

UNIVERSIDAD DE SAN BUENAVENTURA

ANALISIS Y DISEÑO DE ALGORITMOS

BITACORA CURSO PYTHON

LUIS FELIPE VELASCO TAO

15 DE MAYO

2020



Tabla de contenido

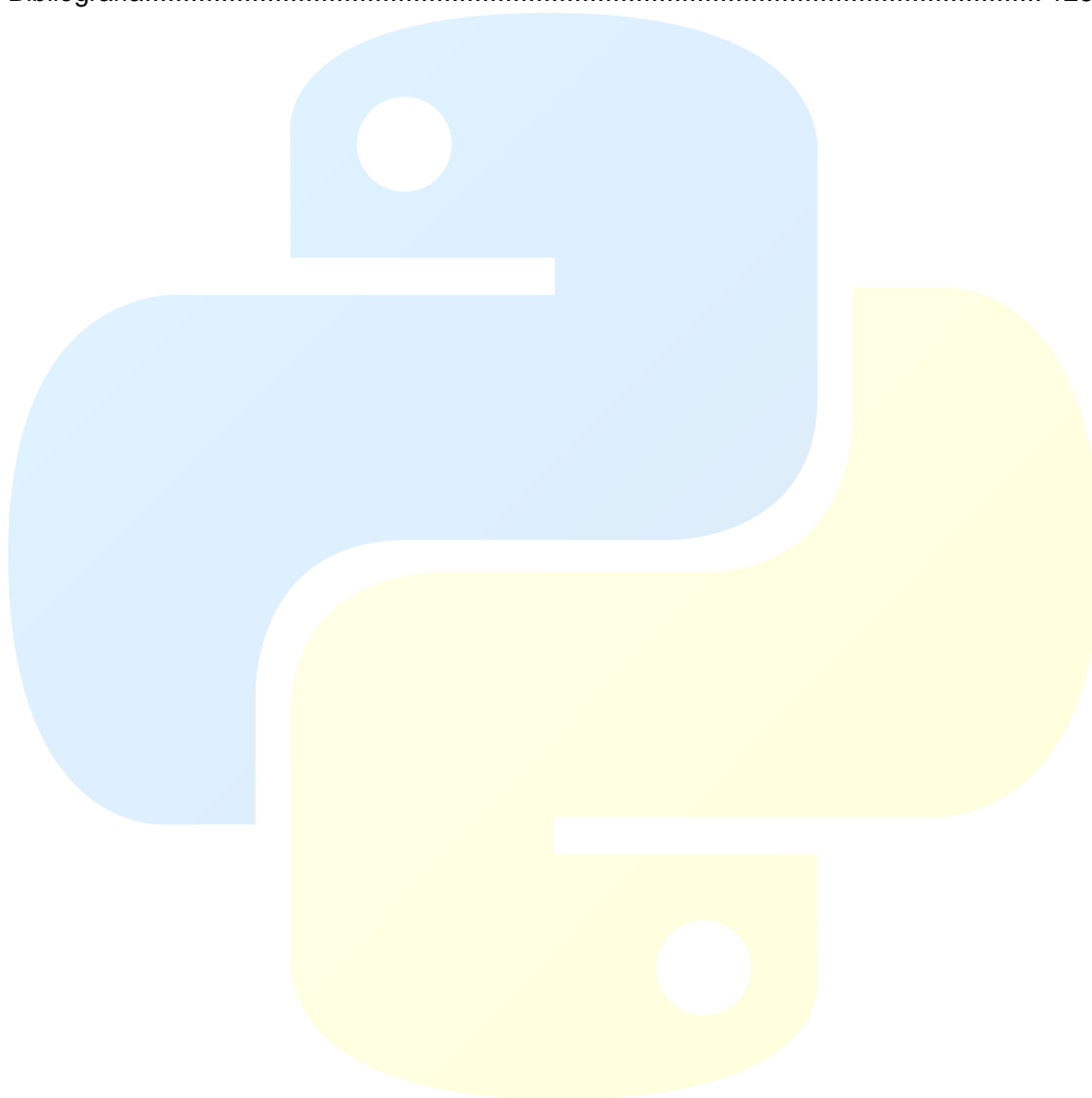
INTRODUCCIÓN.....	5
Video 1:.....	5
Video 2:.....	5
Video 3:.....	9
Video 4:.....	13
Video 5:.....	18
Video 6:.....	20
Video 7:.....	21
Video 8:.....	25
Video 9:.....	28
CONDICIONALES	30
Video 10:.....	30
Video 11:.....	32
Ejercicios basados en condicionales If, Else y Elif.....	36
Video 12:.....	38
Video 13:.....	40
BUCLES	42
Video 14:.....	42
Video 15:.....	43
Video 16:.....	46
Ejercicios del video 16:.....	47
Video 17:.....	49
Ejercicios del video 17:.....	50
Video 18:.....	51
GENERADORES.....	52
Video 19.....	52
Video 20:.....	55
EXCEPCIONES.....	57
Video 21:.....	57
Video 22:.....	61
Video 23:.....	62



PROGRAMACIÓN ORIENTADA A OBJETOS.....	64
Video 24:.....	64
Video 25:.....	68
1. Clase.....	68
2. Objeto	68
3. Instancia de clase	69
4. Modularización	69
5. Encapsulamiento	70
Video 26:.....	71
Video 27:.....	73
Video 28:.....	75
Video 29:.....	76
Video 30:.....	79
Video 31:.....	82
Video 32:.....	85
Video 33:.....	87
Ejercicio del video 33:	89
Módulos y paquetes.....	90
Video 34:.....	90
Video 35:.....	94
Paquete general	94
Subpaquete 1 – cálculos básicos.....	94
Subpaquete 2 – redondeo	95
Video 36:.....	96
Archivos externos	100
Video 37:.....	100
Video 38:.....	102
Video 39:.....	104
Video 40:.....	106
Video 41:.....	108
Interfaces graficas	110
Video 42:.....	110
Video 44:.....	116



Video 45:.....	118
Video 46:.....	121
Video 47 – 48 – 49	123
Video 50:.....	127
Bibliografía.....	128





CURSO PYTHON - MÓDULO 1

INTRODUCCIÓN

En esta sección del curso se presentará el lenguaje de programación a trabajar, su historia, entornos de desarrollo, acondicionamiento del entorno de desarrollo y primeros pasos en el lenguaje.

Video 1:

Se presentó el contenido temático del curso, se resolvieron preguntas típicas y se hizo una pequeña presentación de todos los requerimientos y actividades que se realizarán a lo largo del curso.

Video 2:

Se habla de la historia de Python:

Guido van Rossum ideó el lenguaje Python a finales de los 80 y comenzó a implementarlo en diciembre de 1989. En febrero de 1991 publicó la primera versión pública, la versión 0.9.0. La versión 1.0 se publicó en enero de 1994, la versión 2.0 se publicó en octubre de 2000 y la versión 3.0 se publicó en diciembre de 2008. Hasta 2018, el desarrollo de Python lo lleva a cabo un colectivo de programadores dirigido por Guido van Rossum y bajo el paraguas de la fundación Python Software Foundation. En julio de 2018 Guido van Rossum anunció que dejaría de dirigir el desarrollo de Python y a principios de 2019 se pondrá en marcha un consejo director de cinco miembros elegidos entre los desarrolladores de Python.



Ilustración 1 El logotipo de Python presenta a dos serpientes (las cuales se pueden pensar que son pitones) formando un símbolo del Yin y el Yang en colores azul y amarillo.



Ilustración 2 Fotografía del creador de Python, Guido Van Rossum, científico de la computación.



Se suele creer que el nombre de este lenguaje de programación fue tomado del de la serpiente pitón (Python en inglés), pero el nombre realmente hace referencia al gusto de Guido van Rossum por el grupo británico de comida llamado "Monty Python"

Luego se nos presentan las características de este lenguaje:

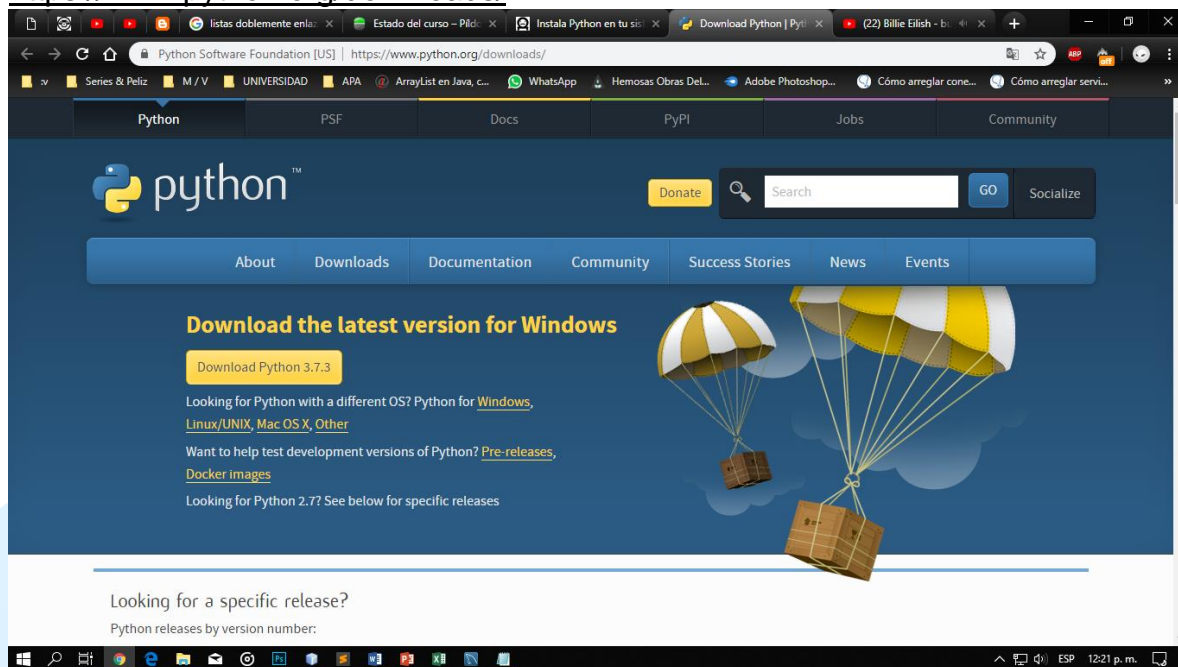
Python entre sus características guarda:

- Es un lenguaje de programación de alto nivel, lo que quiere decir que es fácil de comprender por cualquier persona sin importa que tenga conocimientos de programación previos o no, con una gramática sencilla, clara y muy legible. Como acotación, se menciona que, si se tiene conocimientos en inglés, esto facilitara la comprensión y producción de programas en este lenguaje.
- Este lenguaje es de tipado dinámico y fuerte. Al decir que es de tipado dinámico se habla de que los tipos de variables o atributos que se definan no deben ser especificador por un tipo de atributo sino por el valor que se le dé a cada atributo, y por otro lado, es de tipado fuerte, estas variables deben ser de cierto tipo, es decir, no se le puede dar a una variable de tipo **int** un valor de **String**, ya que al momento de la ejecución se lanzara un error relacionado con esto. Cabe destacar que el tipado también es fuerte debido a que, aunque este lenguaje prescinda de el uso de ciertos elementos como los punto y comas, aún conserva la necesidad de demarcar los espacios o tabulaciones para conservar la estructura deseada.
- Python es uno de los lenguajes orientado a objetos, esto quiere decir que nos otorga la posibilidad de crear programas con mayor grado de complejidad ligados al paradigma de la programación dirigida a objetos, y todos sus componentes.
- Python también hace parte del grupo **Open Source**, el cual se compone de software al cual, tanto usuarios como programadores pueden hacer uso de él y modificarlo a su conveniencia.
- Python posee una gran variedad de librerías las cuales brindan muchas posibilidades al programador, librerías como Pygame, la cual nos permite crear multimedia, HTTP, la cual nos ayuda con recursos web y su desarrollo en Python, o Pillow que nos posibilidades de uso y manipulación de imágenes.

