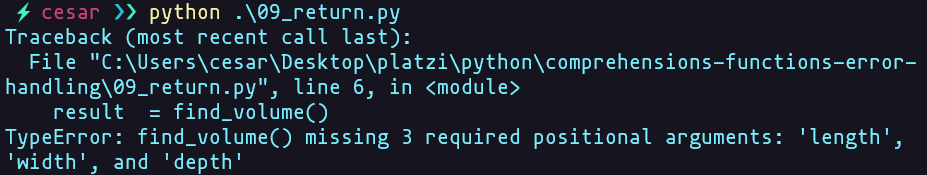
Además, podemos reutilizar ese valor para poder procesar otros datos



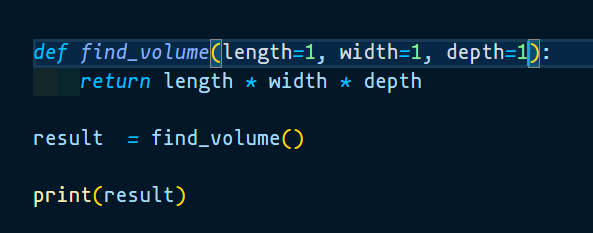


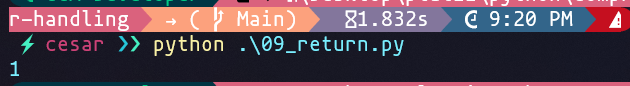
# EL SCOPE





Se puede definir un valor por defecto





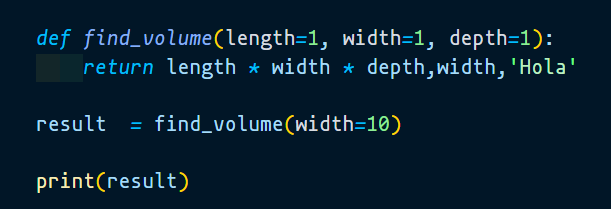
También se puede pasar un valor de un parámetro especifico





Se puede retornar mutiples valores, en el ejemplo se agrego width y el string Hola

Lo que retornara una tupla



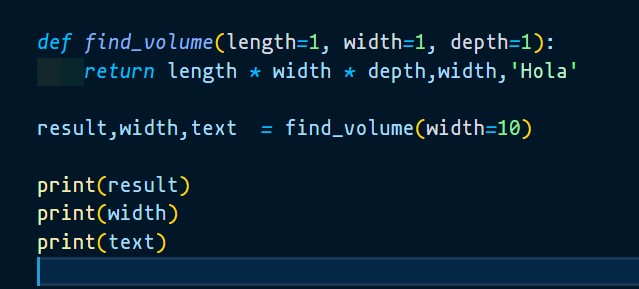


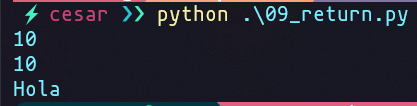
Y de esta tuple se puede tomar algún valor indicando la posicion, en el ejemplo es cero



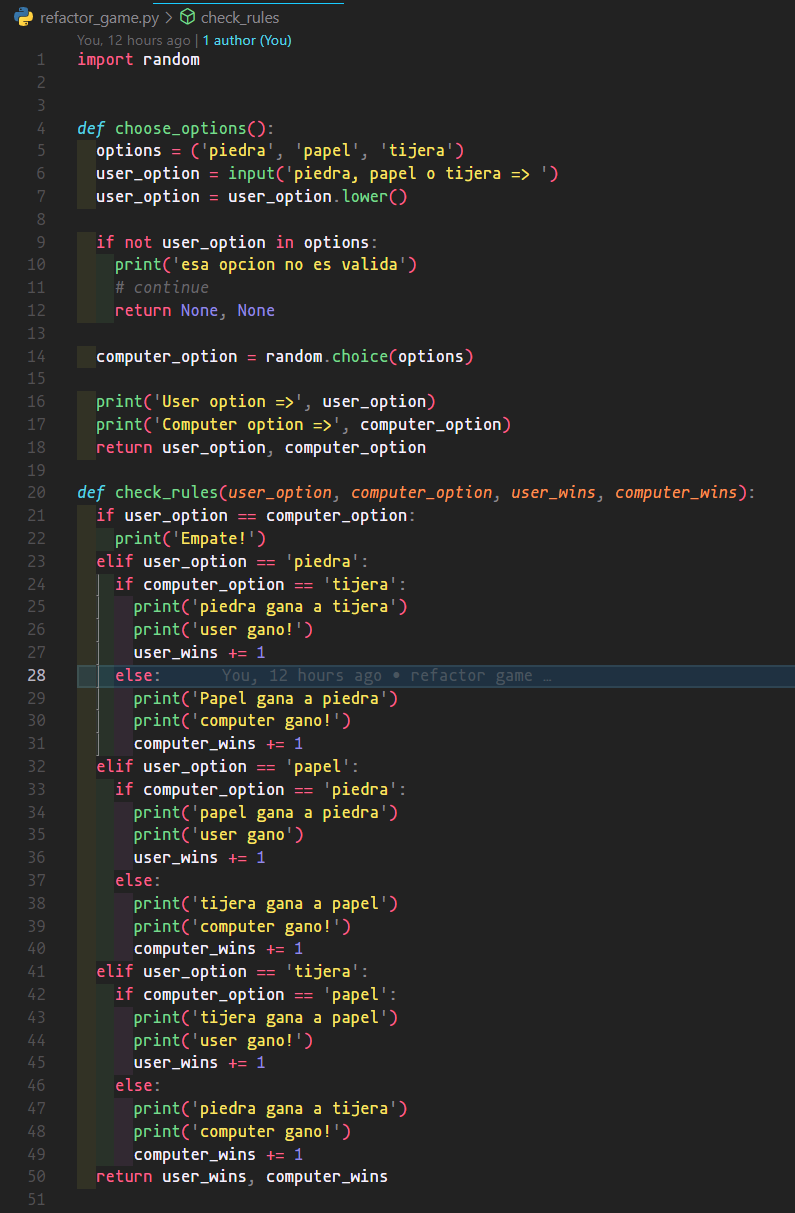


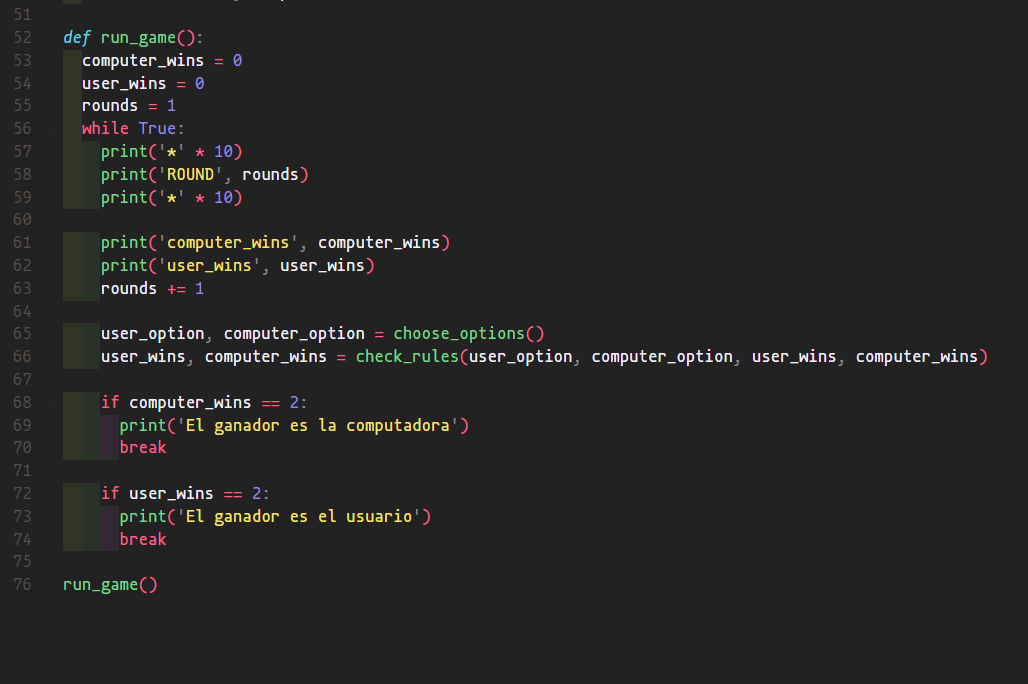
También esos múltiples valores puedes mandarlos a imprimir





