Bitacora de PHYTON

Nicolas Caro Ñustez

Tabla de contenido

[Historia de Payton y características 3](#_Toc33133262)

[Sintaxis básica 3](#_Toc33133263)

[Sintaxis Básica 2 4](#_Toc33133264)

[Funciones 7](#_Toc33133265)

[Sintaxis Básica 4(Funciones 2) 8](#_Toc33133266)

[Listas 9](#_Toc33133267)

[Sintaxis Básica 6 10](#_Toc33133268)

[Sintaxis Básica 7(Diccionarios) 12](#_Toc33133269)

[Condicionales 1 13](#_Toc33133270)

[Condicionales 2 14](#_Toc33133271)

[Condicionales 3 14](#_Toc33133272)

[Condicionales 3 16](#_Toc33133273)

[Bucles 17](#_Toc33133274)

[Bucles 2 18](#_Toc33133275)

[Bucles 3 20](#_Toc33133276)

[Bucles 4 21](#_Toc33133277)

[Bucles 6 23](#_Toc33133278)

[Generadores 24](#_Toc33133279)

[Generadores 2 27](#_Toc33133280)

[Excepciones 28](#_Toc33133281)

[Excepciones 2 30](#_Toc33133282)

[Excepciones 3 31](#_Toc33133283)

# Historia de Phyton y características

-Creado por Guido Van Rossum a comienzos de los 90

-Lenguaje de alto nivel. Gramática sencilla, clara, muy legible.

-Tipado dinámico (No es necesario establecer la variable) y fuerte

-Orientado a objetos.

-Sobrecarga de constructores. Herencia múltiple. Encapsulación. Interfaces. Polimorfismo

-Open Source.

-Fácil de aprender

-Librería estándar muy amplia

-Interpretado

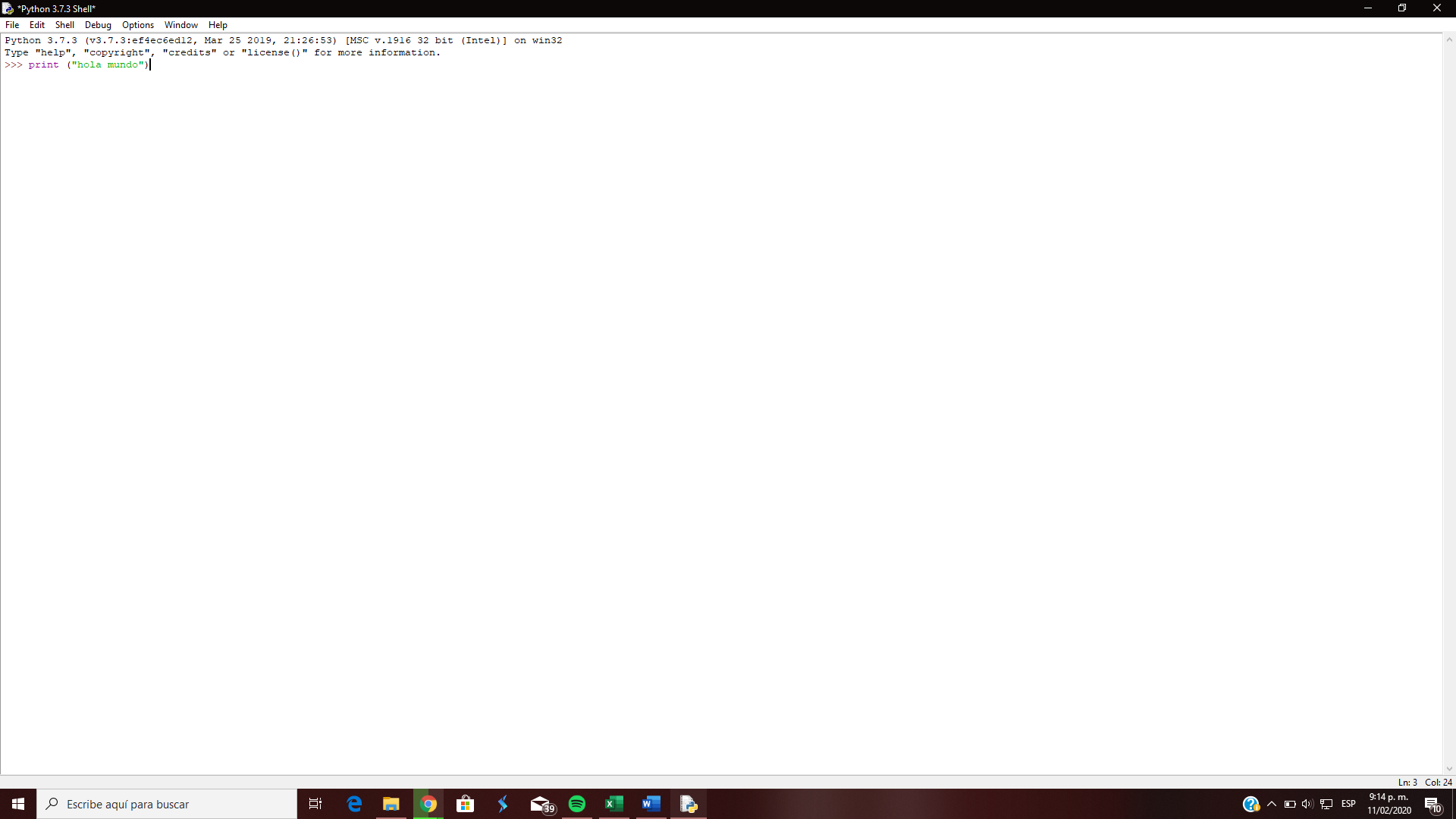
-Versátil

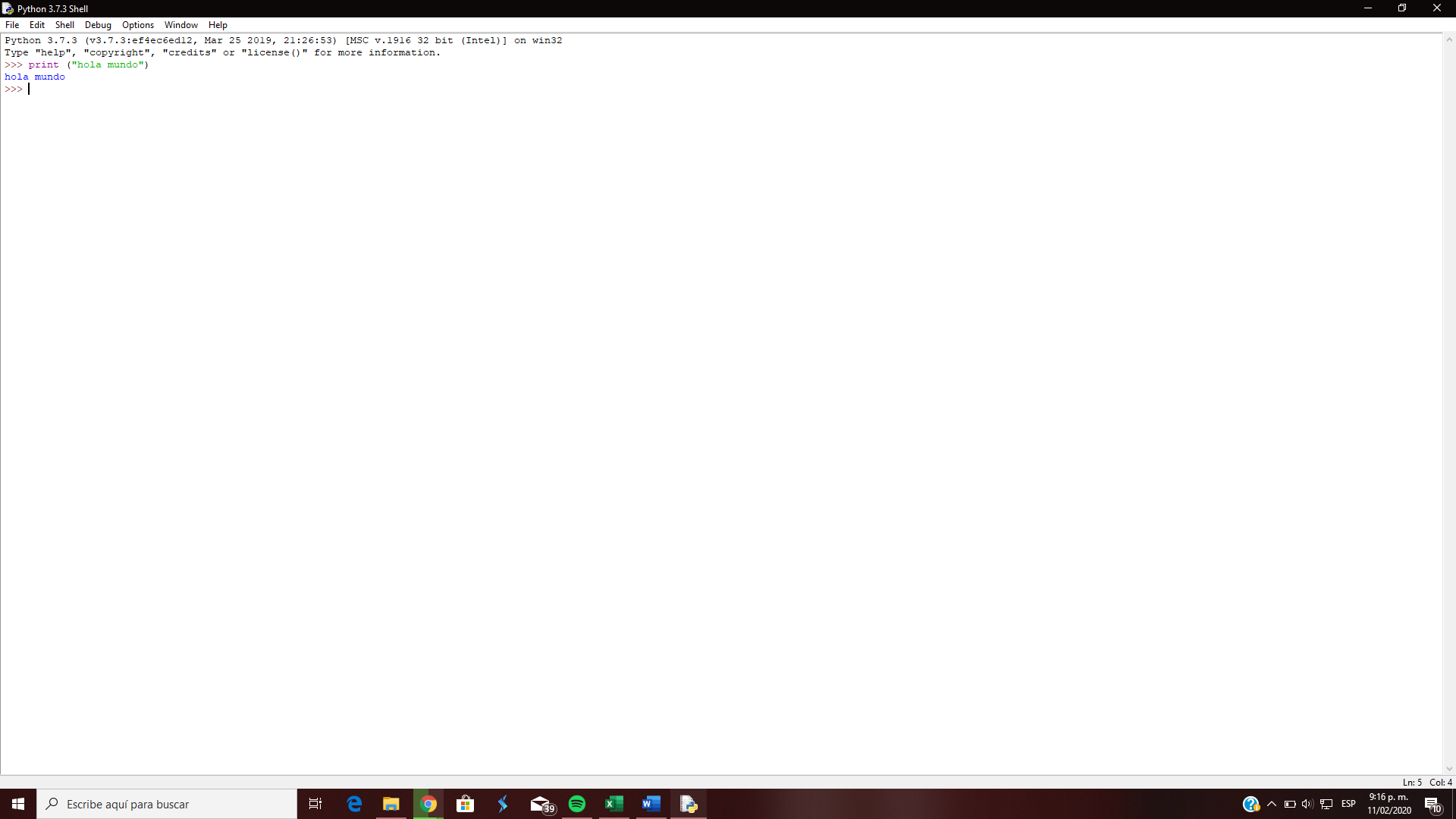
-Aplicaciones de escritorio, aplicaciones de servidor, aplicaciones web.