Después se especifican los requerimientos para instalar Python en nuestros computadores:



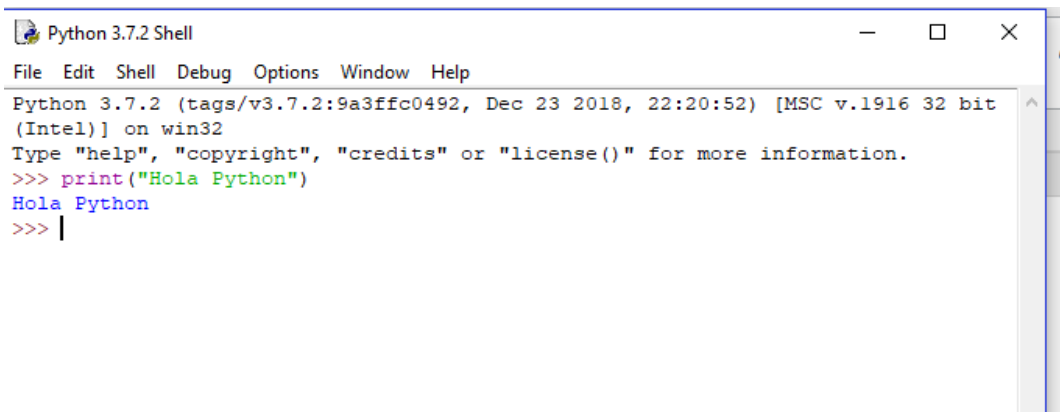
- Para instalar el IDLE de Python deben ir a la pagina <https://www.python.org/downloads/>



- La última versión de Python (3.7.3) puede ser utilizada en cualquier estructura de sistema (32 o 64bits) por lo cual se nos descarga un archivo .exe el cual nos ayudara la instalación de todo paquete.
- La instalación no tiene mucha ciencia, solo se deben seguir los pasos, y dar en siguiente y aceptar.
- Al finalizar encontraremos con:



- Abrimos IDLE (Python 3.7 32 o 64 bit) y nos encontraremos con el IDE de Python que trae por defecto:



```
Python 3.7.2 Shell
File Edit Shell Debug Options Window Help
Python 3.7.2 (tags/v3.7.2:9a3ffc0492, Dec 23 2018, 22:20:52) [MSC v.1916 32 bit
(Intel)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>> print("Hola Python")
Hola Python
>>> |
```

Esta última imagen muestra el IDLE (Entorno de desarrollo integrado y limitado) de Python, el cual no será el IDE que usaremos para aprender este lenguaje, debido a sus limitadas capacidades.



Video 3:

Se nos van mostrando a lo largo del video

- la función print() la cual nos sirve para mostrar datos por consola.
- Declaración de variables String.
- El uso del slash izquierdo (\) para dar saltos si se requiere en una línea de código.
- El uso del numeral (#) para poner comentarios en nuestro código
- El uso de un ciclo for para imprimir el valor de una variable incremental
- El uso de la clase range.

```
Primer codigo.py - C:\Users\FELIPE\Documents\PYTHON ACTIVITIES\Primer codigo.py (3.7.2)
File Edit Format Run Options Window Help
Python 3.7.2 (tags/v3.7.2:9a3ffc0492, Dec 23 2018, 22:20:52) [MSC v.1916 32 bit (Intel)]
Type "help", "copyright", "credits" or "license()" for more information.
>>> print("Hola Python")
Hola Python
>>> print("Hola Profesor");print("Hasta luego");
Hola Profesor
Hasta luego
>>> # Los comentarios se inician con numeral
>>> mi_nombre="Mi nombre es felipe"
>>> mi_nombre
'Mi nombre es felipe'
>>> mi_nombre:"Mi nombre es\
Felipe"
>>> #Por medio del uso del slash izquierdo(\) se puede hacer un salto en el String
>>> mi_nombre="mi nombre es\
Felipe"
>>> mi_nombre
'mi nombre esFelipe'
>>>#Se nos muestra como es un ciclo for y un abre bocas de la clase Range
>>> a= 0
>>> for i in range(5):
    a+=2
    print(a)

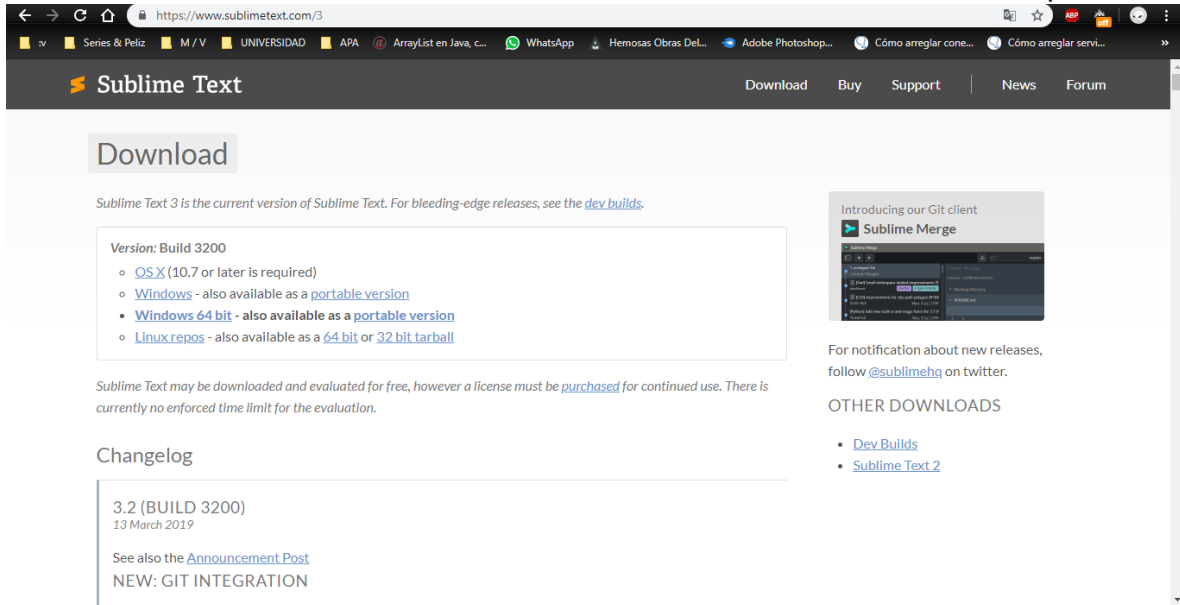
2
4
6
8
10
>>>
```

Luego probar el IDLE de Python procedimos a la instalación del editor de texto **Sublime Text 3**. Para ello, entramos a <https://www.sublimetext.com/3> en donde descargaremos la última versión de Sublime acorde a nuestro sistema operativo y

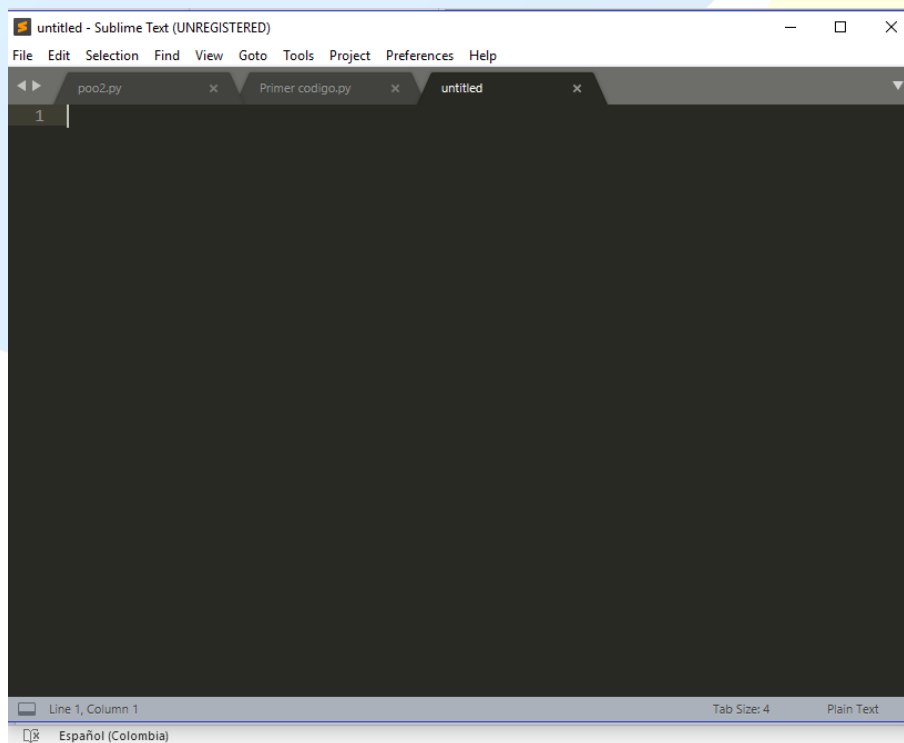


su

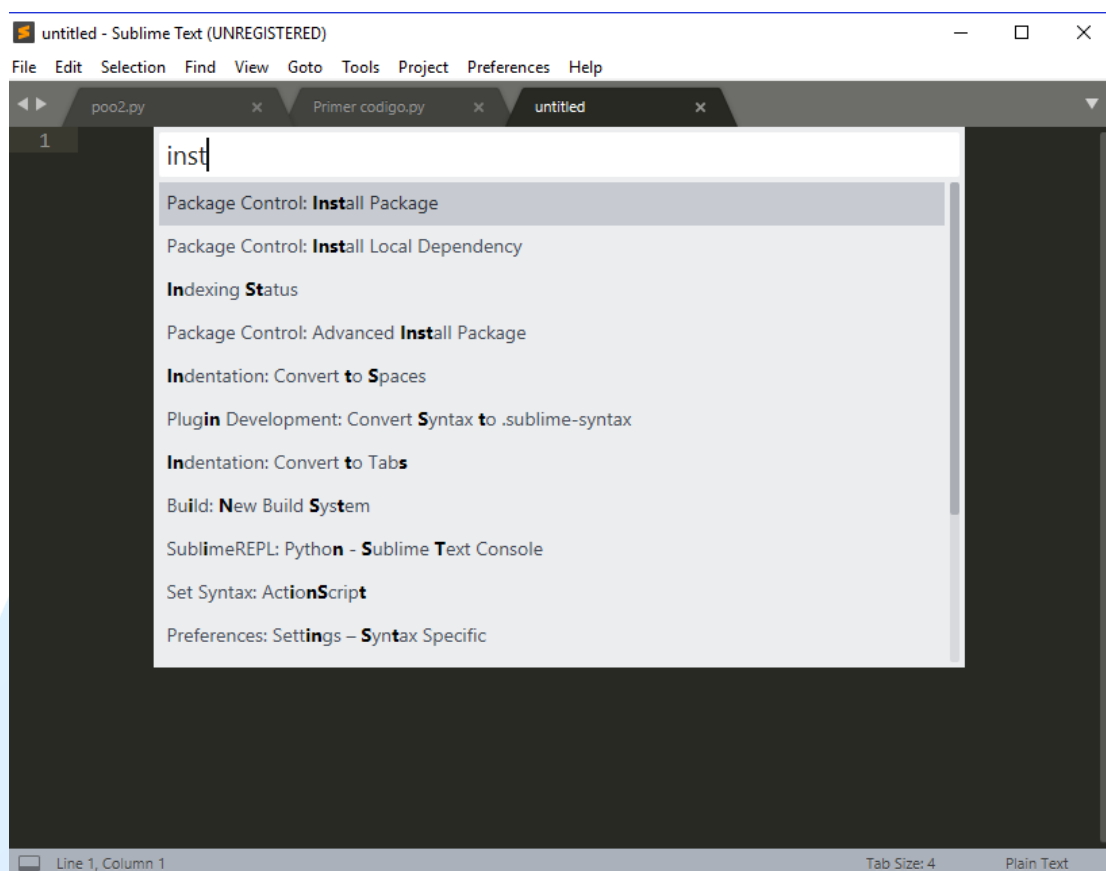
arquitectura:



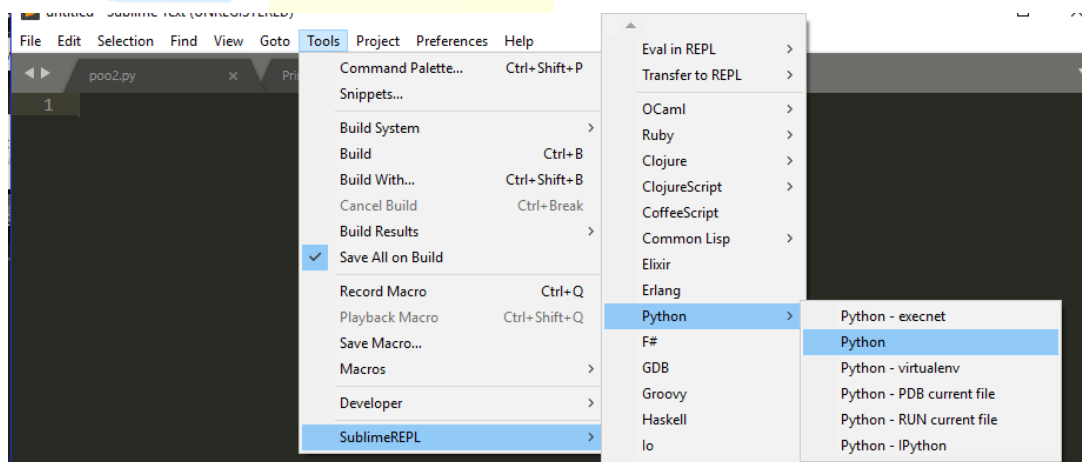
Después de descargar y seguir las instrucciones de instalación abriremos el panel principal de Sublime:



Luego debes realizar la instalación de un Paquete de Control para ello debemos ir a Tools / Command Pallete y escribimos "Install Package Control".



Después de haber instalado el paquete vamos a Tools / Sublime REPL / Python / Python y aquí nos encontraremos con algo muy similar a lo que encontramos en IDLE de Python. En este entorno podremos desarrollar código de la misma forma en que podíamos realizarlo en el IDLE de Python.

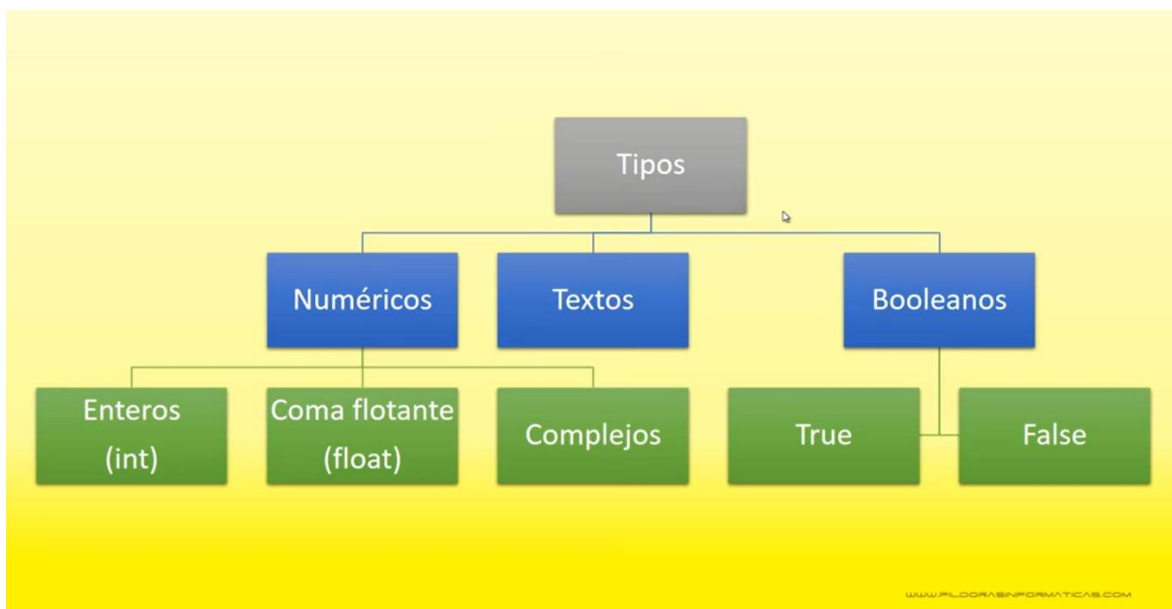




```
*REPL* [python] - Sublime Text (UNREGISTERED)
File Edit Selection Find View Goto Tools Project Preferences Help
poo2.py x Primer codigo.py x *REPL* [python] x
Python 3.7.2 (tags/v3.7.2:9a3ffc0492, Dec 23 2018, 22:20:52) [MSC v.1916 32 bit (Intel)] on
win32
Type "help", "copyright", "credits" or "license" for more information.
>>> print("Bienvido a Sublime")
Bienvido a Sublime
>>> |
```

**Video 4:**

En este video se nos presentan los tipos de datos y operadores en Python.

**Numéricos:**

- Int = Datos de tipo entero (ejm: 1,2,-300,100,etc)
- Float = (Coma flotante o decimal) : Datos de tipo decimal (ejm: 1.2000)
- Complejos = (estos datos no se utilizan en este curso) se utilizan para realizar cálculos matemáticos complejos.

Textos:

- Str = (String o cadena) Datos de tipo cadena de caracteres.

Booleanos:

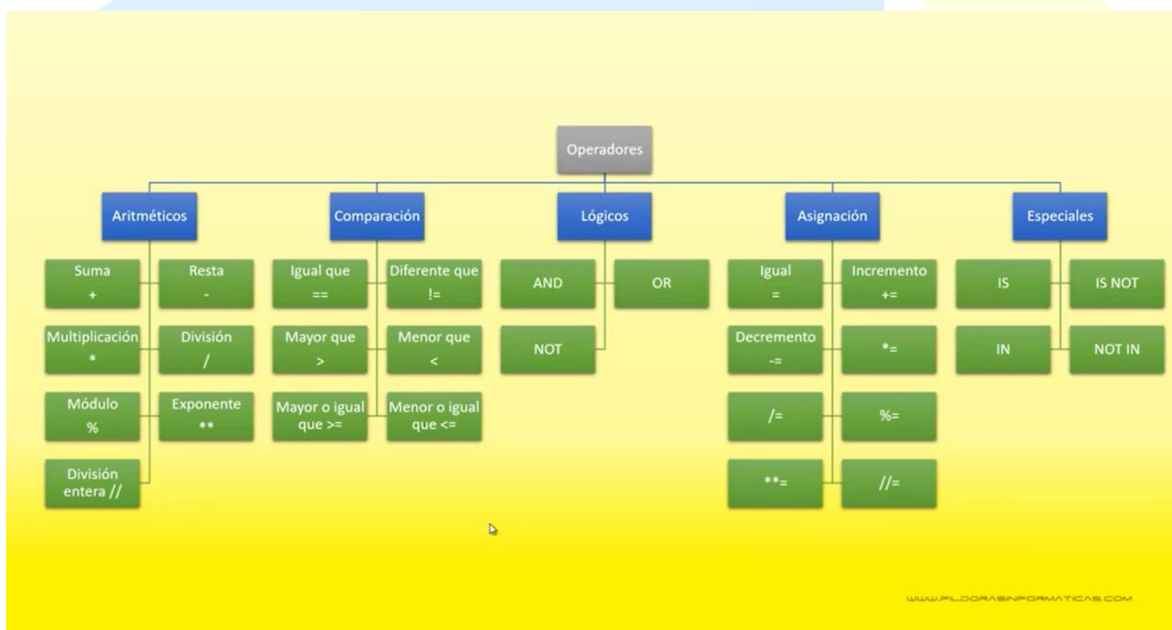
- Este tipo de dato solo puede tener dos valores (True o False).

Aquí se agrega una tabla con información adicional:



Tipo	Clase	Notas	Ejemplo
str	Cadena	Inmutable	'Cadena'
unicode	Cadena	Versión <u>Unicode</u> de str	u'Cadena'
list	Secuencia	Mutable, puede contener objetos de diversos tipos	[4.0, 'Cadena', True]
tuple	Secuencia	Inmutable, puede contener objetos de diversos tipos	(4.0, 'Cadena', True)
set	Conjunto	Mutable, sin orden, no contiene duplicados	set([4.0, 'Cadena', True])
frozenset	Conjunto	Inmutable, sin orden, no contiene duplicados	frozenset([4.0, 'Cadena', True])
dict	Mapping	Grupo de pares clave:valor	{'key1': 1.0, 'key2': False}
int	Número entero	Precisión fija, convertido en <i>long</i> en caso de overflow.	42
long	Número entero	Precisión arbitraria	42L ó 456966786151987643L
float	Número decimal	Coma flotante de doble precisión	3.1415927
bool	Booleano	Valor booleano verdadero o falso	True o False

Luego vemos un diagrama en donde vemos algunos de los tipos de operadores que podemos usar en Python:



Aritméticos:

- Suma: se utiliza el símbolo estándar de la suma de enteros y decimales (+)
- Resta: se utiliza el símbolo estándar de la resta de enteros y decimales (-)
- Multiplicación: Se utiliza el símbolo (*)
- División: Se utiliza el símbolo (/)
- Modulo: Recordar que el modulo nos ayuda a encontrar el residuo de una división. Se usa el símbolo porcentaje (%).
- Exponente: se utiliza el símbolo de la multiplicación dos veces (**).
- División entera: se utiliza el símbolo de la división dos veces (//).

**Comparativos:**

- Igual que: (==) se usa para comparar dos variables del mismo tipo.
- Diferente que: (!=) se usa para comparar dos variables del mismo tipo.
- Mayor que (>) = se usa para comparar variables de tipo numéricas
- Menor que (<) = se usa para comparar variables de tipo numéricas
- Mayor o igual que (>=) = se usa para comparar variables de tipo numéricas
- Menor o igual que (<=) = se usa para comparar variables de tipo numéricas

Lógicos: (Hay que recordar que estos se utilizan en condicionales If o Elif)

- AND: Si dos condiciones se cumplen el valor de estas será TRUE, si alguna de las dos no se cumple, el resultado será FALSE.
- OR: Si alguna de las dos o más condiciones se cumple el valor resultante será TRUE, de lo contrario, se retornará FALSE.
- NOT: Operador lógico de negación de una o un conjunto de condiciones (retorna el valor opuesto de la condición)

Asignación:

- Igualación o asignación (=): le da un valor a una variable. (Cualquier tipo de dato)
- Incremento: aquí encontramos a:
 - o += (más igual): sumara el valor de la variable con otro obtenido. (Valores numéricos y texto)
 - o *= (por igual): multiplicara el valor de la variable con otro obtenido. (Solo numéricos)
 - o **= (elevado igual): elevara el valor de la variable con otro obtenido. (Solo numéricos)
- Decremento: aquí encontramos
 - o -= (resta igual): restara el valor de la variable con otro obtenido. (Solo numéricos)
 - o /= (dividido igual): dividirá el valor de la variable con otro obtenido. (Solo numéricos)
 - o //= (división entera igual): realizara una división entera al valor de la variable con otro obtenido. (Solo numéricos)

Especiales:

- IS: se puede asociar como operador de igualación más preciso, muy similar a $\rightarrow x == y$.
- IS NOT: se puede asociar como operador de desigualdad más preciso, muy similar a $\rightarrow x != y$.