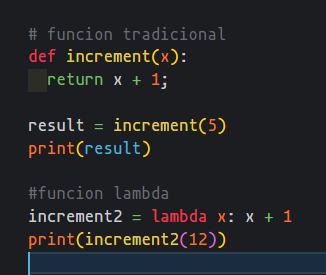
# REFACTOR GAME

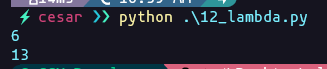




# FUNCIONES ANONIMAS LAMBDA

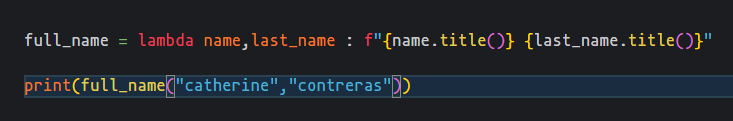
Las funciones lambda es una manera corta de escribir una función.   
Puede recibir parámetros la funcionalidad esta separada por “:” .También permite ser almacenada dentro de una variable





En este ejemplo lambda recibe dos parámetros separados por coma luego se muestra la concatenación de los parámetros usando formato.

Luego el lambda se almacena en la variable full\_name y se manda a imprimir





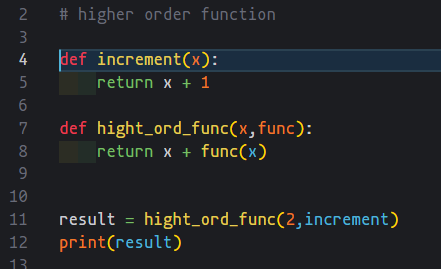
# HIGHER ORDER FUNCTION: UNA FUNCION DENTRO DE OTRA FUNCION

Creamos una función llamada increment esta recibe un parámetro x

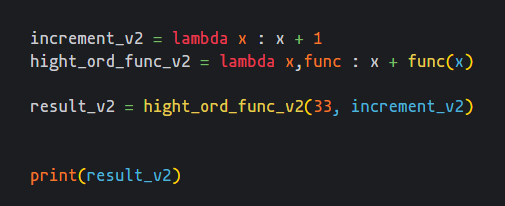
Luego tenemos la función principal hight\_ord\_func que recibe como parámetros x y una función

Esto devolverá la suma de x con el resultado de la función increment que recibirá el parámetro x de la función hight\_ord como parámetro

El resultado se imprimirá.





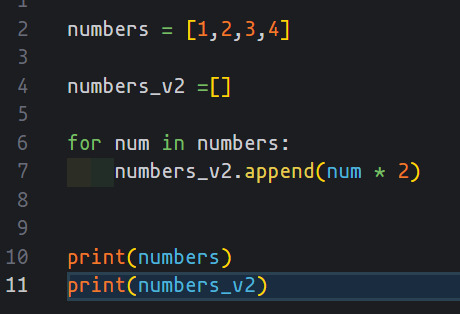




# MAP

Ejecuta una función específica para cada elemento en un iterable el elemento se envía a la funcio0n como un parámetro

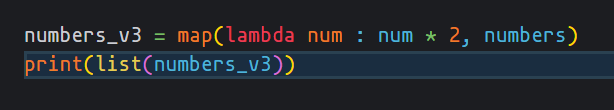
Va devolver 4 elementos de una lista, pero transformado

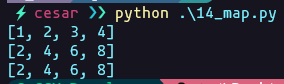




También podemos emplear map de una manera corta, comprimida.

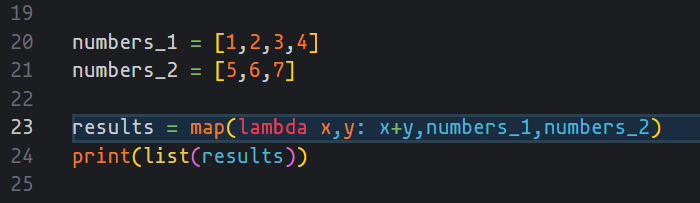
Aplicamos map() y dentro hacemos una función lambda donde tomara el parámetro iterable en este caso lo multiplicara por 2 y luego indicara de que lista esta iterando.





Creando con lambda se puede hacer todo en una sola línea

En este ejemplo se pide sumar los elementos de dos listas. Como la lista numbers\_2 tiene solo 3 elemento y no 4 como la primera lista no hay con quien lo sume por lo tanto solo se tendrá el resultado de 3 sumas.





# MAP CON DICCIONARIOS

Tenemos una lista llamada ítems que contiene 3 diccionarios.

Queremos mostrar solo los precios de cada producto utilizando la función lambda





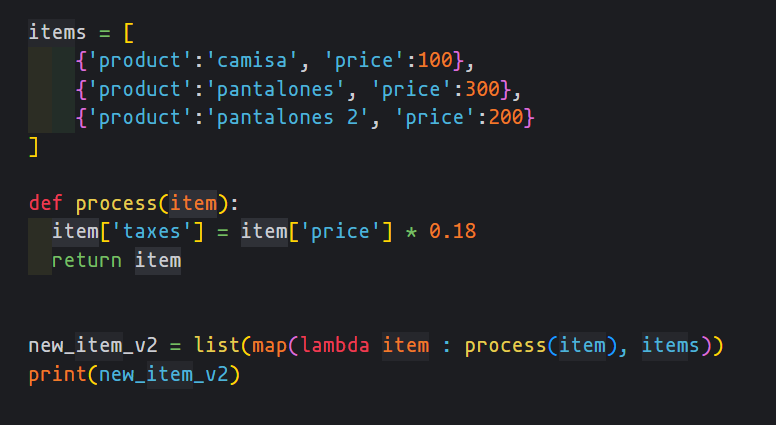
En este ejemplo queremos agregar una nueva propiedad a cada diccionario, esta propiedad será llamada taxes que representan el 19% del precio del producto.

Aquí se tiene que tener en cuenta que trabajar con una función lambda implica utilizar una línea. En este caso tenemos que crear una nueva propiedad(taxes) con un valor venido de la operación desde otro valor(price)

Entonces utilizamos una función que procese el item, saque el valor de Price, lo múltiple por 0.18 y lo almacene en una nueva propiedad llamada taxes.