-Multi plataforma

# Sintaxis básica

>>> Este símbolo es lo que se le conoce como el prompt,es el punto exacto dentro de la consola donde introduciremos las instrucciones

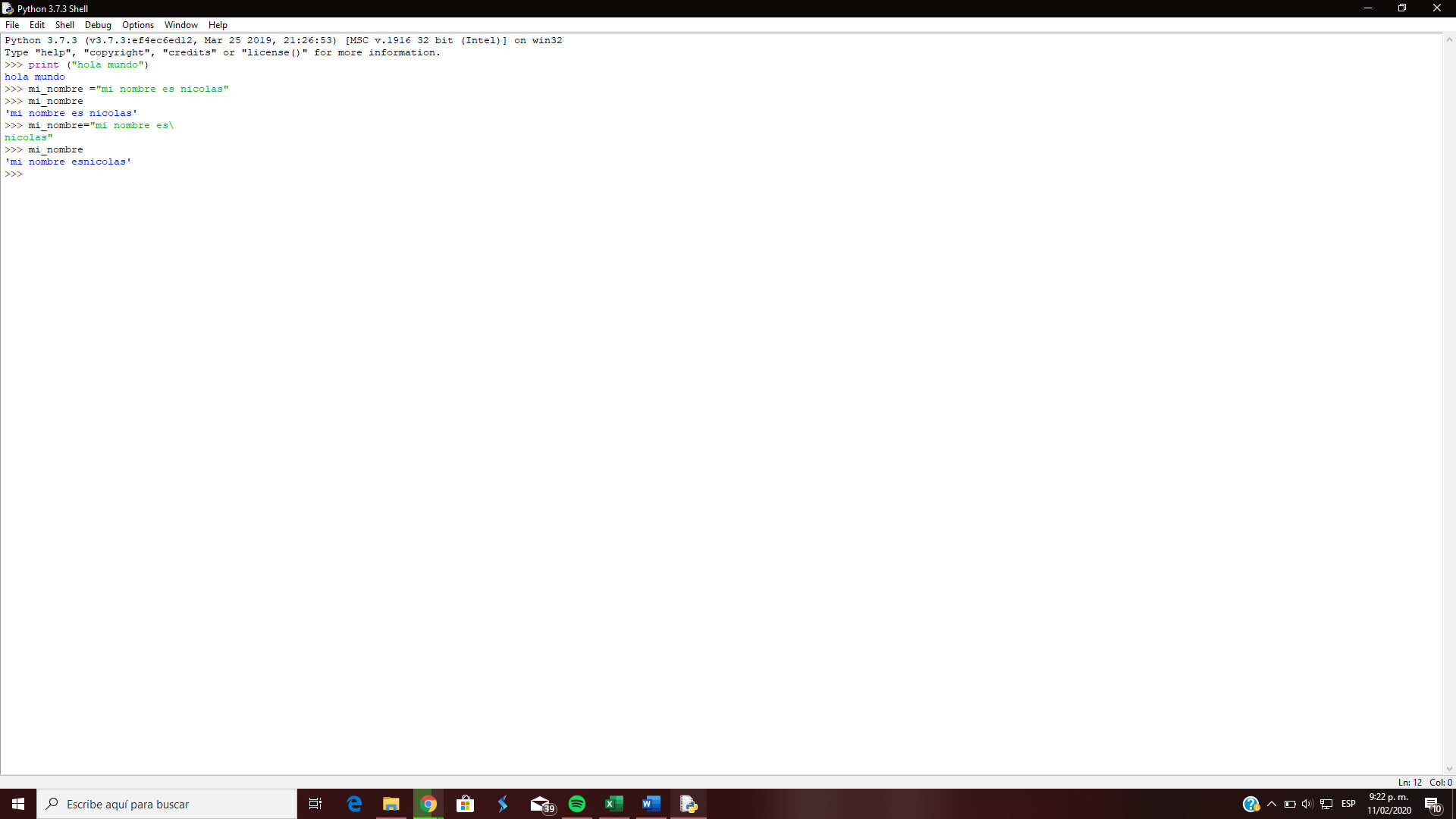
A esto en payton se le denomina como una instrucción, cada instrucción no lleva punto y coma.



Este seria el resultado de dicha instrucción. Cada instrucción se hace en una línea diferente

Los comentarios se hacen con las almohadillas (#)

La barra invertida se usa para separar las instrucciones en varias líneas \

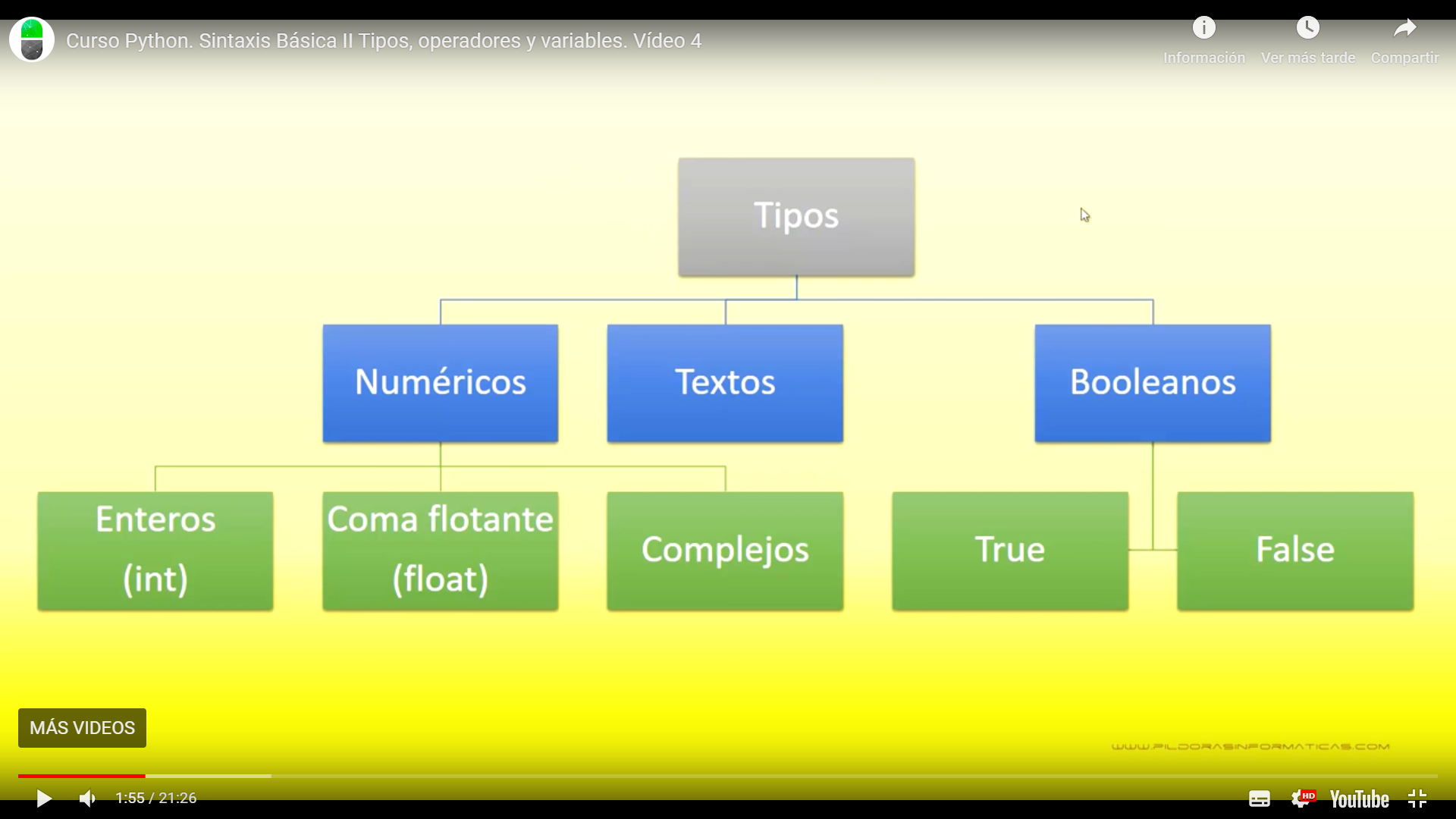


Aquí se puede ver como seria el uso de la barra invertida, no es recomendable usarlo y en pocas ocasiones servirá realmente

Identación: dejar una pequeña sangria al momento de hacer bloques de código para cada bloque de programación