- IN: se puede asociar como el hecho de que una variable este contenida a otra.
- IN NOT: se puede asociar como el hecho de que una variable no este contenida a otra.

Realizamos unos cuantos ejemplos del uso de operadores aritméticos:

```
*REPL* [python] - Sublime Text (UNREGISTERED)
File Edit Selection Find View Goto Tools Project Preferences Help
poo2.py x primeras funciones.py *REPL* [python] x
Python 3.7.2 (tags/v3.7.2:9a3ffc0492, Dec 23 2018, 22:20:52) [MSC v.1916 32 bit (Intel)] on
win32
Type "help", "copyright", "credits" or "license" for more information.
>>> 5*6
30
>>> ##Multiplicación
... 10%3
1
>>> ##Modulo de la division
... 5**3
125
>>> ##Potencia
... 9//2
4
>>> ##Division entera
...
>>>
```

Luego se nos muestran ejemplos de cómo nombrar variables:

Se utiliza Snake Case para nombrar variables, es decir, al querer iniciar una nueva palabra, en vez de un espacio, usamos un guion bajo para separar palabras. Se pueden usar palabras que inicien en mayúscula o solo caracteres.

```
*REPL* [python] - Sublime Text (UNREGISTERED)
File Edit Selection Find View Goto Tools Project Preferences Help
poo2.py x primeras funciones.py *REPL* [python] x
Python 3.7.2 (tags/v3.7.2:9a3ffc0492, Dec 23 2018, 22:20:52) [MSC v.1916 32 bit (Intel)] on
win32
Type "help", "copyright", "credits" or "license" for more information.
>>> nombre="Felipe"
nombre_completo="Luis Felipe Velasco Tao"
nombre2 ="Luis"
print (nombre2,"\n",nombre,"\n",nombre_completo)
>>> >>> Luis
Felipe
Luis Felipe Velasco Tao
>>> |
```

Si queremos saber de qué tipo es una variable usamos la función “**type()**”:



```
>>> x = 55
>>> type(x)
<class 'int'>
>>> type(nombre)
<class 'str'>
>>> y = 0.15
>>> type(y)
<class 'float'>
>>>
```

Aquí podemos ver como el valor retornado es la clase a la que cada variable pertenece. Hay que recordar que las variables en Python son como objetos instanciados de cada clase.

Y finalmente se nos muestra cómo realizar saltos de línea en una cadena de texto y el uso de un condicional If y else.

```
Python 3.7.2 (tags/v3.7.2:9a3ffc0492, Dec 23 2018, 22:20:52) [MSC v.1916 32
win32
Type "help", "copyright", "credits" or "license" for more information.
>>> mensaje="""Esto
... Tiene
... Tres lineas"""
>>> print(mensaje)
Esto
Tiene
Tres lineas
>>> a = 10
>>> b = 1
>>> if a>b:
...     print("El numero 1 es mayor")
... else:
...     print("El numero 2 es mayor")
...
El numero 1 es mayor
>>>
```



Video 5:

En este video se nos presenta el concepto de función:

- Una función es un conjunto de líneas de código que funcionan como una unidad destinada a cumplir una tarea.
- Una función puede devolver valores
- Una función puede recibir valores de entrada (parámetros/argumentos).
- Cuando una función se encuentra dentro de una clase se le llama método.

En las diapositivas se nos muestra cómo se crea una función

Sintaxis

- `def nombre_función():`
 - Instrucciones de la función
 - `return` (opcional)
- `def nombre_función(parámetros)`
 - Instrucciones de la función
 - `return` (opcional)

Para llamar a una función debemos hacer solo lo siguiente

Ejecución

- `nombre_función()`
- `Nombre_función(parámetros)`

A continuación, se muestra un ejemplo de un método.



```
1
2  def repetir():
3
4      print("Estamos aqui")
5      print("Estamos lejos")
6      print("Estamos aprendiendo")
7      print("Estamos en el curso")
8
9  repetir()
10 print("////////")
11 repetir()
12 print("////////")
13 repetir()
14
```

```
Estamos aqui
Estamos lejos
Estamos aprendiendo
Estamos en el curso
////////
Estamos aqui
Estamos lejos
Estamos aprendiendo
Estamos en el curso
////////
Estamos aqui
Estamos lejos
Estamos aprendiendo
Estamos en el curso
[Finished in 0.4s]
```

No olvidar que al definir el nombre de la función y poner los dos puntos, debe estar correctamente indentado el contenido de esta, es decir debe tener una sangría lo cual indicara que las acciones que indicamos pertenecen a la función. Luego, solo debemos llamar la función con su nombre y los parámetros de entrada que esta tenga.



Video 6:

En este video se nos enseña a cómo crear una función con parámetros de entrada y valores retornados.

```
1 #Esta es una funcion sin valores de entrada ni retorno
2 def suma1():
3     num1=5
4     num2=7
5     print(num1+num2)
6
7 #Esta es una funcion que recibe valores de entrada pero no retorna ninguno
8 def suma2(num1,num2):
9     print(num1+num2)
10
11 # Esta es una funcion con valores de entrada y retorno
12 def suma3(num1,num2):
13     resultado=num1+num2
14     return resultado
15
16 #Llama de la funcion sin valores de entrada ni retorno
17 suma1()
18
19 #Llamada de la funcion con valores de entrada y sin retorno
20 suma2(20,10)
21
22 #Impresion de la funcion con valores de entrada y retorno
23 print(suma3(20,10))
24
```

Cuando creamos una función que recibe valores de entrada, recordar que deben ser del mismo tipo de los que se declararon adentro de la función.

```
Funcion sin valores de entrada ni retorno
12
Funcion solo con valores de entrada
30
Funcion de suma con valores de entrada y retorno 30
[Finished in 0.4s]
```

Los valores que se muestran en consola son estos (ejecutar código = primero guardar (control+s) y luego ejecutar(control+b))

Las funciones son muy útiles para la reutilización de líneas de código



Video 7:

En este video se nos muestra las distintas operaciones que podemos realizar con las listas.

Primero definimos que es una lista:

- Una lista es una colección de datos, muy similares a los arrays, pero con la particularidad de que las listas pueden almacenar distintos tipos de datos (Strings, int, float, boolean) además de que su tamaño es dinámico ya que se pueden añadir cuantos elementos queramos.
- La sintaxis de una lista es la siguiente:

```
nombreLista=[elem1, elem2, elem3.....]
```



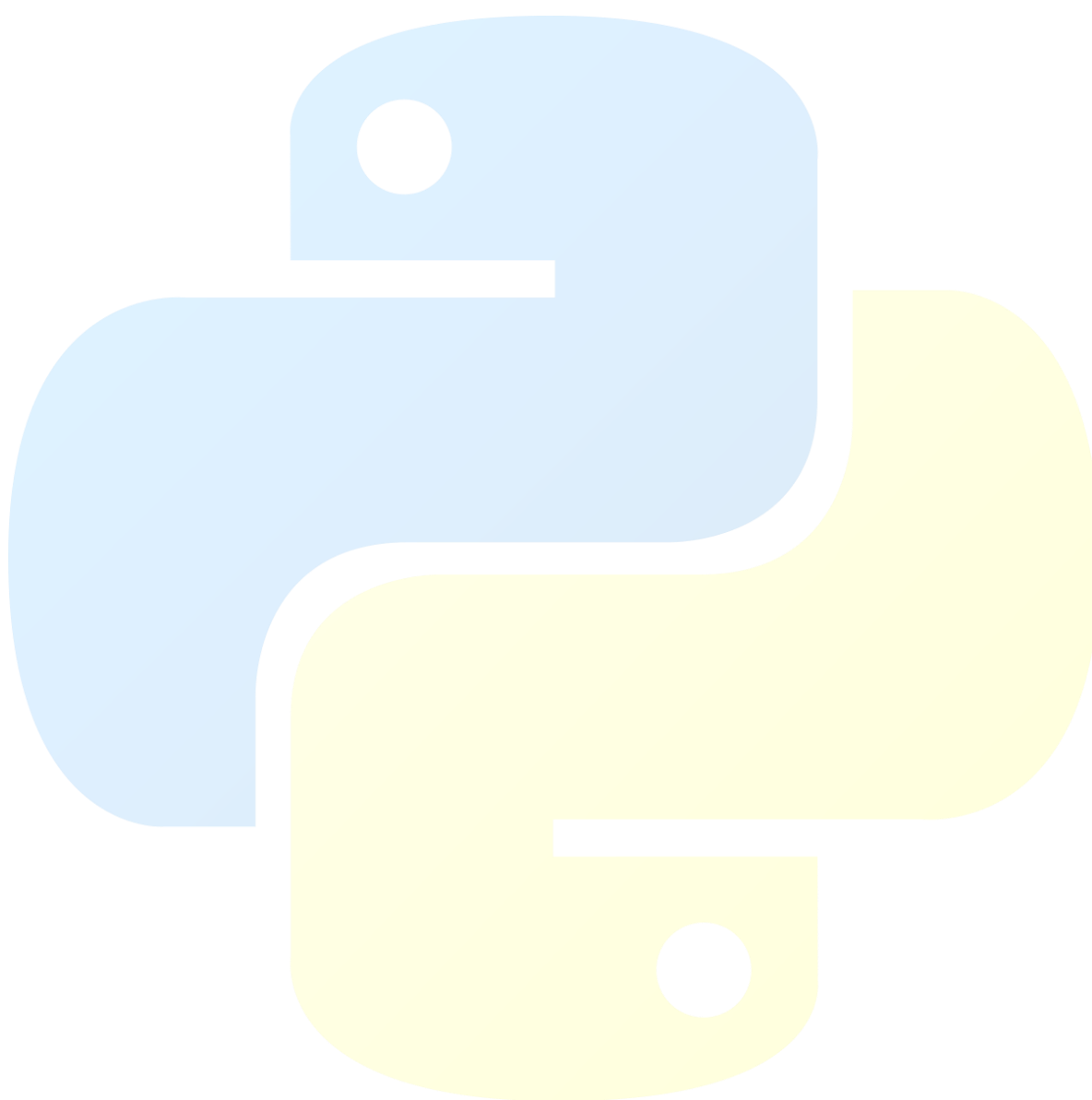
- Cada elemento de la lista tendrá un índice (Como lo vemos en la imagen anterior) al igual que cada uno de los elementos en un Array.



```
1 #inicializar una lista
2 milista=["Maria","Pepe","Marta","Yo"]
3 #Imprimir toda la lista
4 print(milista[:])
5 print("////////")
6 #Imprimir un elemento por medio de su indice
7 print(milista[2])
8 print("////////")
9 #Imprimir en un rango de indices
10 print(milista[:3])
11 print("////////")
12 print(milista[1:2])
13 print("////////")
14 print(milista[2:])
15 #Añadir un nuevo elemento al final
16 milista.append("mamá")
17 print(milista[:])
18 print("////////")
19 #Añadir un valor en cualquier posicion de la lista
20 milista.insert(2,"Kiara")
21 print(milista[:])
22 print("////////")
23 #Añadir varios elementos a la lista
24 milista.extend(["carmen","cecilia"])
25 print(milista[:])
26 print("////////")
27 #Saber la posicion de un elemento en la lista
28 print(milista.index("Kiara"))
29 print("////////")
30 #Saber si un elemento se encuentra en la lista
31 print("Kiara" in milista)
32 print("////////")
33 print("Marlen" in milista)
34 print("////////")
35 ##### lista con distintos elementos
36 listaV=[20,"Cama",True,5.15]
37 print(listaV[:])
38 print("////////")
39 ##Eliminar un elemento
40 milista.remove("Kiara")
41 print(milista[:])
42 print("////////")
43 ##Eliminar el ultimo elemento de la lista
44 milista.pop()
45 print(milista[:])
46 print("////////")
47 #
48 milista2=["Cama","Comida"]
49 ##Unir dos listas
50 milista3=milista+milista2
51 print(milista3[:])
52 print("////////")
53 ##Repetir una lista n veces
54 listaV=listaV*3
55 print(listaV[:])
56 print("////////")
```