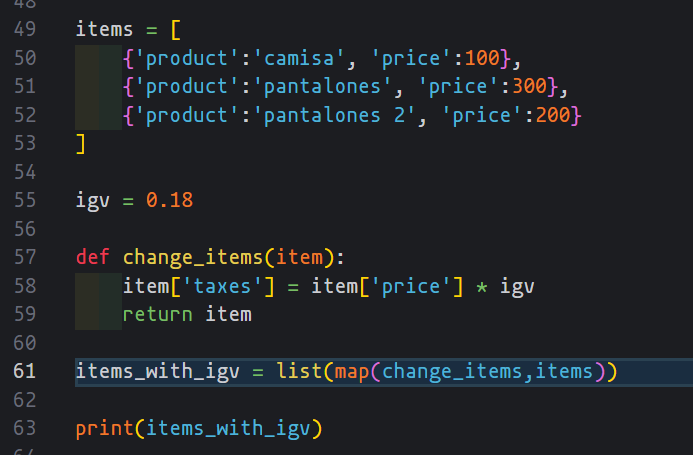
Finalmente, esta función debe retornar el item completo.

En la función lambda solo le pasamos el item desde ítems.



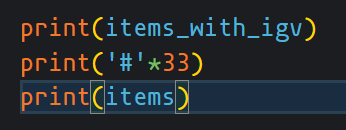
Otra manera de desarrollar el ejercicio anterior es aplicandola funcion.

Nota que a la funcion change\_items no le pasas argumento dentro de lista map

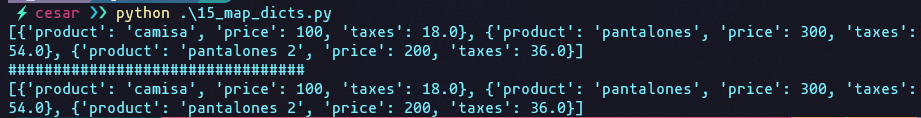




Si tu imprimes ítems, supuestamente no debería haber mutando la lista ítems, pero no es así



Ítems mantiene los cambios hecho con la función change\_items



# RETO MAP CON INMUTABILIDAD

Referencia en memoria

El diccionario se asigna como referencia de memoria

Los 2 comparten la misma referencia de memoria

Por lo tanto si se hace un cambio en el nuevo item también tendrá un impacto en el antiguo item por compartir un espacio de memoria en común.

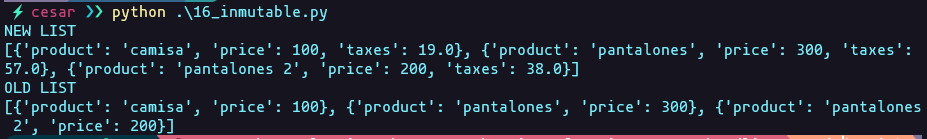
En el ejemplo donde se hace el cambio al diccionario es en esta línea:



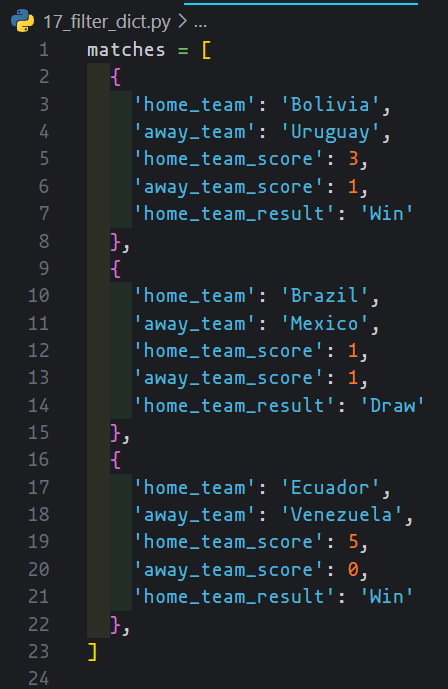
Para solucionar esto debemos crear una copia del item y asignarle una nueva variable

La copia del item se crea utilizando el método copy()



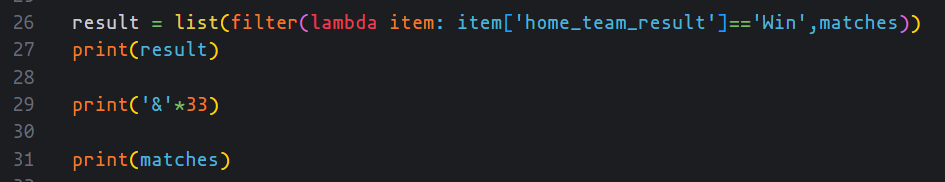


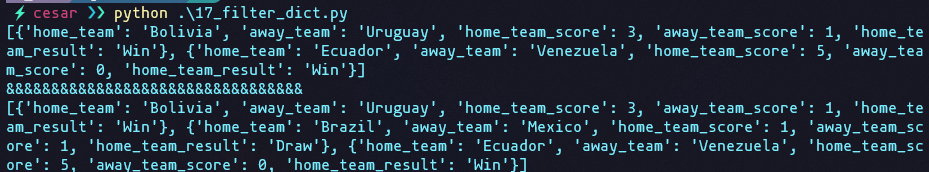
# FILTER

­

Se emplea filter que devuelve las coincidencias que hagan match con la condición

En el resultado puedes ver que la lista matches original no se ha modificado.





# REDUCE

Reduce algo a un solo valor. Tomar una lista y sacar una conclusión de esa lista.

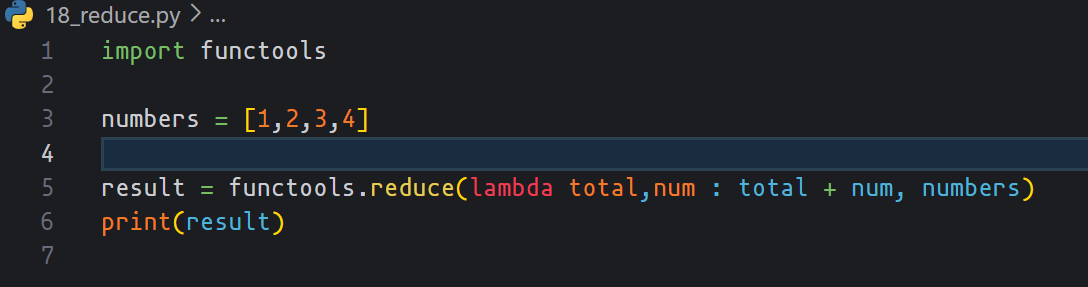
Por ejemplo, de una lista de 1,2,3,4 y queremos saber la suma total seria 10.

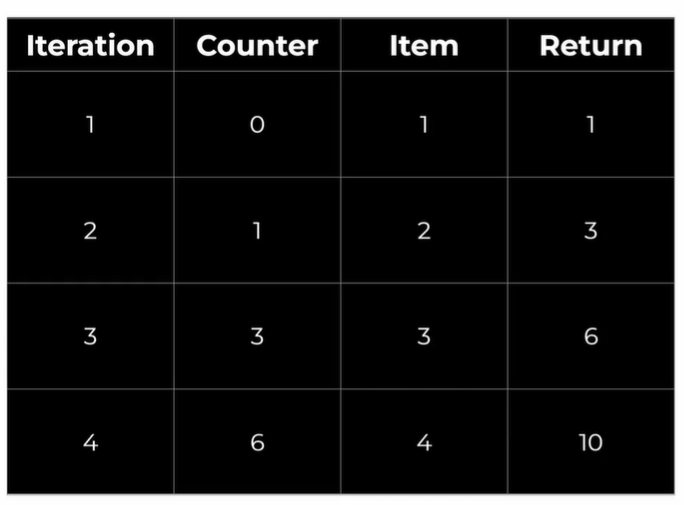
También podemos reducir esa lista preguntando cual es el numero mas grande en ese caso seria 4.