# Sintaxis Básica 2

Tipos de datos

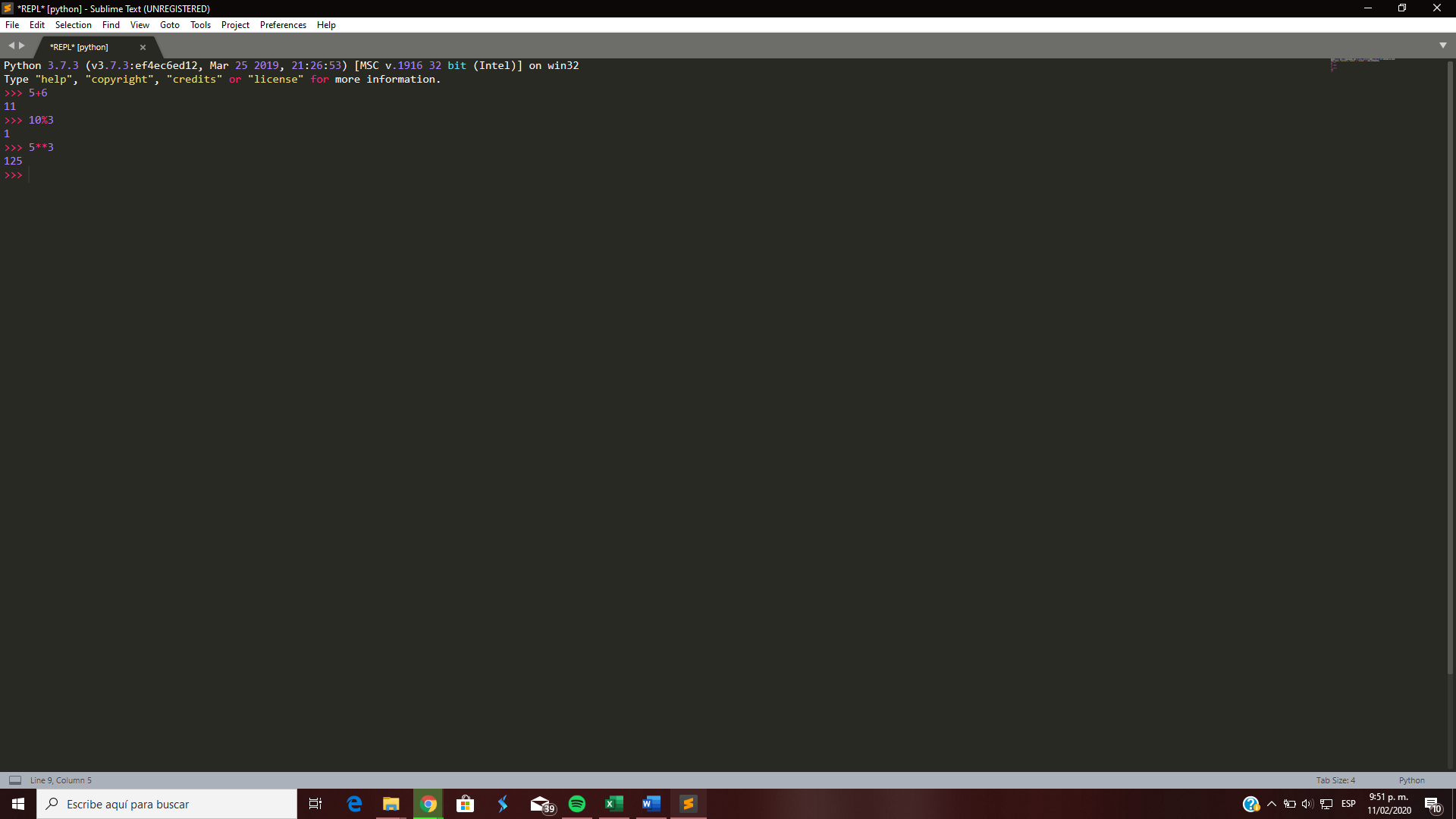


Los textos pueden llegar a llevar comilla simple, doble o hasta triple.

Imagen que contiene captura de pantalla

Descripción generada automáticamenteOPERADORES

Variable es un espacio en la memoria del ordenador donde se almacenara un valor que podrá cambiar durante la ejecución del programa



El operador exponente se usa con dos \* así como se denota en la imagen

Declaración de una variable

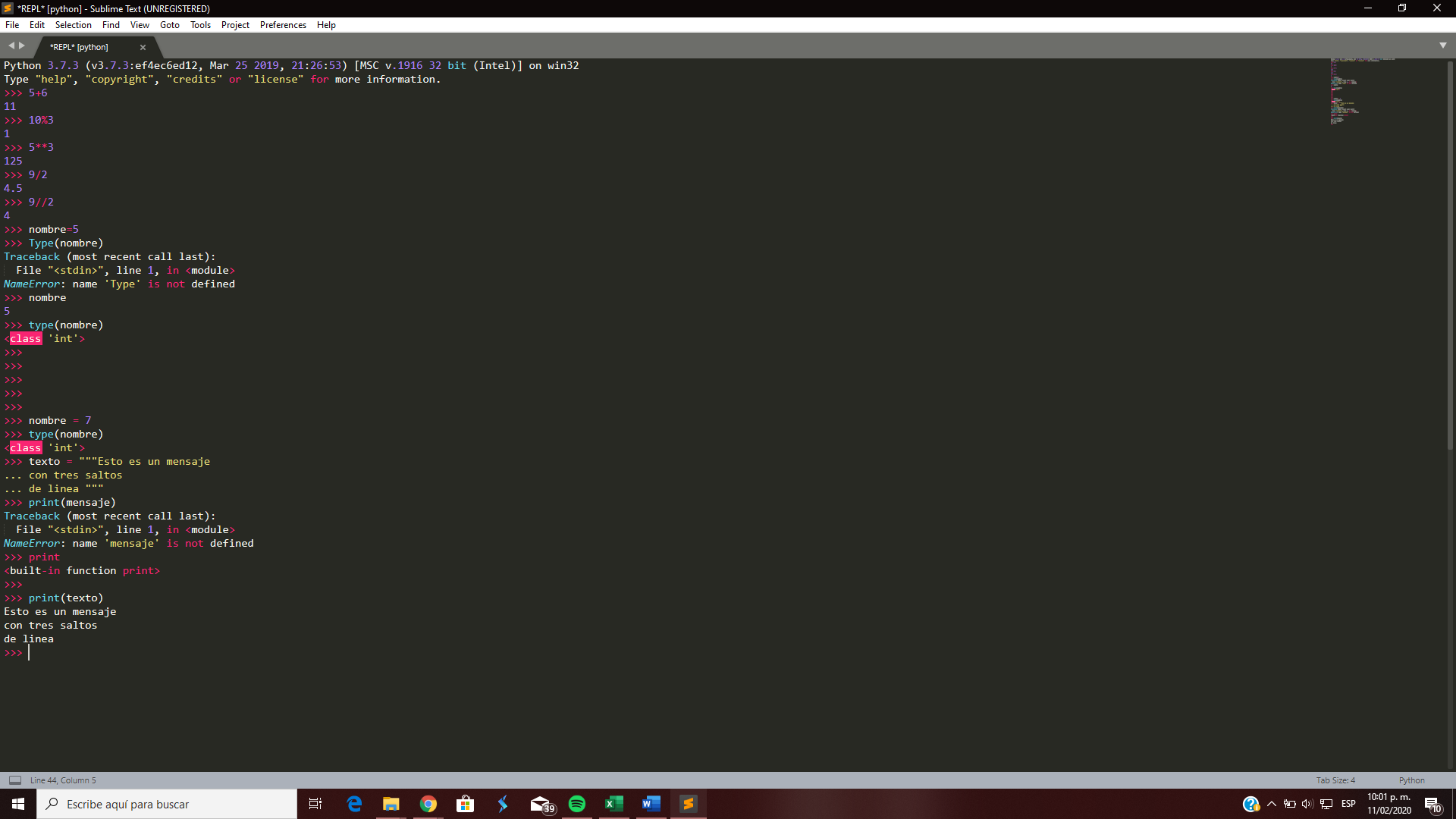
Se usa \_ para dos o mas palabras, el tipo de variable en payton lo define el contenido es decir no hay que declarar la variable