EXPLICACIÓN CODIGO

- Lista llenada o quemada.
Lista=[x,y,z,...]
- Imprimir toda una lista
Print(lista[:])
- Imprimir un elemento
Print(lista[2])
- Imprimir rangos de una lista:
Desde el 0 hasta x
Print(lista[:2])
Entre un rango específico
Print(lista[1:3])
Desde un x hasta el final
Print(lista[2:])
- Añadir un elemento:
Lista.append("xxxxx")
- Añadir un elemento en una posición de la lista
Lista.insert(2,"YYYYY")
- Añadir varios elementos a la lista
Lista.extend(["AAAAA","BBBBB"])
- Ver la posición de un elemento en la lista
Lista.index("XXXXX")
- Saber si un elemento esta en la lista:
"XXXXXX" in lista
- Eliminar un elemento
Lista.remove("xxxx")
- Eliminar el ultimo elemento de la lista:
Lista.pop()
- Unir dos listas
Nlista=lista1+lista2
- Repetir una lista n veces
Lista=lista*n





Salida

```
['Maria', 'Pepe', 'Marta', 'Yo']  
/////////  
Marta  
/////////  
['Maria', 'Pepe', 'Marta']  
/////////  
['Pepe']  
/////////  
['Marta', 'Yo']  
['Maria', 'Pepe', 'Marta', 'Yo', 'mamá']  
/////////  
['Maria', 'Pepe', 'Kiara', 'Marta', 'Yo', 'mamá']  
/////////  
['Maria', 'Pepe', 'Kiara', 'Marta', 'Yo', 'mamá', 'carmen', 'cecilia']  
/////////  
2  
/////////  
True  
/////////  
False  
/////////  
[20, 'Cama', True, 5.15]  
/////////  
['Maria', 'Pepe', 'Marta', 'Yo', 'mamá', 'carmen', 'cecilia']  
/////////  
['Maria', 'Pepe', 'Marta', 'Yo', 'mamá', 'carmen']  
/////////  
['Maria', 'Pepe', 'Marta', 'Yo', 'mamá', 'carmen', 'Cama', 'Comida']  
/////////  
[20, 'Cama', True, 5.15, 20, 'Cama', True, 5.15, 20, 'Cama', True, 5.15]  
/////////
```




Video 8:

En este video se toca un término nuevo como lo son las tuplas.

Primero definimos que es una tupla:

- Una tupla es una lista inmutable, esto quiere decir que no pueden ser modificadas después de su creación, por lo cual no se permite añadir, eliminar o mover elementos (No permite el uso de *append*, *extend* o *remove*)
- Como actividades nos permite extraer alguna posición de ella, pero esto conlleva a la creación de una nueva tupla.
- Permiten corroborar si un elemento se encuentra o no en la tupla
- Como ventajas en comparación a las listas tiene que son más rápidas, consumen menos espacio debido a que no tendrán un tamaño variable sino estático, permiten formatear Strings y pueden ser utilizadas como claves en un diccionario, ya que las listas no lo permiten.

La sintaxis de una tupla es la siguiente:

```
nombreLista=(elem1, elem2, elem3.....)
```

[0]

[1]

[2]

La forma de marcar los índices en una tupla es la misma que en una lista, y recordar que las tuplas se contienen entre paréntesis (los cuales pueden ser opcionales), a diferencia de las listas que se contienen en llaves cuadradas.



CODIGO DE PRUEBA

```
1 tupla1=("Hola","Mi",25,"nombre",25)
2 print(tupla1[2])
3 print("////////")
4 #Convertir una tupla a lista
5 lista=list(tupla1)
6 print(lista)
7 print("////////")
8 ##Conveter una lista a tupla
9 tuplar=tuple(lista)
10 print(tuplar)
11 print("////////")
12 ## Buscar un valor en una tupla
13 print("Hola" in tupla1)
14 print("////////")
15 ##Saber cuantas veces ets aun elemento en una lista
16 print(tupla1.count(25))
17 print("////////")
18 ##Saber la longitud de una tupla
19 print(len(tupla1))
20 print("////////")
21 #Tupla unitaria
22 tuplaunitaria=(258,)
23 print(tuplaunitaria)
24 print(len(tuplaunitaria))
25 print("////////")
26 # Tupla sin parentesis
27 tuplan = 10,20,30,40
28 print(tuplan)
29 print("////////")
30 #Distribucion de valores de una tupla en variables
31 nombre,dia,mes,agnio,cosa=tupla1
32 print(nombre)
33 print(dia)
34 print(mes)
```



Salida

```
Elemento en la posicion 2 : 25
////////
Tupla convertida en lista
['Hola', 'Mi', 25, 'nombre', 25]
////////
Lista convertida en tupla
('Hola', 'Mi', 25, 'nombre', 25)
////////
True
////////
Cuantas veces esta 25 en la tupla? 2
////////
Longitud de la tupla : 5
////////
Tupla de un solo elemento (258,)
1
////////
Tupla sin parentesis : (10, 20, 30, 40)
////////
Hola
Mi
25
```

Nota: en las últimas versiones de Python si se permite el uso de la función index en las tuplas para saber la ubicación de u elemento.



Video 9:

En este video se presenta otra estructura no conocida y por lo visto propia este lenguaje como lo son los diccionarios.

Primero se define que es un diccionario:

- Los diccionarios son estructuras de datos en las cuales se pueden almacenar datos de distintos tipos (int, float, String o boolean) hasta tuplas u otros diccionarios.
- En los diccionarios, cada uno de los elementos que lo componen tienen una clase de relación los cuales los identifica. Esta relación es de tipo **clave: dato**. Esa clave es un identificador para cada elemento.
- En los diccionarios no se tiene en cuenta el orden de los elementos.

Código

```
1  dicc1={"Alemania":"Berlin","Francia":"Paris","UK":"Londres","España":"Madrid"}
2  #Acceder al valor de una clave en el Dicc
3  print(dicc1["Francia"])
4  print("////////")
5  #
6  print("Diccionario 1\n",dicc1)
7  print("////////")
8  #Añadir
9  dicc1["Italia"]="Peru"
10 print("Diccionario 1\n",dicc1)
11 print("////////")
12 #Cambiar valor de una clave
13 dicc1["Italia"]="Roma"
14 print("Diccionario 1\n",dicc1)
15 print("////////")
16 #Eliminar elemento
17 del dicc1["UK"]
18 print("Diccionario 1\n",dicc1)
19 print("////////")
20 ##### Diccionario de elementos variados
21 dicc2={"Alemania":"Berlin","Jordan":23,"¿Mayor de 18?":True}
22 print("Diccionario de elementos variados:\n",dicc2)
23 tupla=["España","Italia","Francia","UK"]
24 ## Diccionario compuesto por claves de una tupla
25 dicc3={tupla[0]:"Madrid",tupla[1]:"Roma",tupla[2]:"Francia",tupla[3]:"Londres"}
26 print("Diccionario con claves de una tupla:\n",dicc3)
27 ## Diccionario que almacene una tupla
28 europa=tupla
29 asia=["China","Japon","Corea del norte","Indonesia"]
30 dicc4={"Europa":europa,"Asia":asia}
31 print("Diccionario compuesto de tuplas\n",dicc4)
```



Salida

```
Paris
/////
Diccionario 1
{'Alemania': 'Berlin', 'Francia': 'Paris', 'UK': 'Londres', 'España': 'Madrid'}
/////
Diccionario 1
{'Alemania': 'Berlin', 'Francia': 'Paris', 'UK': 'Londres', 'España': 'Madrid', 'Italia': 'Peru'}
/////
Diccionario 1
{'Alemania': 'Berlin', 'Francia': 'Paris', 'UK': 'Londres', 'España': 'Madrid', 'Italia': 'Roma'}
/////
Diccionario 1
{'Alemania': 'Berlin', 'Francia': 'Paris', 'España': 'Madrid', 'Italia': 'Roma'}
/////
Diccionario de elementos variados:
{'Alemania': 'Berlin', 'Jordan': 23, '¿Mayor de 18?': True}
Diccionario con claves de una tupla:
{'España': 'Madrid', 'Italia': 'Roma', 'Francia': 'Francia', 'UK': 'Londres'}
Diccionario compuesto de tuplas
{'Europa': ['España', 'Italia', 'Francia', 'UK'], 'Asia': ['China', 'Japon', 'Corea del norte', 'Indonesia']}
>>>|
```



CONDICIONALES

En esta sección del curso se habla del uso de los condicionales y sentencias condicionadoras del flujo en el código del programa mediante el uso de condiciones de valor booleano.

Video 10:

En este video se presenta el condicional básico por el cual podemos evaluar de verdad de una sentencia, este condicional es If.

Se define a un condicional if como:

- Un bloque condicional el cual evaluará el valor de verdad de una sentencia (se puede tomar esto como una pregunta la cual solo tendrá como respuesta un sí o un no), y si su valor es verdadero, se ejecutará el bloque de instrucciones que este en medio de él.
- Recordar la importancia del uso de sangría, ya que esta servirá para poder entender que acciones se cumplirán al momento de que la sentencia se cumpla.
- Este condicional posibilita el acceso a código el cual solo se puede hacer bajo distintas condiciones.
- El uso del condicional If se usará también como la estructura *Switch -case* en java, ya que en Python no se posee una estructura similar a esta.

Código

```
1 print("1. Programa de evaluacion de alumnos")
2 print("   ingrese su nota")
3
4 nota=input()
5
6 def evaluacion (nota):
7     valoracion="Aprobado"
8     if nota<5:
9         #Si la nota es menor a 5, la variable 'Valoracion' cambiara a "Suspenso"
10        valoracion="Suspenso"
11    return valoracion
12
13 print(evaluacion(int(nota)))
14
15 print("2. Programa de evaluacion de alumnos")
16 print("   ingrese su nota")
17 nota=input()
18
19 print(evaluacion(int(nota)))
```

En este bloque de código, se encapsula la sentencia if en un método, el cual evaluara si la nota obtenida por un estudiante es suficiente para aprobar o suspender su materia.