Siempre se tiene que importar la librería functools.

De functools se puede acceder al método reduce.

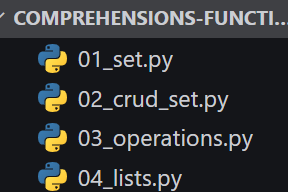
Dentro puede venir la función lambda.





# MODULOS

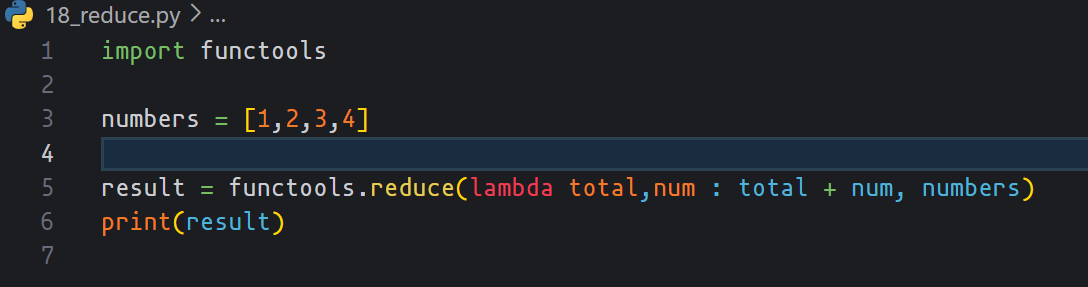
Un modulo puede ser un archivo de Python, como todos los que hemos esta creando en esta clase.



Python trae módulos predeterminados

## **functools**

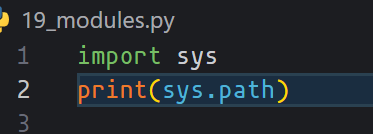
que trae el método reduce y otros mas



## **sys**

que pregunta sobre el sistema operativo

en el ejemplo estamos preguntando la ruta(path) del sistema

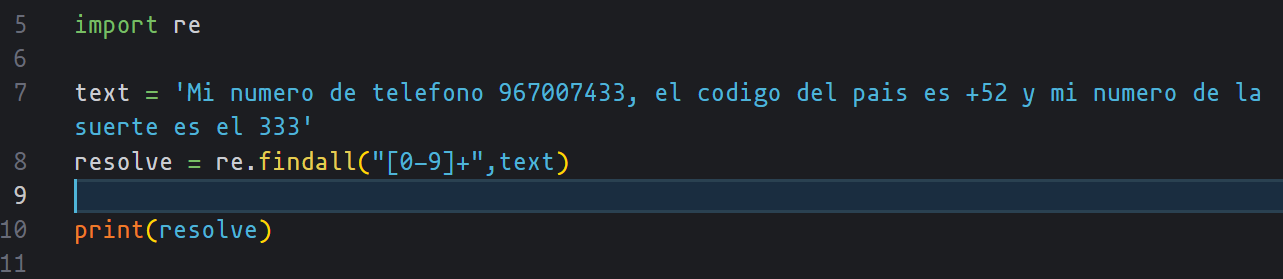




## **Re**

Permite utilizar expresiones regulares sobre algún contenido.

En el ejemplo, tenemos un texto a través de la librería re buscamos todas las coincidencias que contengan un numero dentro del texto

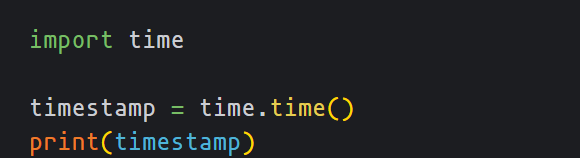




## **Time**

Permite manejar las horas y fechas

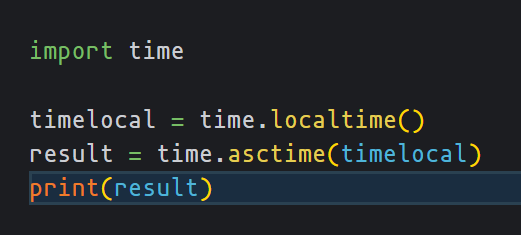
El método time() importa el numero de fecha entendido para la computadora, formato unix.





Para obtener la fecha que pueda ser entendida para los humanos primero debemos extraer el tiempo local con localtime() luego transformarlo con el método asctime()

La fecha se refiere a la fecha donde se esta corriendo en los servidores.

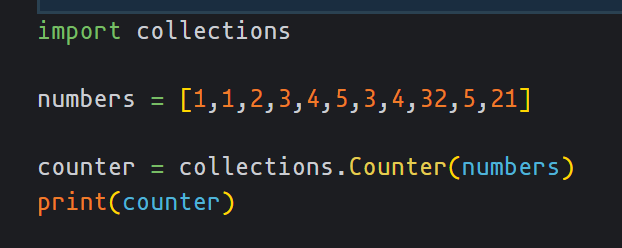




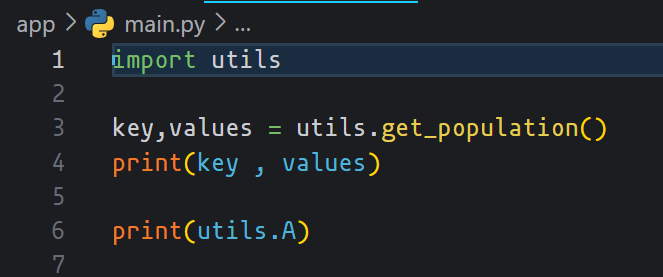
## **Collections**

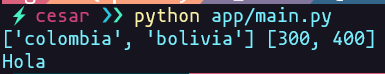
Permite utilidad para manejar listas

En el ejemplo se está usando el método counter que devolverá la frecuencia de una lista.





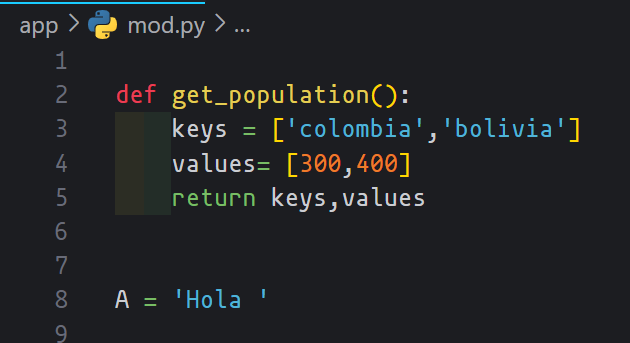




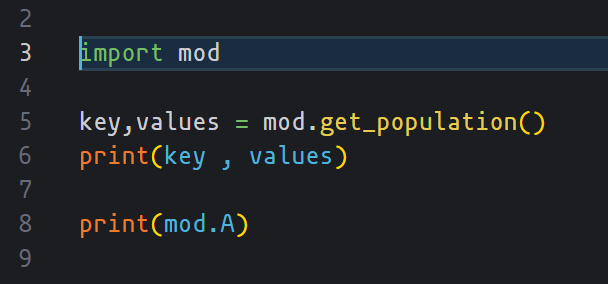
# MIS PROPIOS MODULOS

Todos los archivos py ya se comportan como módulos

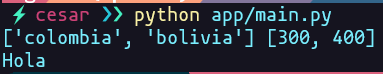
Mod.py



Main.py

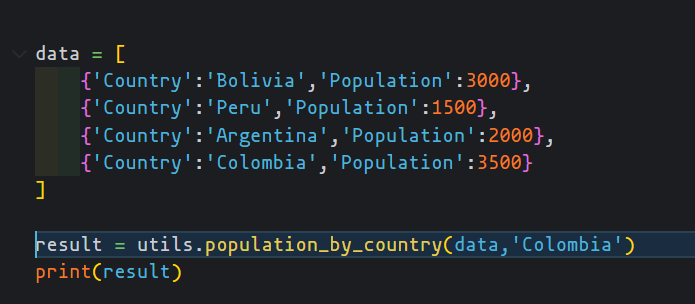


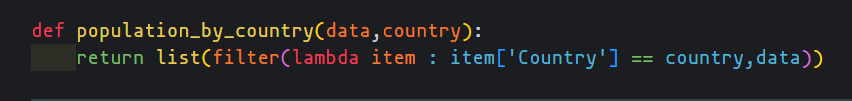
Salida:



En main.py

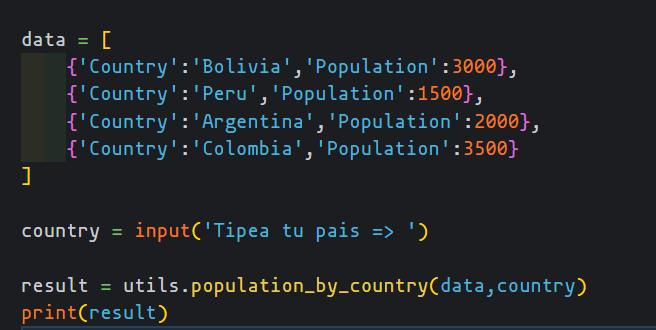


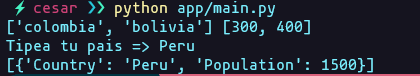




También podemos pedir la ejecución por script.

En este caso, pedimos el nombre del país que se desea información mediante input.





# MODULOS COMO SCRIPTS: \_\_NAME\_\_Y\_\_MAIN\_\_

Ahora que vimos lo que es módulos. Nosotros podemos corren un programa mediante un script o sea llamando al archivo desde el terminal

O en todo caso ejecutándolo como un módulo,

Los modulo en Python se pueden correr de 2 maneras

Declarando las funciones dentro de ese modulo como por ejemplo desde utils.py ya que tiene dentro las definiciones de las funciones

Y esas funciones se corren importándolas desde otro archivo como hacemos desde el archivo main.py donde llamamos a las funciones utilizando import desde main.py para utilizar las funciones que existen dentro de utils.py

También podríamos correr de forma directa como scripts la cual es la manera en como hemos estado corriendo nuestros archivos, yendo a la terminal colocando Python y la ruta archivo py

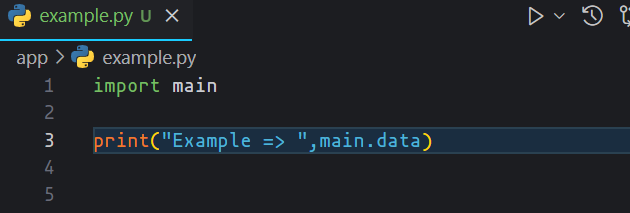
La otra manera de correr seria mediante otro archivo de manera modular.

la dualidad de poder correrlo desde un script o desde un módulo puede causar conflictos. ¿Como podemos corremos en ambas formas sin que nos de problemas?

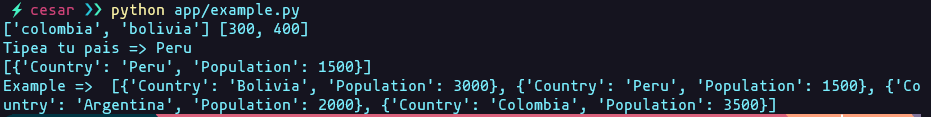
Del ejercicio anterior dentro de la carpeta app se encuentra main.py que se considera un módulo.

Creo u archivo nuevo y deseo imprimir el contenido de la variable data del archivo main,py

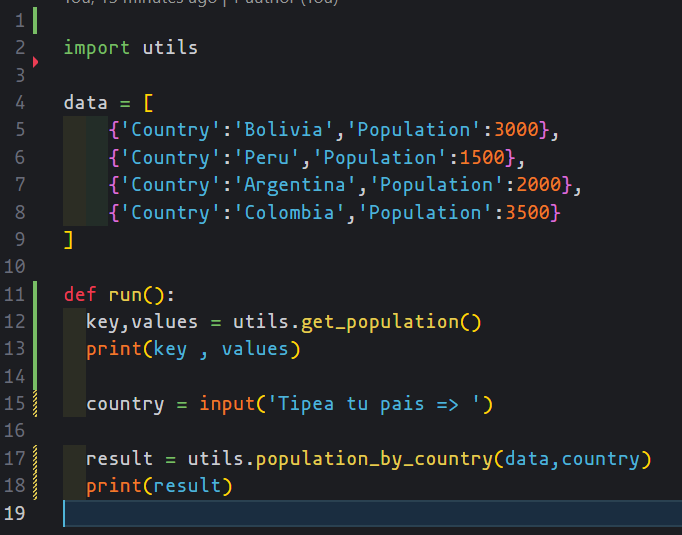
Por lo tanto, importo main e imprimo main.data

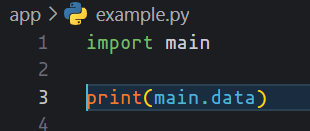


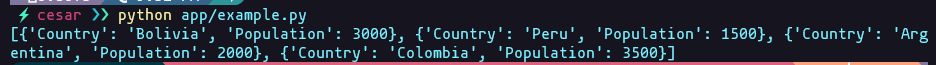
Se peude esperar que el resultado sea el contenido de data pero extrañamente imprime también el contenido imprimible de main.py



La solución pasa por modularizar, una manera seria meter el contenido que no se quiere ejecutar en una funcion



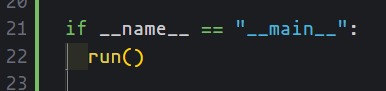




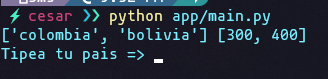
De esta manera podemos mostrar el contenido de data. Sin embargo si queremos ejecutar main.py como script no podemos

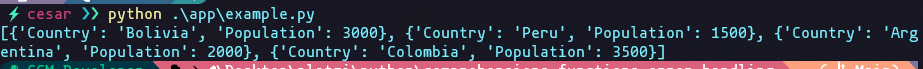


Para solucionar esto podemos emplear lo sgte



Esto indica que cuando se llame a main se ejecute la función run(). De esta forma podemos ejecutar desde el script llamando a main,py o modular desde example.py





# PAQUETES

Es una carpeta que contiene varios módulos

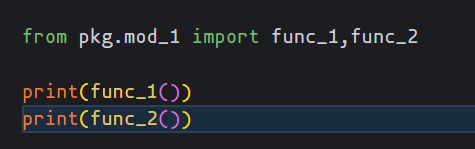
Creamos una carpeta pkg

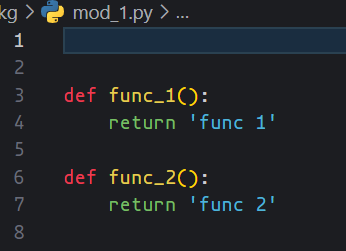


Un archivo main.py



Main.py puede importar desde el paquete pkg y acceder a las funciones func\_1 y func\_2 luego poder ejecutarlas e imprimir.