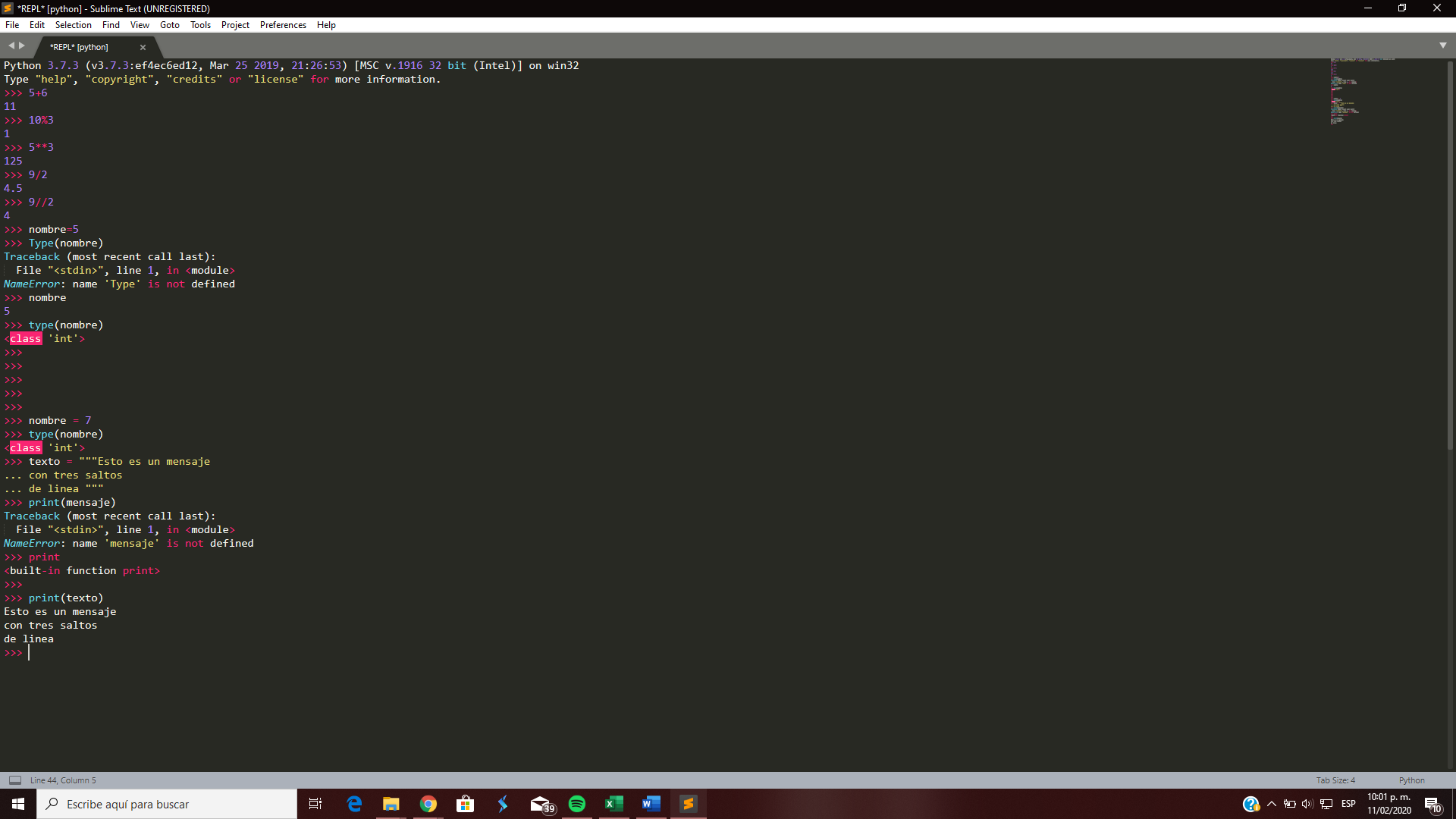
Absolutamente todo es un objeto en payton



Ejemplo de asignación y comprobación del objeto

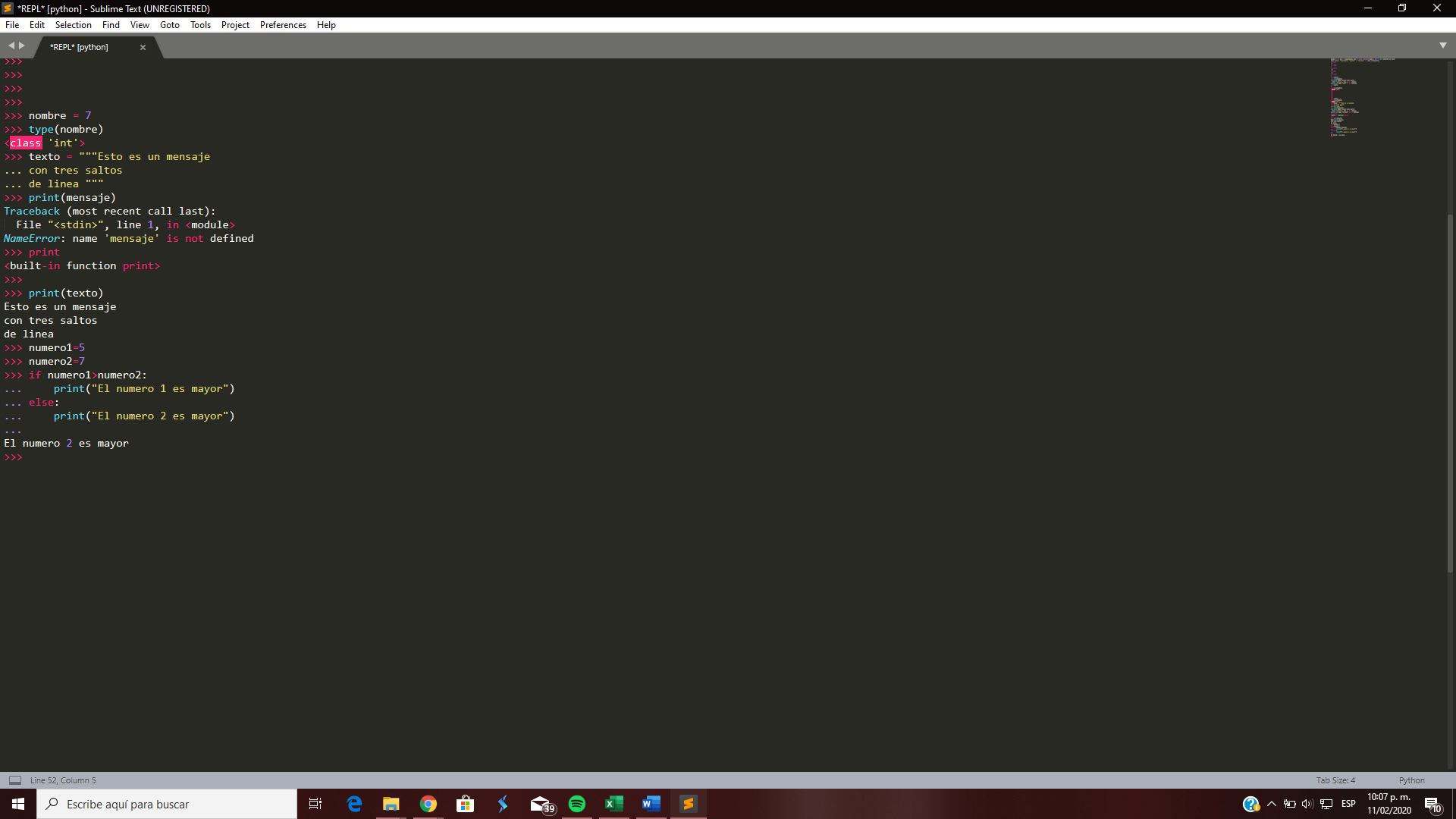
las tres comillas son utilizadas para el salto de línea

 Así se debería usar las tres comillas



Así se mostrara el mensaje

CONDICIONAL IF para ver los comparadores de mayor que



# Funciones

-Una función es una línea de código agrupadas que funcionan como una unidad realizando una tarea especifica

-Las funciones en Python pueden devolver valores

-Las funciones en Python pueden tener parámetros/argumentos

-A las funciones también se las denomina “métodos” cuando se encuentran definidas dentro de una clase.

Utilidades

-Reutilización de código

SINTAXIS DE UNA FUNCION EN Python

* 0def nom\_función():

-Instrucciones de la función

-return(opcional)

* Def nombre\_función(parámetros)