Se llaman dos veces el código para poder ver cómo actúa el condicional If cuando la sentencia es verdadera o fals

Salida

```
1. Programa de evaluacion de alumnos
ingrese su nota
5
La materia esta : Aprobada
2. Programa de evaluacion de alumnos
ingrese su nota
1
La materia esta : Suspendida
>>>
```

**Video 11:**

En este video seguimos utilizando el condicional If, haciendo uso en ella de otro condicional interno en el como lo es Else y Elif.

- Un Else en una sentencia If nos ayudara en resumen a realizar un bloque de acciones en el caso de que la sentencia evaluada en el If no resulte ser verdad.
- Else = "Si no entonces..."

Código

```
def edad():  
    print ("Verificar edad")  
    edad=int(input("Introduce tu edad: "))  
    if edad < 18:  
        print ("No puedes pasar - Eres menor de edad")  
    else :  
        print("Puedes pasar - eres mayor de edad")  
    print("//////////")  
edad()  
edad()
```

Salida

```
Verificar edad  
Introduce tu edad: 12  
No puedes pasar - Eres menor de edad  
//////////  
Verificar edad  
Introduce tu edad: 19  
Puedes pasar - eres mayor de edad  
//////////
```

- Por otro lado, si la sentencia no es verdad, pero queremos evaluar si la variable evaluada inicialmente cumple otra condición, podemos hacer uso de un Elif.
- **Elif** puede ser asimilado como un Else if o un "Entonces si..."
- Por medio del uso de Elif puede crear una estructura similar a un Switch – case. Creando así un menú que evaluara si la sentencia cumple con la condición para luego ejecutar o no el bloque de código que contenga.



Código

```
13 def nota():
14     print ("Verificar nota")
15     nota=int(input("Introduce tu nota: "))
16     ## If usado para simular un Switch - case
17     if nota<5:
18         print("Insuficiente")
19
20     elif nota<6:
21         print("Suficiente")
22
23     elif nota<7:
24         print("Bien")
25
26     elif nota<8:
27         print("notable")
28
29     else:
30         print("Sobresaliente")
31     print("//////////")
32     nota()
33     nota()
```

Nota: Utilizo métodos para poder evaluar de todas las formas posibles el comportamiento del condicional dependiendo de cómo la sentencia puede cobrar o no un valor de verdad.



Salida

```
Verificar nota
Introduce tu nota: 3
Insuficiente
//////////
Verificar nota
Introduce tu nota: 5
Suficiente
//////////
Verificar nota
Introduce tu nota: 6
Bien
//////////
Verificar nota
Introduce tu nota: 7
notable
//////////
Verificar nota
Introduce tu nota: 9
Sobresaliente
//////////
```

A continuación, se presenta el uso del condicional Elif para reestructurar el ejemplo de la edad.

Código



```
1 def edad():
2     print ("Verificar edad")
3     edad=int(input("Introduce tu edad: "))
4
5     if edad >100 or edad <0:
6         print("Edad incorrecta")
7     elif edad < 18:
8         print ("No puedes pasar - Eres menor de edad")
9     else :
10        print("Puedes pasar - eres mayor de edad")
11    print("//////////")
12    edad()
13    edad()
14    edad()
15    edad()
16
```

Salida

```
Verificar edad
Introduce tu edad: -5
Edad incorrecta
//////////
Verificar edad
Introduce tu edad: 110
Edad incorrecta
//////////
Verificar edad
Introduce tu edad: 5
No puedes pasar - Eres menor de edad
//////////
Verificar edad
Introduce tu edad: 18
Puedes pasar - eres mayor de edad
//////////
```



Ejercicios basados en condicionales If, Else y Elif

1. Crea un programa que pida dos números por teclado. El programa tendrá una función llamada "DevuelveMax" encargada de devolver el número más alto de los dos introducidos.

Solución:

```
1 def devuelve():
2     print("PROGRAMA PARA COMPARAR DOS NUMEROS")
3     n1=int(input("Ingrese el primer numero: "))
4     n2=int(input("Ingrese el segundo numero: "))
5     if n1>n2:
6         print(n1," es mayor que ",n2,"\n")
7     else :
8         print(n2," es mayor que ",n1,"\n")
9     devuelve()
10
```

Salida:

```
PROGRAMA PARA COMPARAR DOS NUMEROS
Ingrese el primer numero: 5
Ingrese el segundo numero: 10
10 es mayor que 5
```

2. Crea un programa que pida por teclado "Nombre", "Dirección" y "Telefono". Esos tres datos deberán ser almacenados en una lista y mostrar en consola el mensaje: "Los datos personales son: nombre apellido teléfono" (Se mostrarán los datos introducidos por teclado).

Solución:

```
11 #Por diccionario
12 def listaDatos():
13     datos={}
14     datos["Nombre"] = input("Ingrese su nombre: ")
15     datos["Direccion"] = input("Ingrese su direccion: ")
16     datos["Telefono"] = input("Ingrese su telefono: ")
17     print("Sus datos son ",datos,"\n")
18
19 listaDatos()
20 #Por lista
21 def listaDatos2():
22     nombre=input("Ingrese su nombre: ")
23     direccion=input("Ingrese su direccion: ")
24     telefono=input("Ingrese su telefono: ")
25     datos=[nombre,direccion,telefono]
26     print("Sus datos son: \nnombre: ",datos[0],"\ndireccion: ",datos[1],"\ntelefono: ",datos[2],"\n")
27
28 listaDatos2()
```

Salida:



```
PROGRAMA PARA ALMACENAR DATOS 1
Ingrese su nombre: Luis Felipe
Ingrese su direccion: Cra 6 A Este N 36 - 88
Ingrese su telefono: 3144432469
Sus datos son {'Nombre': 'Luis Felipe', 'Direccion': 'Cra 6 A Este N 36 - 88', 'Telefono': '3144432469'}

PROGRAMA PARA ALMACENAR DATOS 2
Ingrese su nombre: Luis Felipe
Ingrese su direccion: Ca 6 A Este N. 38 - 88
Ingrese su telefono: 3144432469
Sus datos son:
nombre: Luis Felipe
direccion: Ca 6 A Este N. 38 - 88
telefono: 3144432469
```

3. Crea un programa que pida tres números por teclado. El programa imprime en consola la media aritmética de los números introducidos.

Solución:

```
31
32 def mediaA():
33
34     n1=int(input("Ingrese el primer número: "))
35     n2=int(input("Ingrese el segundo número: "))
36     n3=int(input("Ingrese el tercer número: "))
37     print("La media arimetica de ",n1," ",n2," y ",n3," es",((n1+n2+n3)/3),"\n")
38
39 mediaA()
```

Salida:

```
Ingrese el primer número: 5
Ingrese el segundo número: 17
Ingrese el tercer número: 14
La media arimetica de 5 , 17 y 14 es 12.0
```

**Video 12:**

En este video no se hace mayor cosa que el uso de sentencias concatenadas, es decir, por las cuales se deben seguir cada una de las que se encuentren unidas. Al llegar el caso de que al ir revisando cada sentencia y que alguna resulte ser false, la sentencia concatenada dará false.

Esto se puede ver en los siguientes ejemplos:

Código

```
1 def edad():
2     print(" Evaluar edad ")
3     edad = int(input("Ingrese su edad : "))
4     if 0<edad<100:
5         print("La edad es correcta\n")
6     else:
7         print("Edad incorrecta\n")
8     edad()
9     edad()
10    edad()
11
```

Salida

```
Evaluar edad
Ingrese su edad : 5
La edad es correcta

Evaluar edad
Ingrese su edad : 10
La edad es correcta

Evaluar edad
Ingrese su edad : 15
La edad es correcta
```

En el siguiente programa se ve ya como se evalúa un hilo de sentencias las cuales, al solo una de estas resultar ser falsas se lanzará lo contenido en el bloque Else.



Código

```
11
12 def salarios ():
13     salariopr=int(input("Ingrese el salario del presidente: $"))
14     print("Salario Presidente: ",salariopr)
15     salariod=int(input("Ingrese el salario del director: $"))
16     print("Salario Presidente: ",salariod)
17     salariopja=int(input("Ingrese el salario del jefe de area: $"))
18     print("Salario Presidente: ",salariopja)
19     salarioad=int(input("Ingrese el salariod el administrador: $"))
20     print("Salario Presidente: ",salarioad)
21
22     if salarioad<salariopja<salariod<salariopr:
23         print("Todo esta bien\n")
24     else:
25         print("Los salarios no son correctos\n")
26
27
28 salarios()
```

Salida

```
Ingrese el salario del presidente: $50000
Salario Presidente: 50000
Ingrese el salario del director: $40000
Salario Presidente: 40000
Ingrese el salario del jefe de area: $30000
Salario Presidente: 30000
Ingrese el salariod el administrador: $20000
Salario Presidente: 20000
Todo esta bien

Ingrese el salario del presidente: $10000
Salario Presidente: 10000
Ingrese el salario del director: $20000
Salario Presidente: 20000
Ingrese el salario del jefe de area: $50000
Salario Presidente: 50000
Ingrese el salariod el administrador: $30000
Salario Presidente: 30000
Los salarios no son correctos
```



Video 13:

En este video se explica el uso de los operadores lógicos AND y OR y el operador especial IN. Estos operadores ya fueron explicados previamente.

En resumen, el operador AND se utiliza para crear una sentencia compuesta por otras dos las cuales, solo darán True si y solo si las dos son verdaderas, de lo contrario se retronará un False. Por otro lado, el operador lógico OR presenta flexibilidad, ya que permite que alguna de las sentencias al menos sea verdad y así poder ejecutar lo contenido en el bloque lf.

En el ejercicio previamente se utilizó para ver como el comportamiento del condicional puede variar.

Código

```
1
2 def ejercicio1():
3     print("PROGRAMA DE BECAS")
4     distancia=float(input("Ingrese distancia a la que vive de la escuela: "))
5     nhermanos=int(input("Ingrese el numero de hermanos: "))
6     sfamiliar=int(input("Ingrese el salario familiar: $"))
7     print("Distancia: ",distancia,"km.\nNº hermanos: ",nhermanos,"\nSalario familiar: $",sfamiliar)
8     #if distancia>40 and nhermanos>2 and sfamiliar<=20000: Las tres deben ser TRUE
9     if distancia>40 or nhermanos>2 or sfamiliar<=20000:
10        print("El estudiante ha ganado una beca")
11    else:
12        print("El estudiante no ha ganado una beca")
13
14 ejercicio1()
15
```

Salida

```
PROGRAMA DE BECAS
Ingrese distancia a la que vive de la escuela: 10
Ingrese el numero de hermanos: 5
Ingrese el salario familiar: $1000000
Distancia: 10.0 km.
Nº hermanos: 5
Salario familiar: $ 1000000

El estudiante ha ganado una beca

PROGRAMA DE BECAS
Ingrese distancia a la que vive de la escuela: 5
Ingrese el numero de hermanos: 1
Ingrese el salario familiar: $200000
Distancia: 5.0 km.
Nº hermanos: 1
Salario familiar: $ 200000

El estudiante no ha ganado una beca
```

Finalmente se hace uso del operador IN, el cual evalúa si un valor se encuentra contenido o no dentro de otro.



Código

```
18
19 def ejercicio2():
20     print("Asignaturas: \n - Informatica Grafica\n - Pruebas de software\n - Usabilidad y accesibilidad")
21     opc=input("Ingrese la asignatura escogida.")
22     asig=opc.lower()
23     if asig in ("Informatica Grafica","Pruebas de software","Usabilidad y accesibilidad"):
24         print("Asignatura elegida: ",asig)
25     else:
26         print(asig,"No esta en la lista de asignaturas")
27
28
29 ejercicio2()
30
```

Salida

```
Asignaturas:
- Informatica Grafica
- Pruebas de software
- Usabilidad y accesibilidad
Ingrese la asignatura escogida.Usabilidad y accesibilidad
Asignatura elegida: usabilidad y accesibilidad
Asignaturas:
- Informatica Grafica
- Pruebas de software
- Usabilidad y accesibilidad
Ingrese la asignatura escogida.Ninguna
ninguna No esta en la lista de asignaturas
```



BUCLES

En esta sección del curso se explicará todo lo relacionado con los bucles, su estructura, implementación en programas y ejercicios prácticos que ayuden a su aprendizaje.