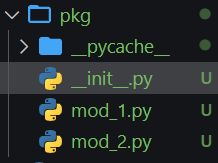




El paquete termina siendo un conjunto de varios módulos

Si se esta manejando una version de Python anterior a la 3.3 entonces necesitaas utilizar un archivo llamado

\_\_init\_\_.py



Desde la 3.3 en adelante serviría para:

Servirá para inicializar alguna variable o importación a ese paquete

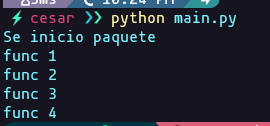
Nótese que al ejecutar main.py se imprime el resultado de las 4 funciones

Pero antes de eso se imprime el contenido de \_init\_\_.py

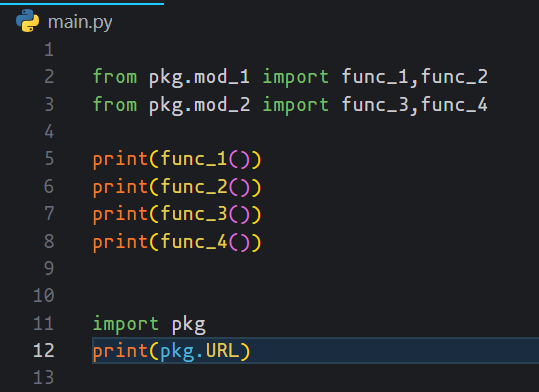
O sea que primero lee \_\_init\_\_.py y luego los archivos importados de main.py

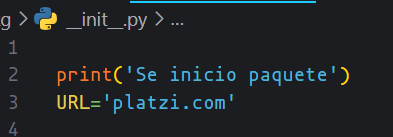
Cuando se llama los paquetes se crea variables. solo se llama el pkg una vez



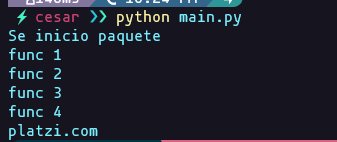


En este caso para probar que cuando se importa pkg se esta llamando automáticamente a \_\_init\_\_

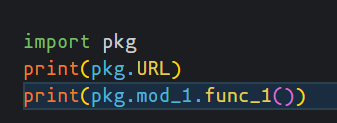


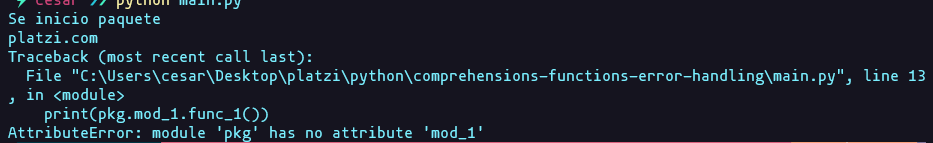


Importamos pkg y de allí ejecutamos URL , que esta en \_\_init\_\_.py

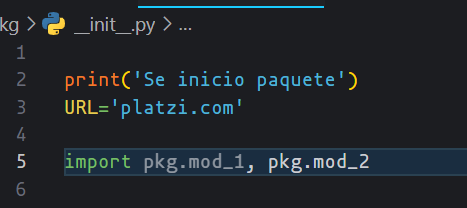


Podríamos pensar que desde main.py podemos importar las funciones de mod\_1 u otras. Sin embargo, no me deja por que no hacemos la importación explicita

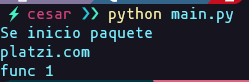




Esto se soluciona importando desde el archivo \_\_init.py

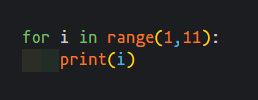


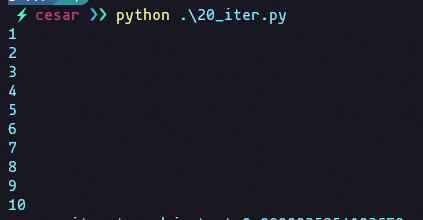




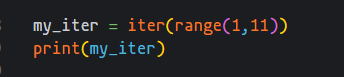
# ITERABLES

Creamos este for para iterar sobre un rango del 1 al 10 e imprimir cada uno de ellos.



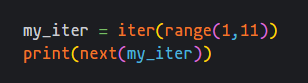


Si ejecutamos una iteración con un rango del 1 al 10 podemos almacenarla y imprimirla. Esto nos dará una valor de tipo object.





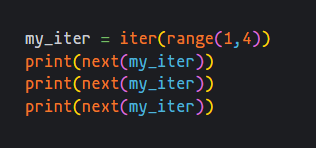
En este caso utilizamos el método next. Que sirve para iterar de manera manual.

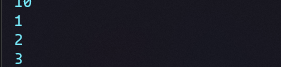




Mientras coloquemos mas iteradores mostraremos el siguiente elemento del rango.

¿Pero que pasa si llegamos al final?









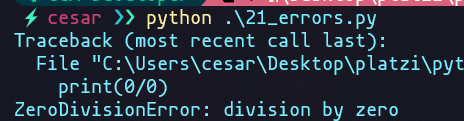
# ERRORES EN PYTHON

Veremos errores que se muestran en Python y nuestros errores personalizados.

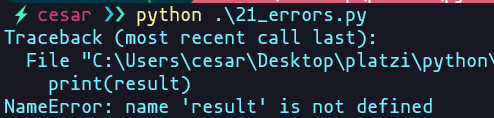
Podemos formar al error en Python.

El error zeroDivisionError se presenta cuando divides 0 entre 0.



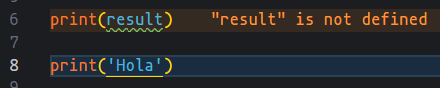


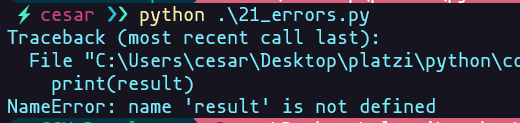
Ahora estamos haciendo referencia a una variable que no existe, nos devuelve otro tipo de error.



Luego vemos que después de intentar imprimir una variable no definida intentamos imprimir un string que no tiene errores.

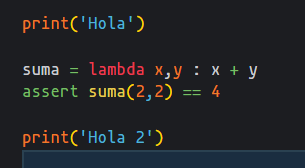
Al intentarlo notamos que el programa para y no ejecuta el código correcto.

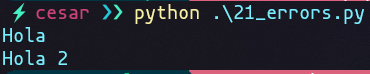




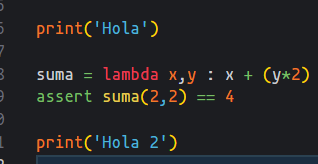
Otro ejemplo:

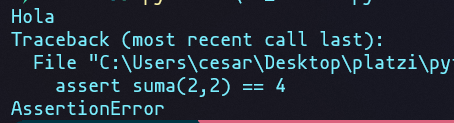
Tenemos una función lambda donde el resultado se verifica si su resultado es igual a 4. Como es correcto imprime “Hola 2”.





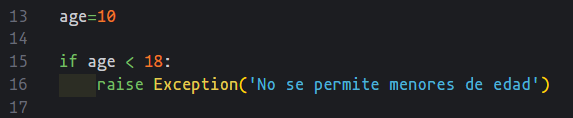
¿Qué pasa si no resulta? Pues te da un error AssertionError.