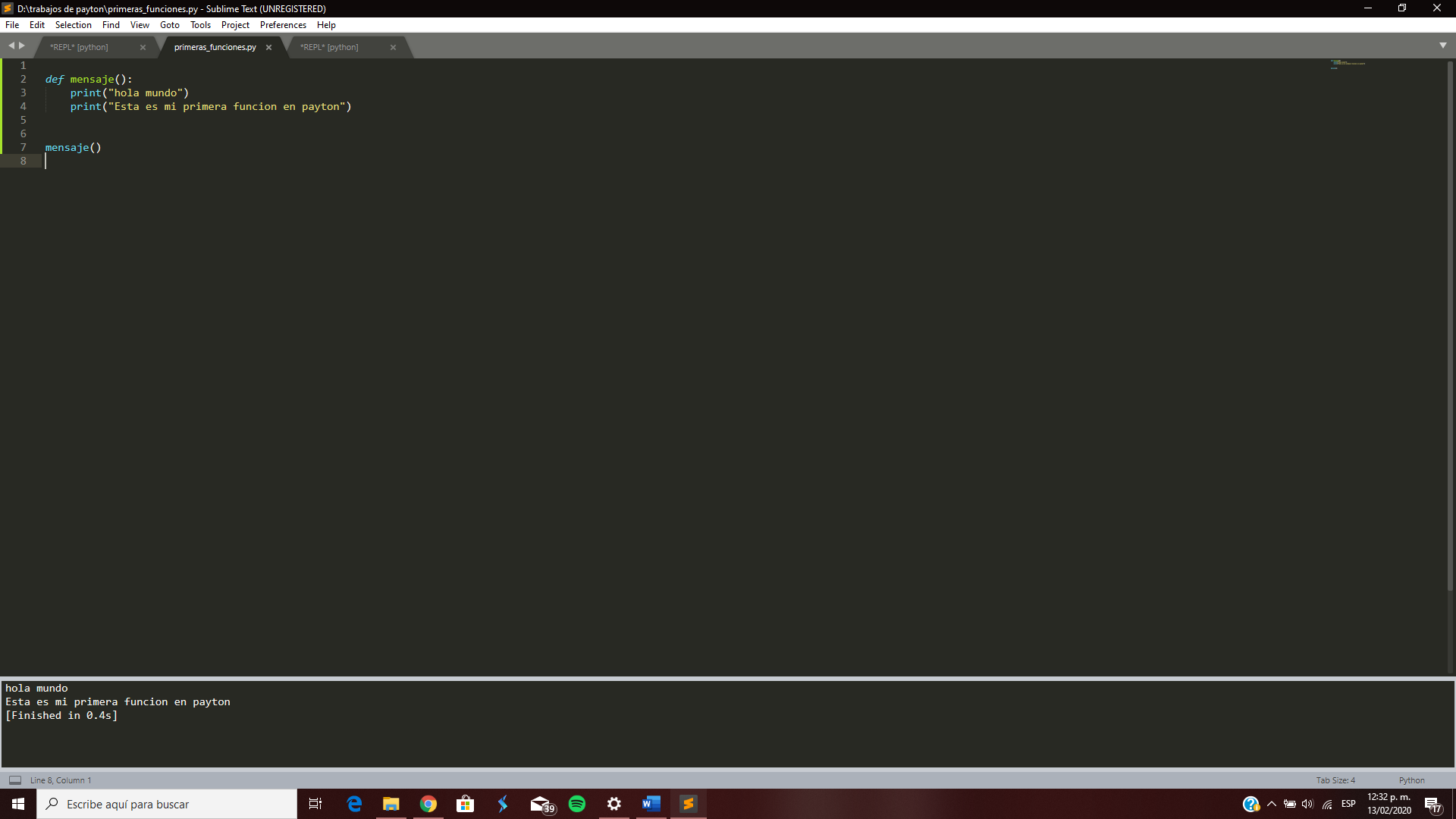
-Instrucciones de la función

-return(opcional)

Declaración

Al momento de guardar un archivo Python se guarda con extensión.py

Ejemplo:



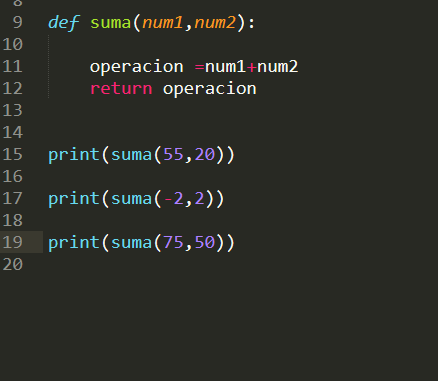
Llamada de la función

# Sintaxis Básica 4(Funciones 2)

Imagen que contiene captura de pantalla

Descripción generada automáticamenteFunciones con parámetro de entrada:

Funciones con return



# Listas

-Son estructuras de datos que nos permiten almacenar una gran cantidad de valores (equivalente al arrayz en otros lenguajes de programación)

-En Python las listas pueden guardar diferentes tipos de valores

-Se pueden expandir dinámicamente añadiendo nuevos elementos (otra novedad respecto a los arrays en otros lenguajes)

Sintaxis lista

Nombre lista =[elem1,elem2,elem3….]

Imagen que contiene captura de pantalla

Descripción generada automáticamente

# Sintaxis Básica 6

Las Tuplas: Son listas inmutables, es decir, no se pueden modificar después de su creación:

* No permiten añadir, eliminar, mover elementos etc. (no append, extend,remove)
* Si permiten extraer porciones, pero el resultado de la extracción es una tupla nueva
* Si permiten comprobar si un elemento se encuentra en la tupla

¿Qué utilidad o ventaja tienen respecto a las listas?

* Más rápidas
* Menos espacio (Mayor optimización)
* Formatean Strings
* Pueden utilizarse como claves en un diccionario. (Las Listas no)

Sintaxis de las tuplas

nombreTupla = (elem1, elem2, elem3…)

Imagen que contiene captura de pantalla

Descripción generada automáticamente

# Sintaxis Básica 7(Diccionarios)

* Estructura de datos que nos permite almacenar valores de diferente tipo e incluso listas y otros diccionarios.
* La principal característica de los diccionarios es que los datos se almacenan asociados a una clave de tal forma que se crea una asociación tipo clave: valor para cada elemento almacenado.
* Imagen que contiene captura de pantalla

  Descripción generada automáticamenteLos elementos almacenados no están ordenados. El orden es indiferente a la hora de almacenar información en un diccionario.

# Condicionales 1

* Instrucción IF

Imagen que contiene captura de pantalla

Descripción generada automáticamente

# Condicionales 2

* Instrucción IF. Instrucción else y elif

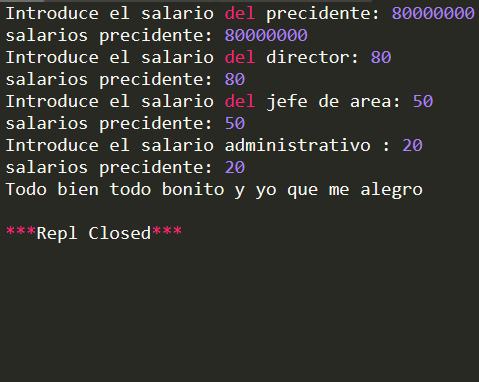
Nota: el elif se anota justo después del primer if y se usa para encadenar todos los if

Imagen que contiene captura de pantalla

Descripción generada automáticamente

# Condicionales 3

* Switch no existe en peython ofrece muchas alternativas
* Permite la concatenación de operadores de comparación
* Operadores lógicos “and” Y “or”
* Operador “in”

Imagen que contiene captura de pantalla

Descripción generada automáticamente

EL str() se utiliza para convertir una variable en string

Este es el resultado del código anterior

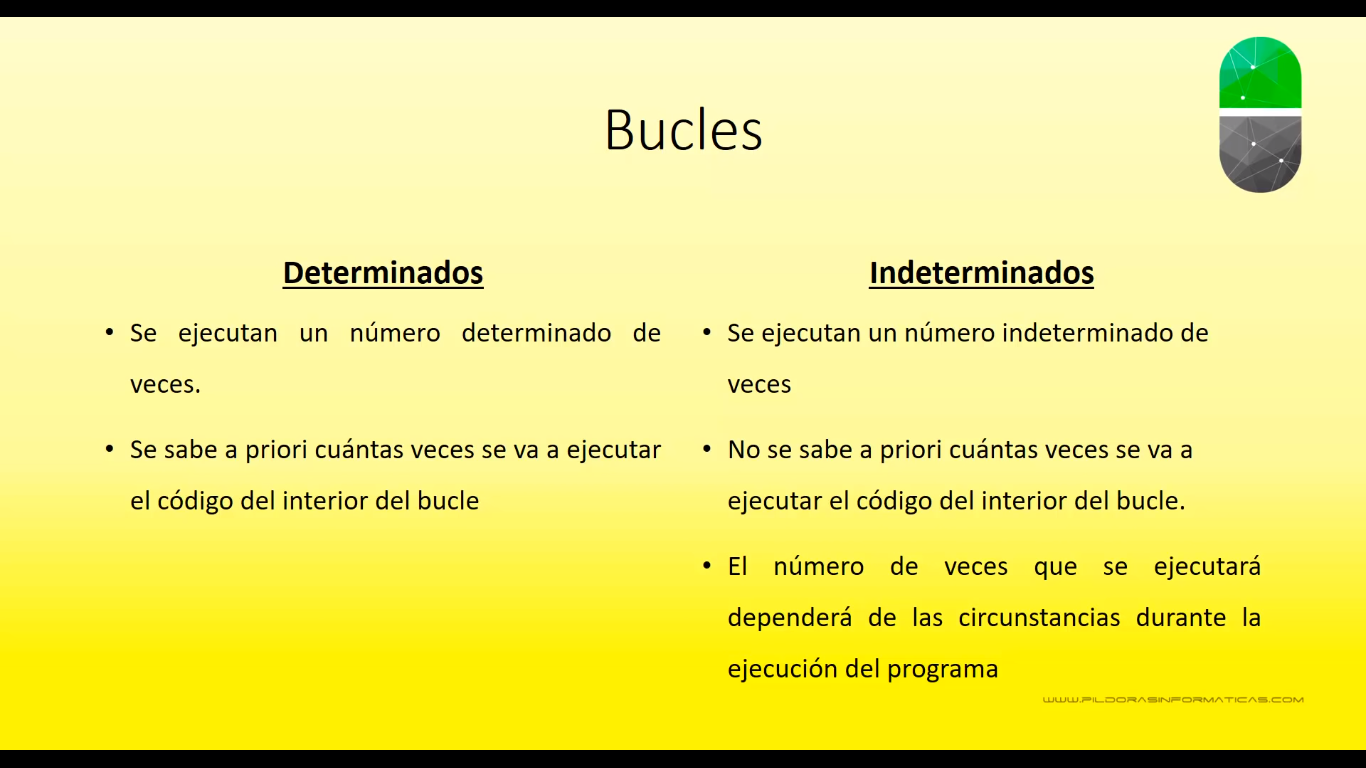
# Imagen que contiene captura de pantalla Descripción generada automáticamenteCondicionales 3