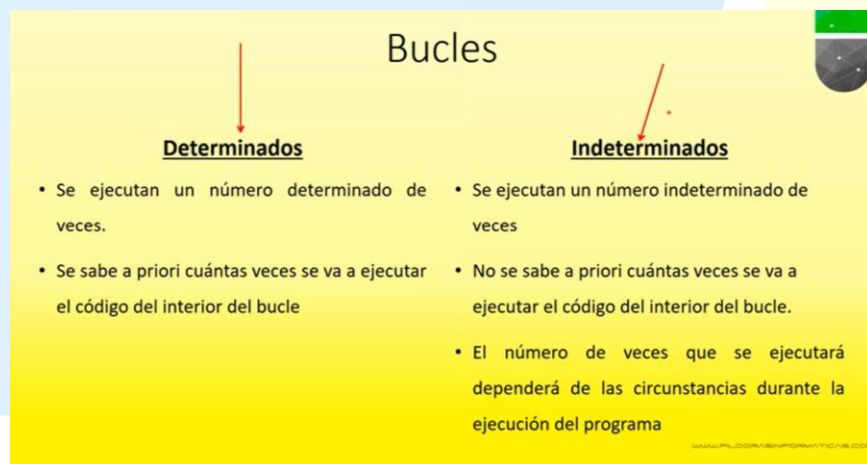
Video 14:

En este video se nos explica que es un bucle:

Un bucle es un bloque de código encapsulado para que por medio de una estructura de flujo se pueda repetir las veces que sean necesarios. Los bucles nos facilitan evitar la repetición de líneas de código, ya sea que, si sean necesarias o no, se sepa la cantidad de iteraciones que son necesarias o no, o que las iteraciones dependan de una variable de control.

En el video nos explican lo importante que es el uso de bucles, ya que nos permite realizar un proceso las veces que sean necesarias, como mostrar una ventana de acceso (login) las veces que sean necesarias hasta que el usuario ingrese los datos (usuario – contraseña) de forma correcta.

En el video se nos explica los dos grupos en los que se clasifican los bucles:





Y finalmente se nos explica cómo se compone un ciclo FOR en Python.

```
aux = 0
x = 0

for k in [2,4,6,8]:
    aux = k+aux;
    print(k, " ", aux)

for j in ("HOLA"):
    print(j)

aux = 0
for x in (range(4)):
    print(" - ", x)
    aux = aux + x
    x+1

print(" ", aux)
```

Video 15:

En este video seguimos trabajando en el bucle FOR y se nos da una explicación de cómo podemos recorrer un String. Se nos pone un ejercicio práctico de cómo podemos utilizar el ciclo FOR para validar si una cadena de texto es un correo electrónico, es decir, que contenga una Arroba (@) y que en dicha cadena de texto este presente la cadena de caracteres ".com". (Corrección, en este punto ya que estamos haciendo el recorrido por cada uno de los caracteres, solo nos es posible saber si contiene puntos ".")

```
def validarcorreo():
    correo = input("Ingrese su correo : ")
    contador=0
    for i in correo:
        if (i == "@" or i == "."):
            contador=contador+1

    if(contador ==2):
        print("Email Correcto")
    else:
        print("Email incorrecto")
```

Para realizar la validación se puede hacer por medio de una variable que acumule los aciertos o las veces que se cumpla las condiciones empleadas en el condicional if (Si i, que es la variable iteradora, es igual a una arroba O es un punto '.'), si se cumple se sumara a la variable acumuladora, la cual después debe validarse si es mayor o igual a 2. Cabe destacar que aquí tenemos un fallo: puede que el usuario



ingrese un correo que contenga dos puntos y haga que se valide como un correo valido sin contener una arroba.

```
def validarcorreo():
    correo = input("Ingrese su correo : ")
    contador=0
    arroba = False
    for i in correo:
        if (i == "."):
            contador=contador+1
        elif (i == "@"):
            arroba = True

    if(contador >=1 and arroba == True):
        print("Email Correcto")
    else:
        print("Email incorrecto")
```

Aquí ya podemos ver cómo, sin importar la cantidad de puntos existan, solo una vez validaremos que exista el carácter arroba "@", por lo cual esto lo haremos por medio de una variable que acumule las veces que encuentre un punto y una variable booleana que solo será verdad si se encuentra una arroba.

También es este video vimos el uso de un elemento concatenador de cadenas muy útil en bucles como lo es "end=""", el cual nos permite imprimir texto sin hacer un salto de línea. (Similar a "\n" en java).

```
for i in ["x","y","z",2]:
    print("Hola ",end="")
```

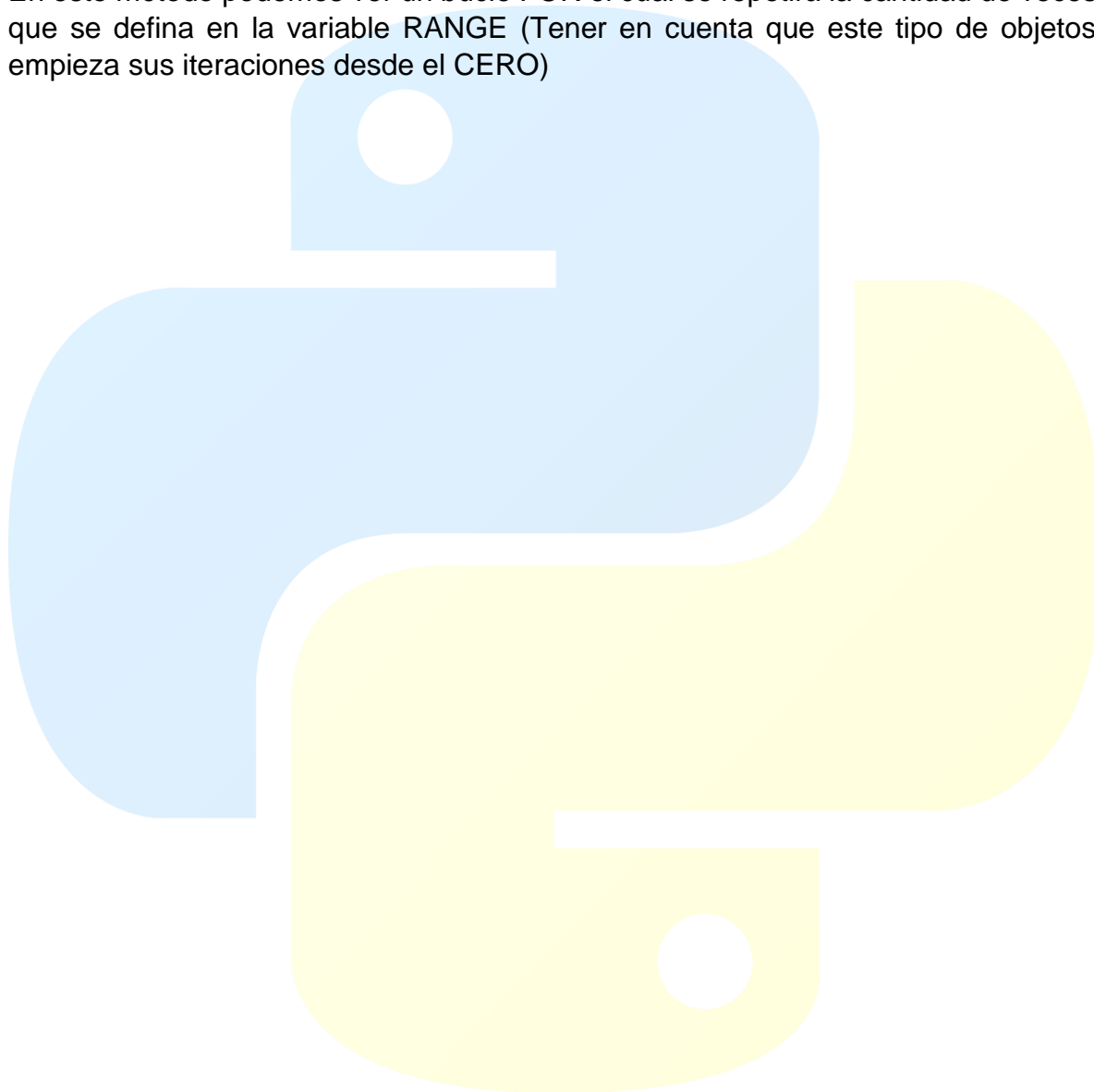
En estas dos líneas de código creamos un bucle FOR, el cual hará su recorrido la cantidad de elementos que contenga la lista ingresada, e imprimirá "Hola" la cantidad de veces que se haya especificado en la misma línea.

Finalmente se habló del uso de un objeto que es reciente en el lenguaje Python, el cual es el objeto RANGE, el cual es una variable que nos ayudara a estipular las iteraciones o veces que queremos se repita un bucle.



```
def rango():  
    for i in range(5):  
        print(i, " xxxx")
```

En este método podemos ver un bucle FOR el cual se repetirá la cantidad de veces que se defina en la variable RANGE (Tener en cuenta que este tipo de objetos empieza sus iteraciones desde el CERO)



**Video 16:**

En este video seguimos viendo el uso del objeto de tipo RANGE y las implementaciones de este, además de seguir viendo notaciones en PRINT.

Primero se nos presenta como hacer el uso de una función en medio de un PRINT por medio del uso de

Print(f"El texto que necesitemos {una variable de tipo numérica}")

Esto nos permite incluir valores numéricos en medio de una cadena de texto:

```
def rango():  
    for i in range(5):  
        print(f"Valor de la variable {i}")  
  
rango()
```

```
Valor de la variable 0  
Valor de la variable 1  
Valor de la variable 2  
Valor de la variable 3  
Valor de la variable 4  
>>> |
```

A continuación, se muestra las posibilidades que tenemos con un objeto de tipo **RANGE**:

```
def rango():  
    ## Uso de range hasta un rango numerico  
    for i in range(5):  
        print(f"Valor de la variable {i}")  
    ## Uso de range entre dos valores  
    for i in range(2,7):  
        print(f"Valor de la variable {i}")  
    ##Uso de Range indicando el tamaño de las iteraciones  
    for i in range(5,50,3):  
        print(f"Valor de la variable {i}")  
  
rango()
```

Finalmente, se nos explica cómo usar el tamaño de una cadena en un objeto de tipo RANGE con el mismo ejercicio del video anterior de validar un email.



```
def email():
    contador = 0;
    valido=False
    email=input("Introduce tu email: ")
    for i in range(len(email)):
        if (email[i] == "."):
            contador=contador+1
        elif (email[i] == "@"):
            arroba = True

    if(contador >=1 and arroba == True):
        print("Email Correcto")
    else:
        print("Email incorrecto")
```

Aquí debemos notar algo importante: al momento de utilizar la longitud del email ingresado estamos hablando de un arreglo, y ya que un objeto de tipo RANGE se asemeja a un arreglo, debemos recordar que las validaciones aquí las debemos hacer con relación a la cadena de String y la posición en la que se cuenta nuestra variable iteradora.