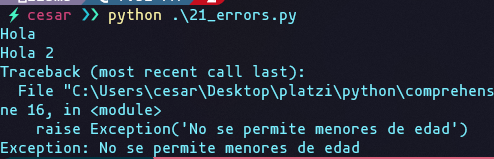




Con assert puedes hacer algún test unitario, pero siempre se recomienda herramientas especializadas para unit testing

Tu podrías lanzar tus propios errores utilizando la palabra reservada raise



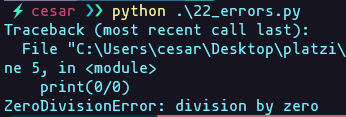


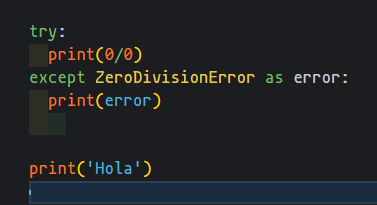
Sin embargo, así sea un error personalizado como el anterior o un error definido por Python una vez encontrado este error no se ejecutarán código después.

NO DEBERIAMOS PERMITIR SI SE DA ALGUN TIPO TODA LA APLICACIÓN CAIGA

# MANEJO DE EXCEPCIONES

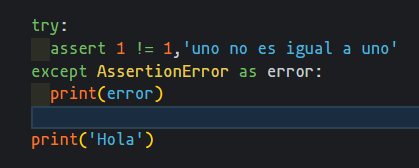


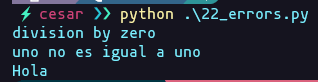


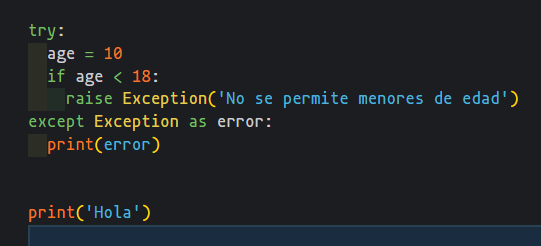


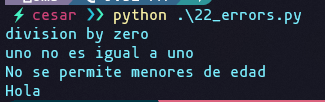


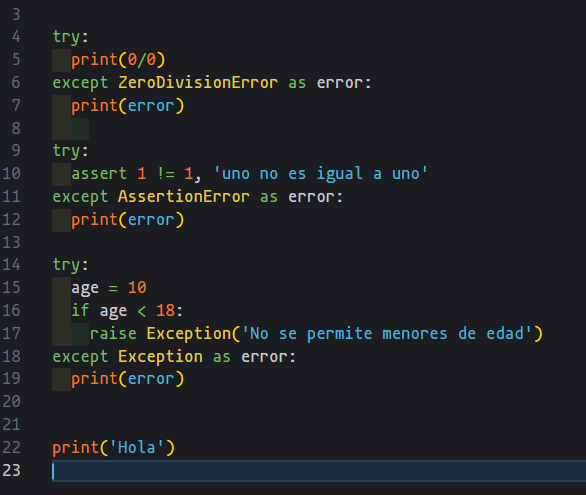
Sistema reporte de fallos

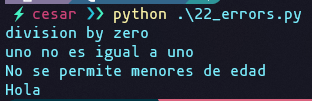








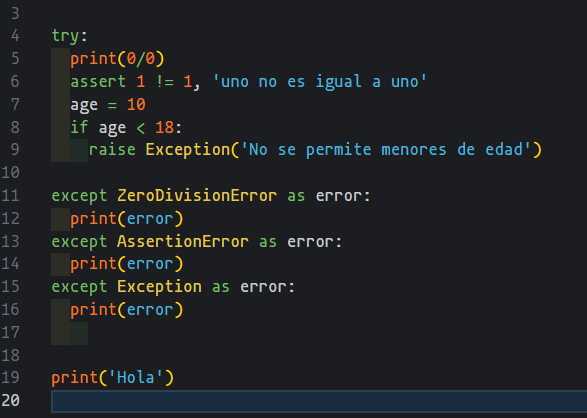




Percátate que también se puede escribir de esta manera en la cual si se da un tipo de error que disparara el print según el tipo de error y luego continuara ejecutándose

Encuentra el primer error: división by zero y luego imprime “hola”

Esto quiere decir que podemos agregar muchísimos tipos de errores

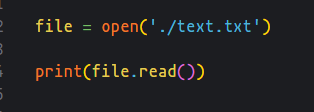




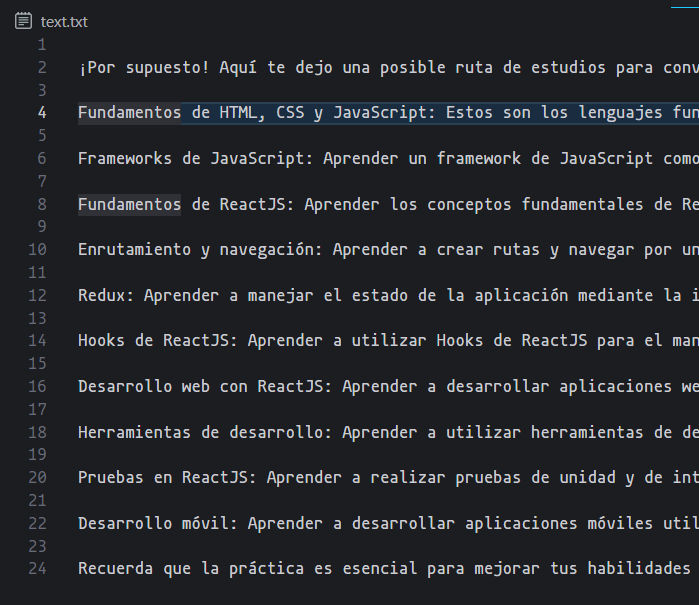
Dentro de cada except se podría agregar lógica

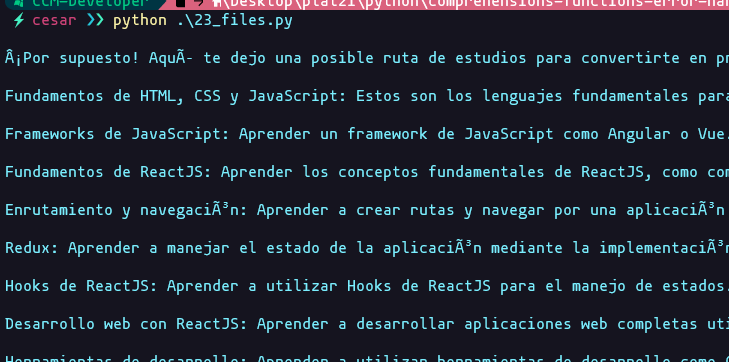
# LEER UN ARCHIVO DE TEXTO

Podemos leer archivos de texto plano

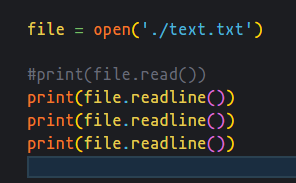


El archivo text.txt contiene esta información.

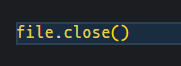




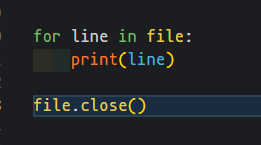
También puedes leer línea por línea:



También debemos cerrarlo y close liberar memoria



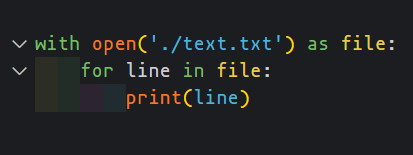
Para leer linea por linea



La manera correctar de escribir el código para leer el texto es la siguiente.