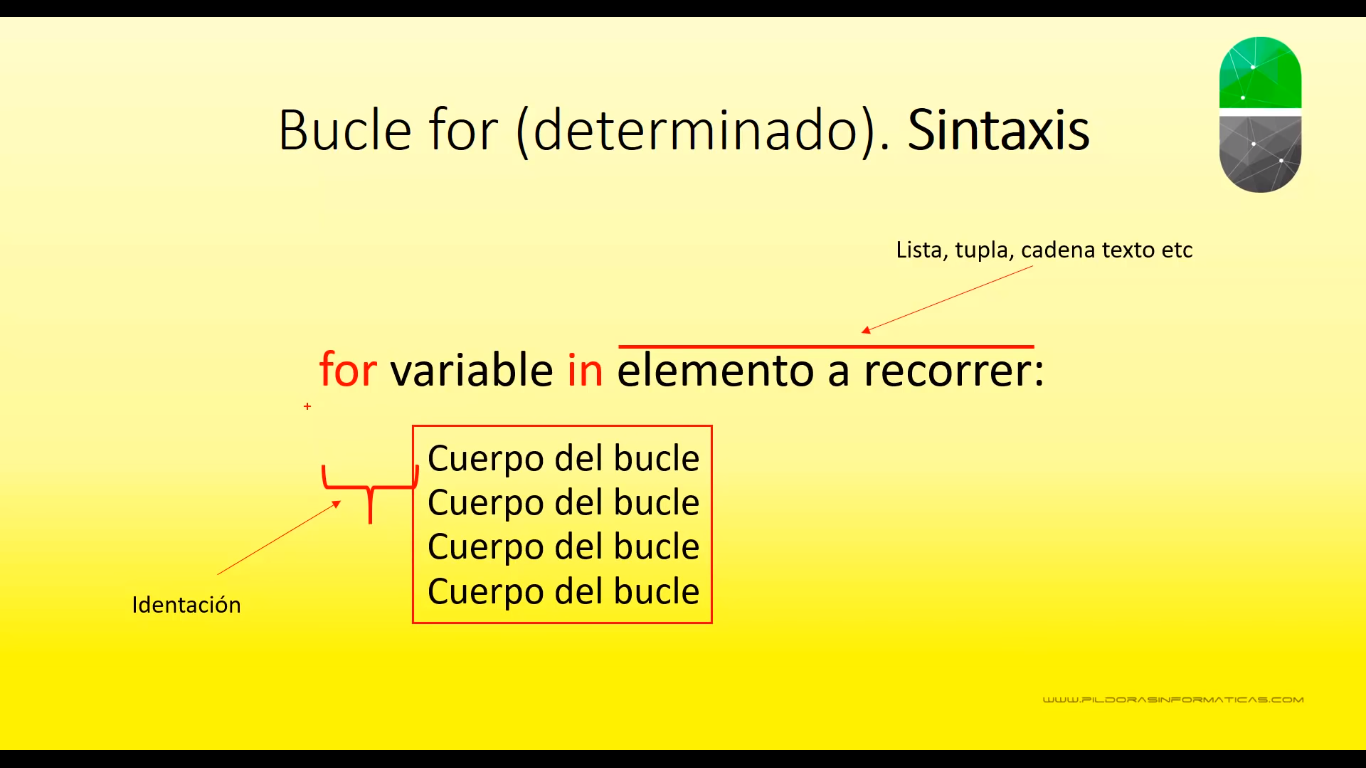
Imagen que contiene captura de pantalla

Descripción generada automáticamente

# Bucles

Bucles estructura de control que repite una serie línea de código y saldrá del bucle según la condición que tiene le bucle





# Bucles 2

#for i in [1,2,3,4,"ha","e"]:

# print("Hola")

#email = False

#for i in "santiech\_99@hotmail.com.ar":

# print("Hola", end=", ")

#\*miEmail = input("Introduce tu direccion de email: ")

#for i in miEmail:

# if i == "@":

# email = True

#if email:

# print("Email correcto")

#else:

# print("El email no es correcto")

contador = 0

miEmail = input("Introduce tu direccion de email: ")

for i in miEmail:

if (i == "@" or i == '.'):

contador = contador +1

if contador == 2 or contador == 3:

print("Email correcto")

else:

print("El email no es correcto")

La palabra ***rango*** es un tipo en Python y es como un arreglo de ciertos elementos en este caso 5, que se mostraría asi: 0,1,2,3,4.

Esto es para que las interacciones que están dentro del for se realicen la cantidad de veces que sea rango.

for i in range(5):

print ("hola")

# Bucles 3



for i in range(5):

print ("hola")

for i in range(5):

print(f"El valor de la variable es: {i}") #la *f* es para concatenar la impresión con variables

print("------------------------------")

for i in range(5,10):

print(f"El valor de la variable es: {i}")

print("------------------------------")

for i in range(5,50,3):# empieza desde el 5 termina en el 49 y aunmenta de 3 en 3

print(f"El valor de la variable es: {i}")

print("-----------------------------")

valido = False

email=input("Introduce tu email: ")

for i in range(len(email)): #Recorre el tamaño del string(cadena de caracteres) y empieza a contar desde cero

if email[i] == '@':

valido= True

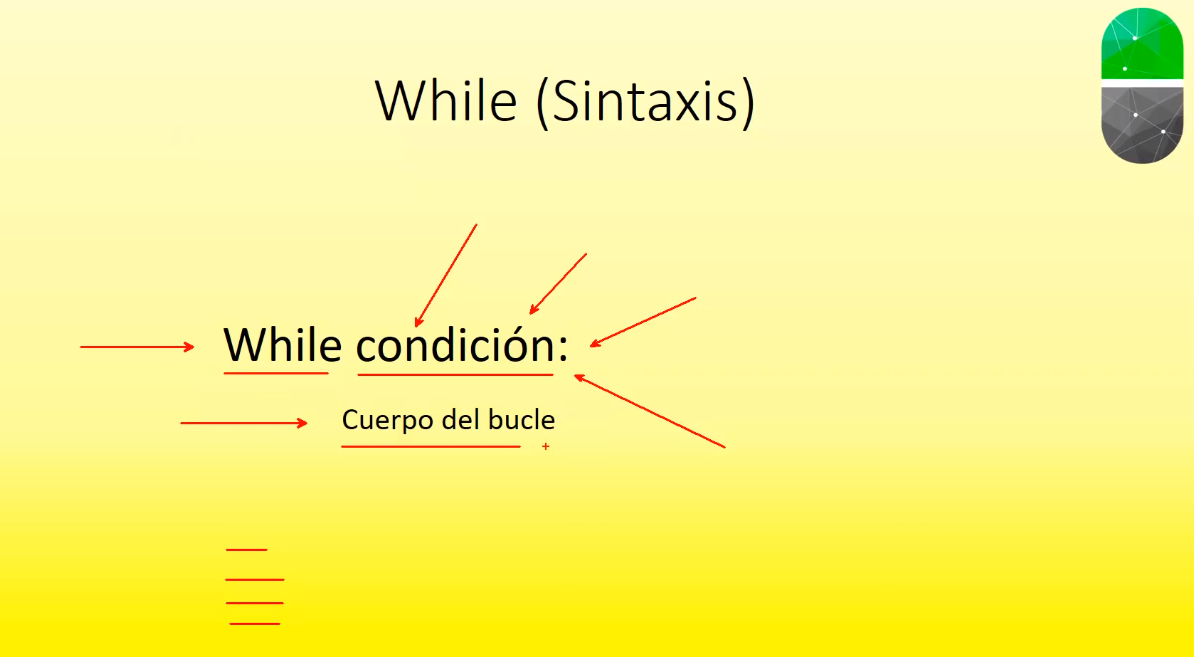
if valido:

print("Email correcto")

else:

print("Email incorrecto")

# Bucles 4



Este bucle es indefinido, ya que no se sabe con exactitud cuantas veces se va a repetir, también es importante recalcar que mientras la condición sea verdadera, se va a seguir repitiendo las cantidad de veces necesaria para que la condición sea falsa.

import math

i=1

while i <= 10:

print("Ejecucion "+ str(i))

i=i+1

print("Terminó la Ejecucion")

print("------------------------------")

edad=int(input("Introduce la edad por favor: "))

while edad < 6 or edad > 100:

print("Has introducido una edad incorrecta. Vuelve a intentarlo")

edad=int(input("Introduce la edad por favor: "))

print("Edad correcta, puedes pasar")

print("Edad de la persona: "+str(edad))

print("---------------------------------")

print("Programa de cálculo de raíz cuadrada")

numero=int(input("Introduce un número por favor: "))

intentos = 0

while numero < 0:

print("No se puede hallar la raíz´de un número negativo")

if intentos == 2:

print("Has hecho demasiados intentos. El programa ha finalizado")

break;

numero=int(input("Introduce un número por favor: "))

if numero < 0:

intentos = intentos +1

if intentos < 2:

solucion = math.sqrt(numero)

print("La raíz cuadrada de "+str(numero) +" es "+ str(solucion))

# Bucles 6

Continue**:**

Pass**:** Esta instrucción es para volver un bucle nulo, como si nunca se ejecutara, o para volver una clase nula también. Esta instrucción se escribe adentro. De lo que vaya a usar

Else**:** En un bucle funciona diferente que, en un if, ya que, en un bucle, este *else* es leído cuando el bucle está vacío es decir cuando ya no tiene más iteraciones, cuando ya halla terminado de recorrer lo que tenga que recorrer, en otras palabras cuando ya acabo todas sus iteraciones y después de eso se cumple el else de un bucle pero si se llega a cumplir lo que hay adentro el else no se itera.

Pero el bucle se tiene que romper con un break para que no quede vacío y no a ejecutar la línea del *else.*

for letra in "Python":

if letra == "h":

continue

# Ese continue lo que hace es ignorar las siguientes líneas de código del bucle y pasar a la # siguiente iteración, después del continue ignora las líneas siguientes del bucle y salta a la # siguiente iteración

print("Viendo la letra: " +letra)

print("-------------------------")

#Hacer un programita que cuente solo las letras de un frase

nombre = "Píldoras Informáticas"

contador=0

for i in nombre:

if i ==" ":

continue

contador+=1

print(contador)

print("--------------------------")

email=input("Introduce tu email, por favor: ")

for i in email:

if i == "@":

arroba=True

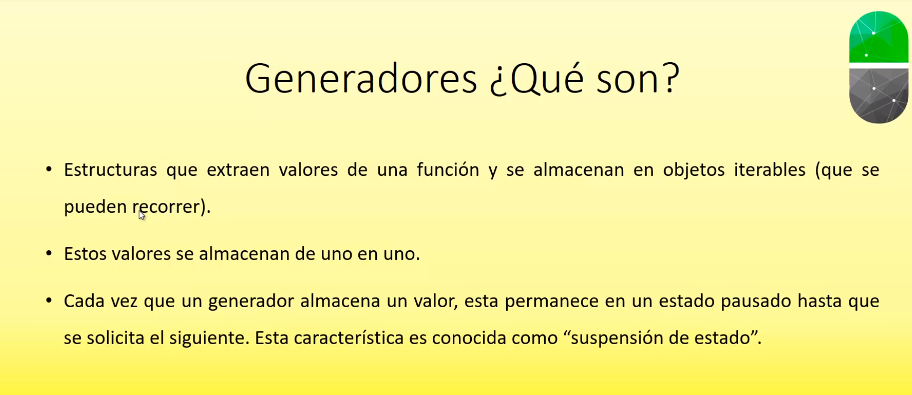
break;

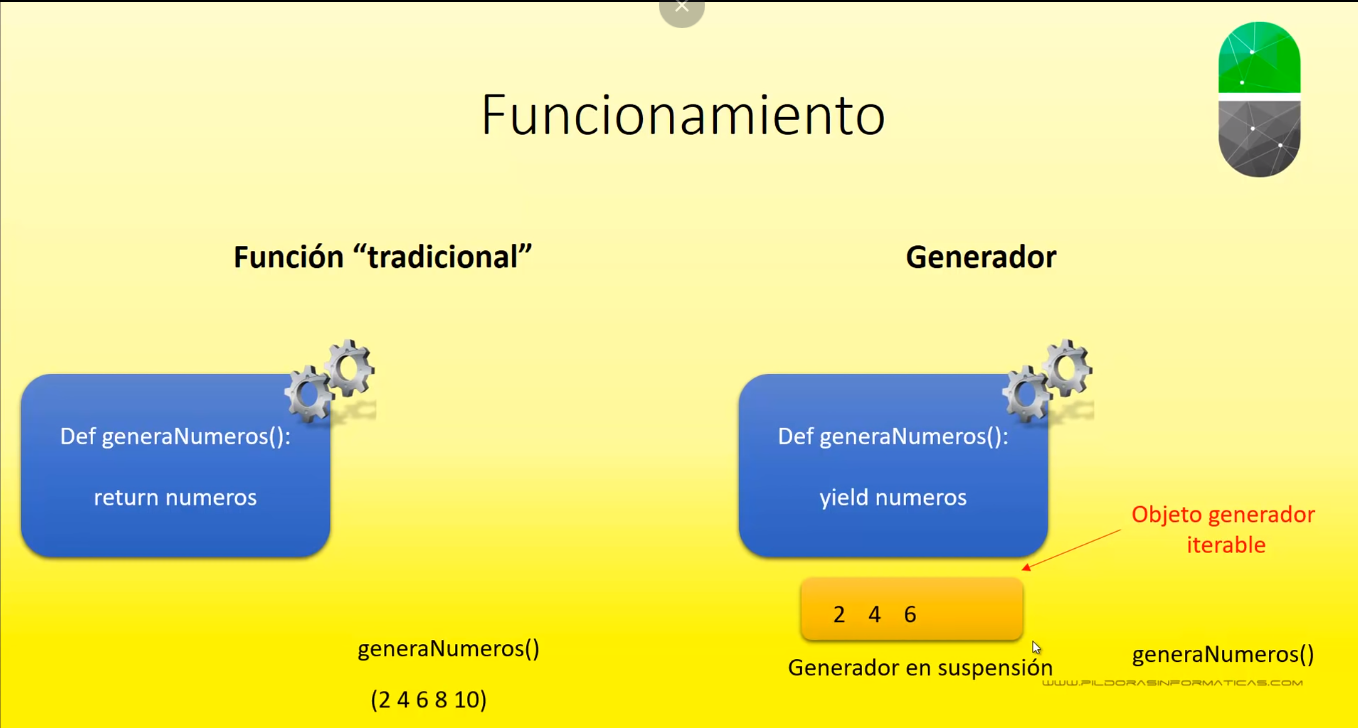
else:

arroba= False

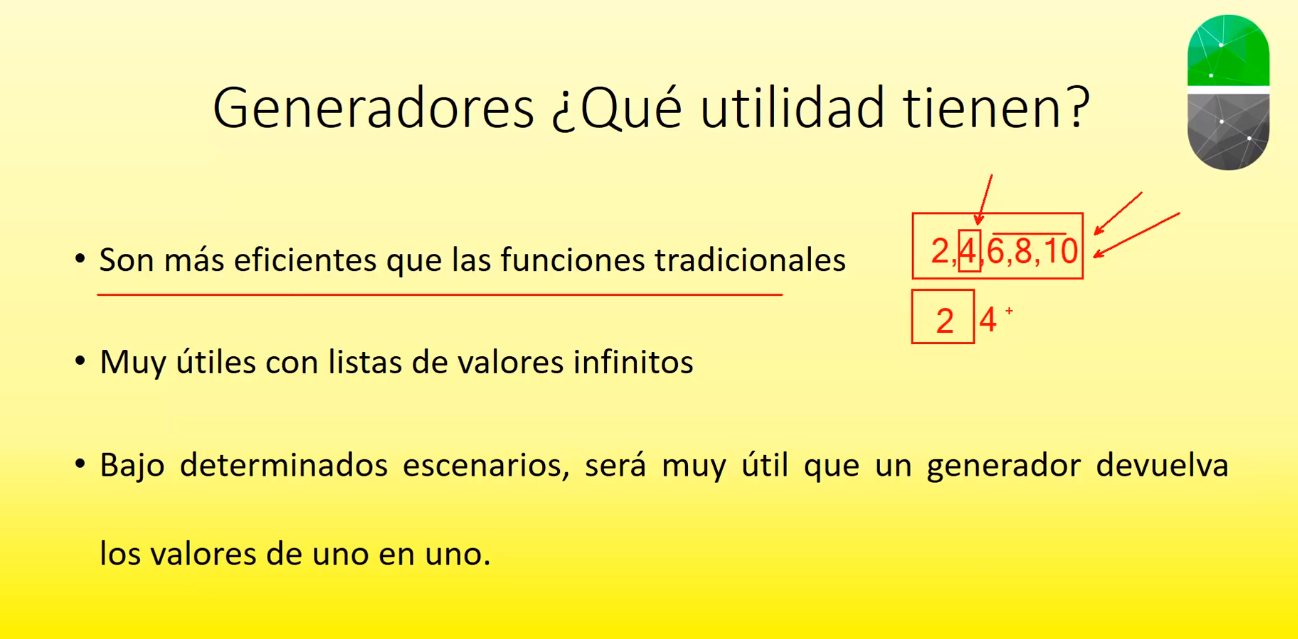
print(arroba) # cuando se realiza el break dentro del bucle este se sale del bucle y no entra al else

# Generadores





Como podemos observar en la imagen se puede hacer una comparación entre el generador y una *función* ya que son parecidos, pero la diferencia es que la función retorna una estructura completa en cambio el generador de vuelve un solo valor de la estructura que se está haciendo y lo devuelve como un objeto generador iterable. Devuelvo un elemento a la vez cada vez que se llama.



.