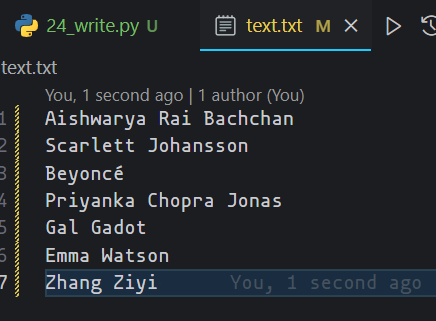
Acceder al archivo text.txt y renombrarlo como file

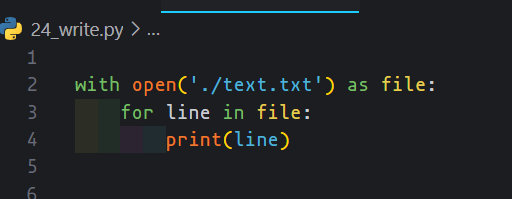
Luego hacer bucle sobre ese file line por line e imprimir line



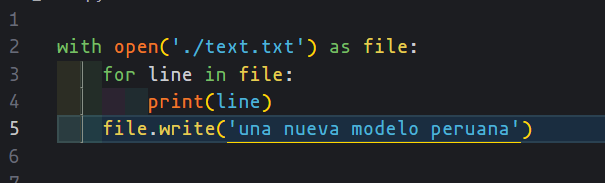
# ESCRIBIR UN ARCHIVO

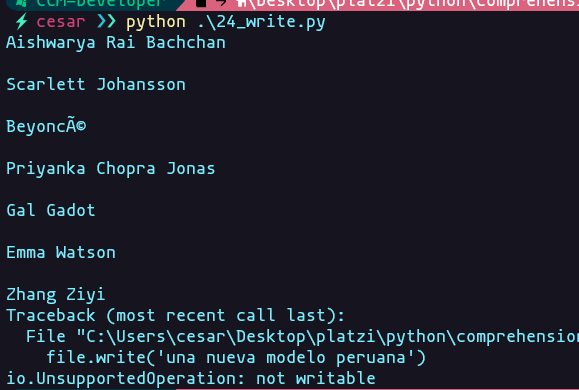
Tengo el sgte archivo text.txt:



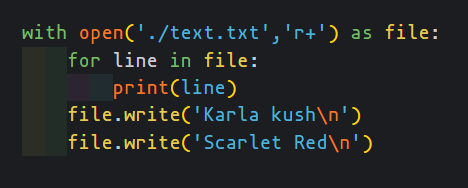


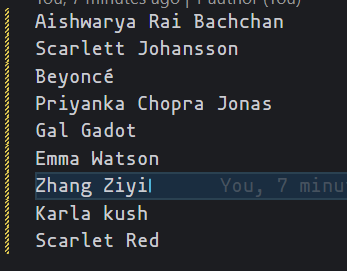
Intentamos escribir en un archivo con file.write pero no tenemos permisos. Esto es por que por defecto solo tenemos permiso para leer pero no para escribir o actualizar.



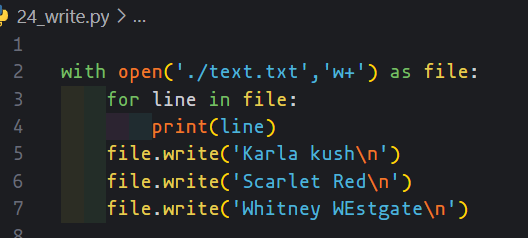


La solución es colocar un segundo argumento dentro de open : “r+” y para que escriba en una linea nueva \n

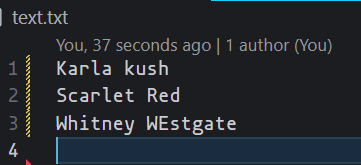




Si quisieras reemplazar todo por las líneas , utiliza w+



Y el contenido se reemplazara con karla kush, scarlet red …



# LEER UN CSV

Leeremos un archivo con filas y columnas



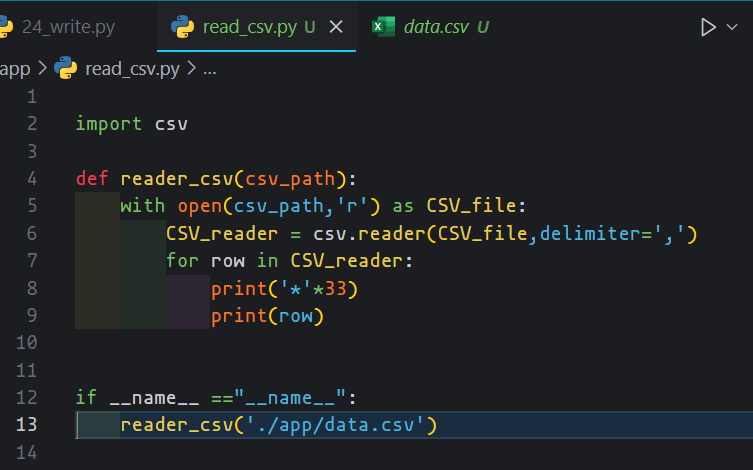
Explicación del código

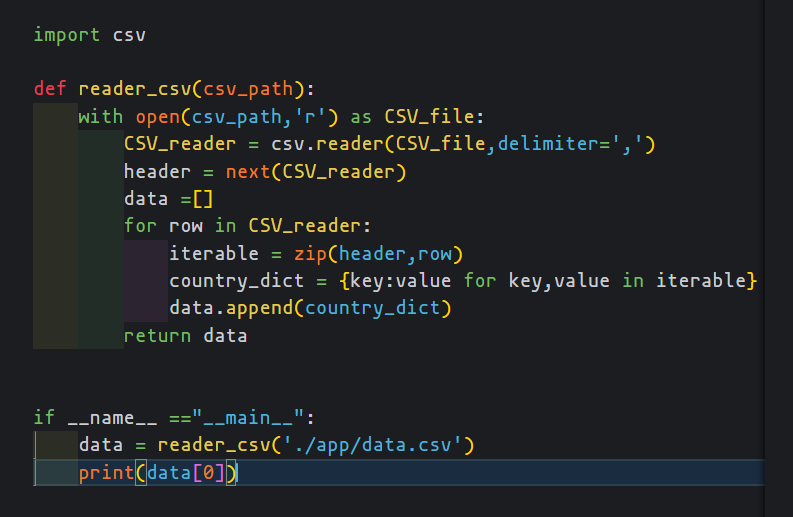
Importamos la librería csv que nos ayudara a leer el archivo

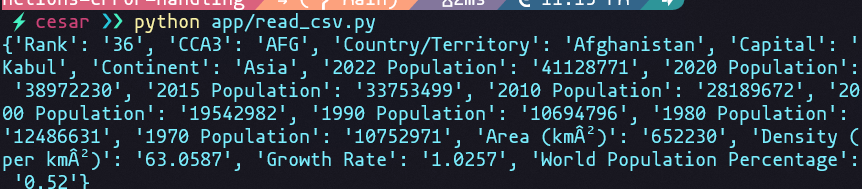
Creamos una función reader\_csv que recibe un string que será un path de ruta del archivo que queremos procesar

Dentro de reader\_csv:

Abrimos el archivo y le damos permiso para ser leído (reader)







# CREANDO UNA GRAFICA