Sintaxis

Def generarNumeros():

Yeild números

Ejemplos

def generaPares(limite):

num = 1

miLista=[]

while num < limite:

miLista.append(num\*2)

num+=1

return miLista

print(generaPares(10))

## Ahora usando Generadores

def generaParess(limite):

num=1

while num < limite:

yield num\*2

num+=1

devuelvePares=generaParess(10)

#for i in devuelvePares:

# print(i)

print(next(devuelvePares))

print("Aquí podría ir más código"

print(next(devuelvePares))

print("Aquí podría ir más código")

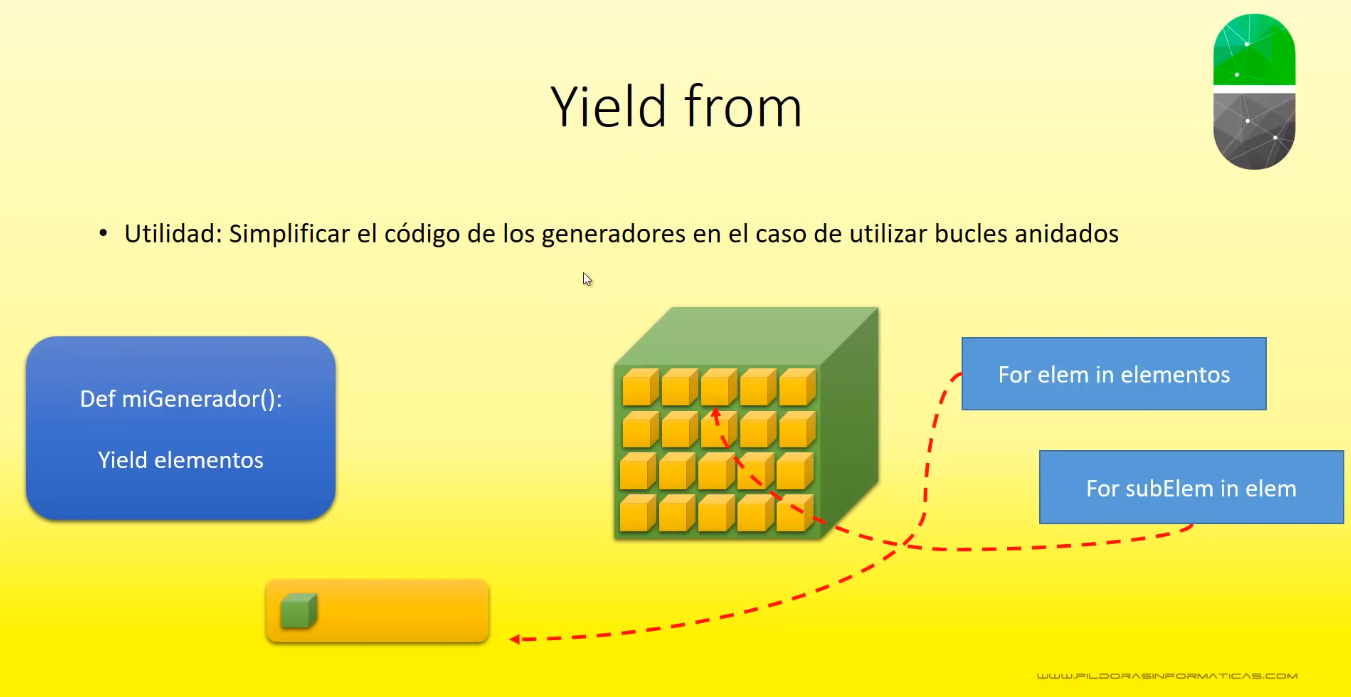
print(next(devuelvePares))

print("Aquí podría ir más código")

# Generadores 2

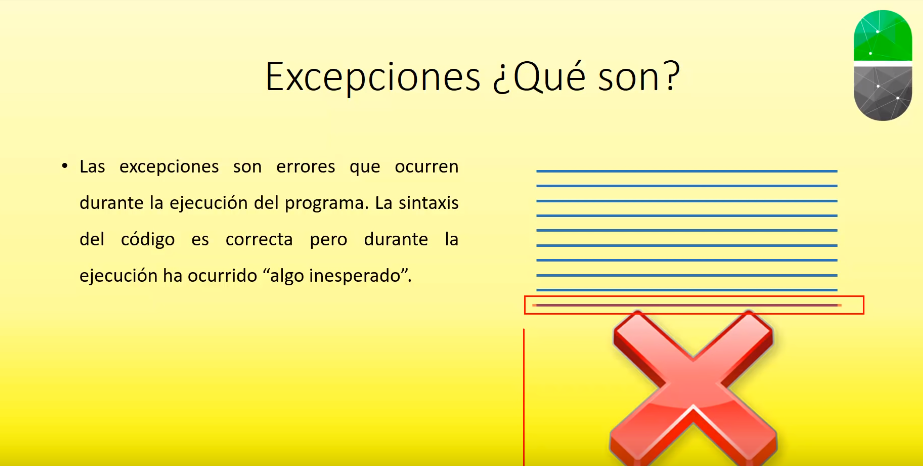
En este video seguiremos viendo generadores y ahora veremos la instrucción *yeild from*

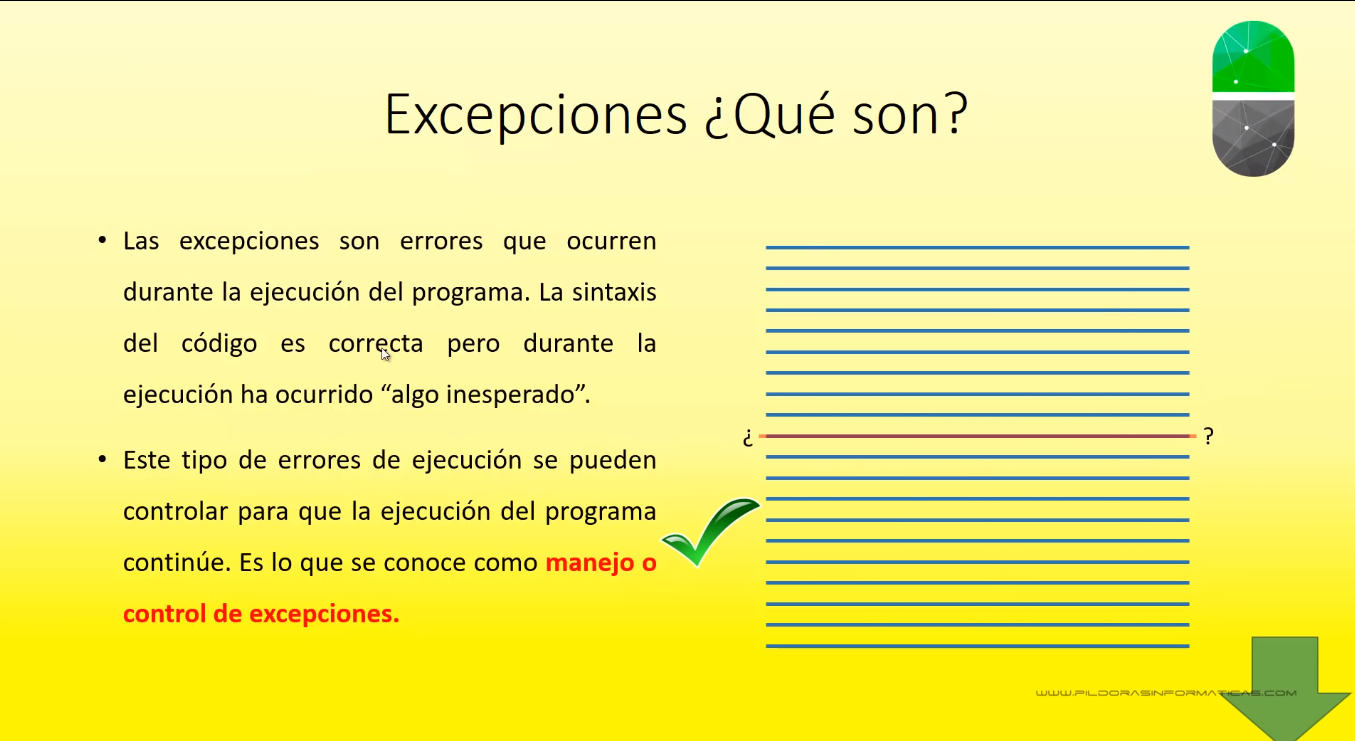
Una función Generador devuelve un objeto generador en el cual se van almacenando uno a uno poco a poco diversos elementos. Son de diferente tipo esos elementos

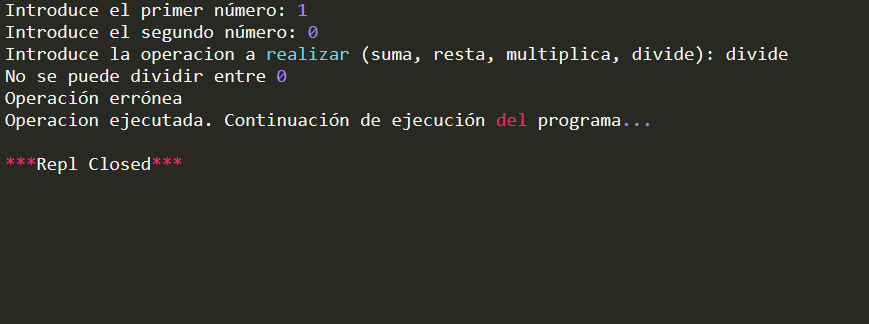


Esta instrucción es para tener acceso a las subelementos de cada uno de los elementos de un objeto generador, es decir suponiendo que de objeto generador sus elementos tienen subelementos entonces se quiere acceder a esas subelementos entonces se usa la instrucción *yeild from* para eso. El funcionamiento de esto es parecido al de una matriz en otros lenguajes de programación.

# Excepciones





Imagen que contiene captura de pantalla

Descripción generada automáticamente

# Excepciones 2

Capturas de varias excepciones

Imagen que contiene texto

Descripción generada automáticamente Cláusula finally

# Excepciones 3

Instrucción Raise

Creación de excepciones propias (más adelante en el curso cuando veamos la POO)

Seamos nosotros como programadores los que lancemos de forma intencionada la excepción, que seamos nosotros los que provoquemos una excepción cuando ocurra un error en la ejecución del programa. Que no sea el programa que lance la excepción sino nosotros de forma intencionada.