Ejercicios del video 16:

```
def num1a1100():
    for i in range(1,101):
        print(i,end=" ")

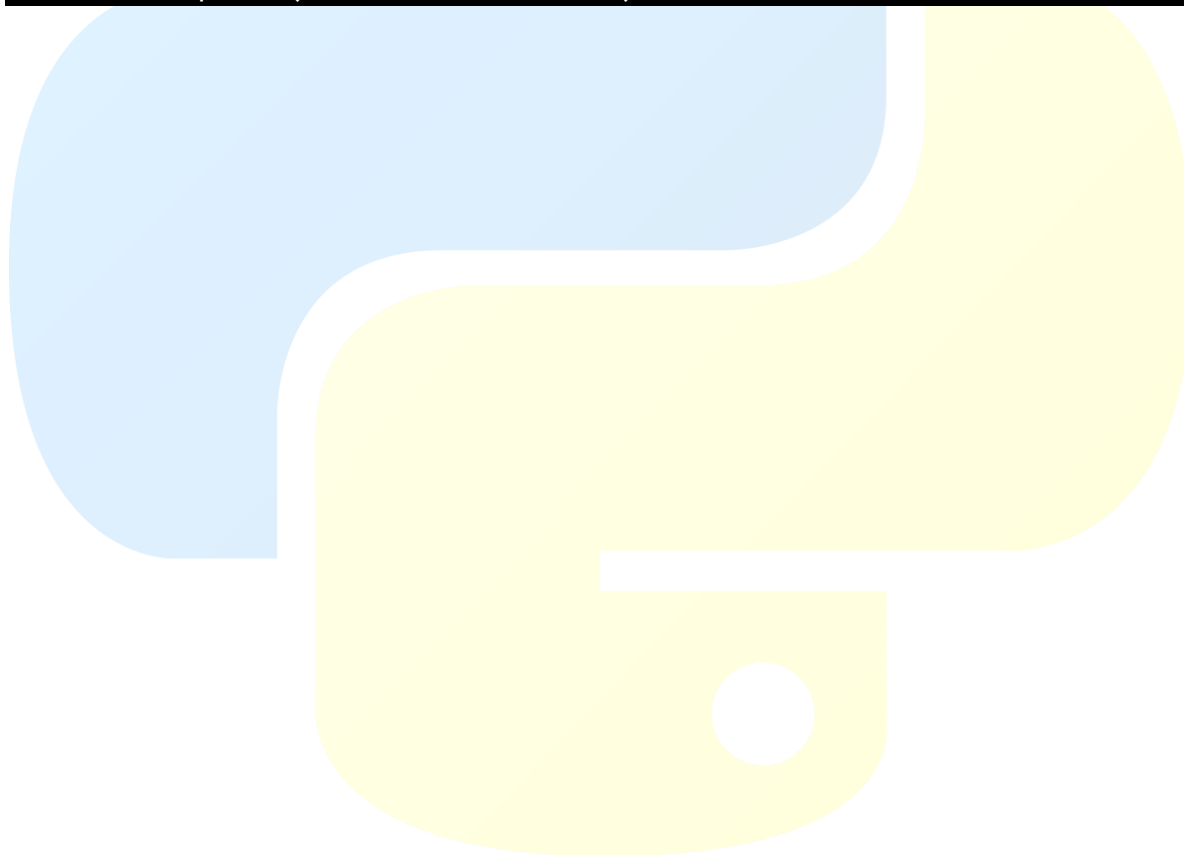
def contrasena():
    cont=input("INGRESE LA CONTRASEÑA: ")
    espacio = False;
    if len(cont) < 8:
        print("La contraseña tiene menos de 8 caracteres")
    else :
        for i in range(len(cont)):
            if cont[i] == " ":
                espacio = True;

        if espacio == True:
            print("La contraseña tiene espacion")
        else :
            print("Contraseña aceptable")
```



```
def validarcorreo():
    correo = input("Ingrese su correo : ")
    contador=0
    arroba = False
    for i in correo:
        if (i == "."):
            contador=contador+1
        elif (i == "@"):
            arroba = True

    if(contador >=1 and arroba == True):
        print("Email Correcto")
    else:
        print("Email incorrecto")
```





Video 17:

En este video se nos presenta el bucle WHILE y cómo podemos implementarlo, además de cómo evitar que se vaya al infinito.

En el primer ejercicio, utilizamos un ciclo WHILE para poder validar una edad:

```
def edad():
    edad=int(input("Edad del usuario: "))
    while edad<0 or edad>100:
        print("Edad erronea, vuelve a intentarlo")
        edad = int(input("Edad de nuevo: "))

    print(" F I N \nEdad = ",edad)
```

Aquí podemos ver como el bucle While se comporta como un condicional IF, en el que revisaremos una condición para saber si se ejecutan o no el bloque que este contiene. Este ciclo al estar ligado a una variable ingresada por el usuario se ve como un ciclo que puede irse al infinito o de forma indefinida.

Finalmente se realiza un ejercicio en el cual queremos hallar la razón cuadrada de un numero ingresado por el usuario.

```
def raices():
    print("Calculadora de raices")
    num=int(input("INGRESE EL NUMERO A EVALUAR: "))
    i=0
    while num < 0:
        print("No se pueden hallar raices de números negativos")

        if i == 2:
            print("Muchos intentos")
            break;

        num=int(input("INGRESE EL NUMERO A EVALUAR: "))
        if num <0:
            i=i+1

    if i<2:
        sol=math.sqrt(num)
        print("La raices cuadrada de ",num," es ",sol)
```



Aquí podemos ver como en un bucle evaluara si un número es negativo, si el número es negativo, se le informara al usuario y se le pide volver a añadir el usuario ingresar otro número, miremos que la variable *i* se evalúa en un condicional IF dentro del bucle: si *i* llega a ser igual a dos, quiere decir que el usuario llego al límite que demarcamos, lanzamos un mensaje para que el usuario sepa que llego al límite y usamos la sentencia BREAK la cual obliga a finalizar el bucle WHILE. Caso opuesto de que el usuario haya dado un numero positivo en sus dos intentos, se procederá a realizar la razón cuadrada del usuario.

NOTA: se hace la importación de una clase de Python para realizar la raíz (import math).

Ejercicios del video 17:

```
▼ def numerosinf():
    x = -999
    y = int(input("Ingrese un numero : "))
    ▼ while y > x:
        x = y
        y = int(input("Ingrese un siguiente numero : "))

    print("FIN DEL PRGRAMA")

numerosinf()

▼ def numpos():
    con = 0;
    y = int(input("Ingrese un numero : "))
    ▼ while y > 0:
        con = con + y;
        y = int(input("Ingrese un siguiente numero : "))
    print("Sumatoria : ", con)

numpos()
```

**Video 18:**

En este video de cierre en los bucles vimos tres instrucciones que son muy utiles en la implementación de ciclos:

- CONTINUE: Se utiliza para cortar el flujo de un ciclo, es decir, no ejecuta las líneas que contenga después de él el bucle y pasara a la siguiente iteración.
- PASS: devuelve un NULL – no ejecuta el bloque de código (útil en otros niveles).
- ELSE: funciona de forma similar que en un condicional IF.

```
def espacios():
    texto="pildoras informaticas"

    print("Tamaño",(len(texto)))
    cont=0
    for letra in texto:
        if letra==" ":
            continue
        cont+=1

    print(cont)

def correos():
    email=input("Ingrese correo : ")
    for i in email:
        if i=="@":
            arroba=True
            break;
    else:
        arroba=False

    print(arroba)

correos()
```

En el primer método podemos ver como por medio de la sentencia CONTINUE hacemos el salto a la siguiente iteración, evitando que el espacio encontrado se sume a contador.

Por último, volvemos a retomar el ejercicio de validar un correo electrónico, de modo tal que si se encuentra una arroba se debe crear la variable booleana Arroba que valida que, si existe, si después de todo el bucle FOR no se encuentra una arroba se lanza la sentencia ELSE que le da el valor a la misma variable booleana como False, diciendo que no se encontró ninguna arroba.



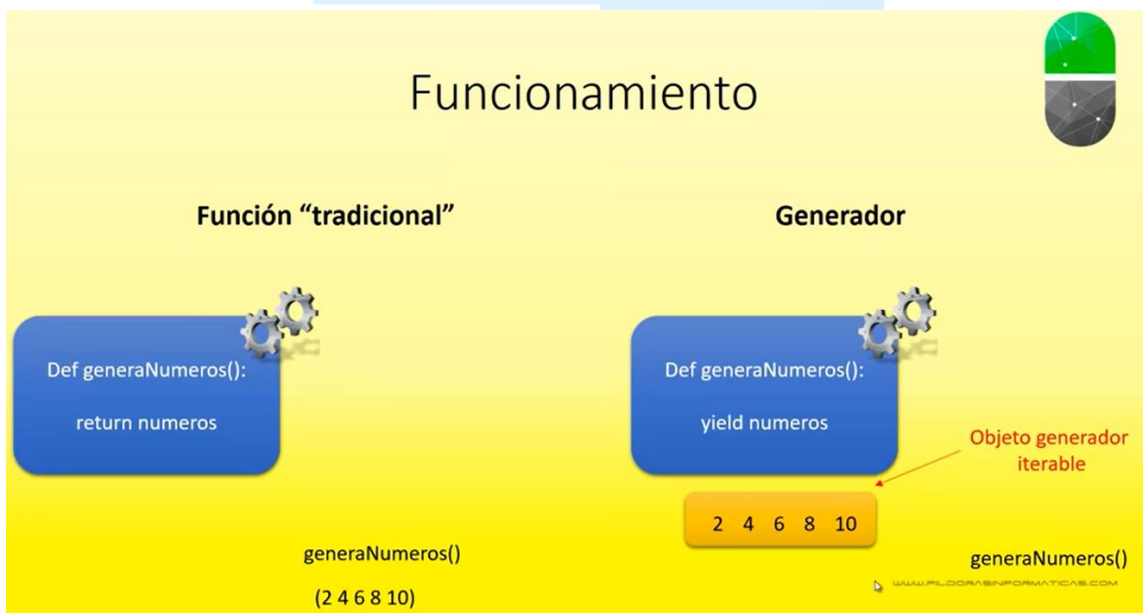
GENERADORES

En esta sección del curso se explicará todo lo relacionado con los generadores, su definición, implementación en el código y usos posibles en la programación.

Video 19

En este video entramos a un nuevo tópico (y enserio nuevo, ya que en java no existe nada similar a esto) el cual se llama generadores. Se define que un generador es una estructura que extrae valores de una función, de la misma forma que se puede hacer con un objeto iterable, dando la posibilidad de extraerlos por medio de un bucle o con otras estructuras.

A continuación, se presentan las diferencias que existen entre una función normal y un generador:



Como vemos, se hace la comparación de una función que nos genera una lista de números, ya sea siguiendo un parámetro de entrada que lo limite o no. Esta función nos retornara una lista de a la cual podemos acceder por medio de un bucle, pero por lo cual se nos darán todos los valores de una sola vez, en cambio, al lado derechos tenemos un generador, el cual sigue los patrones de una función pudiendo recibir una variable limitante o no, pero al momento de instanciar nuestro generador lo haremos usando un objeto que almacene lo generado por dicha función generadora, este objeto será nuestro iterador, y ay sea que accedamos por medio de un bucle, o haciendo uso de la sentencia `next`, podremos acceder a cada uno de los valores generados, uno a uno.



Tener en cuenta que, al crear un generador, no usaremos return para enviar los valores obtenidos a donde haya sido hecha la instancia, sino yield, el cual retornara solo un elemento de los que se encuentren creados.

Las ventajas de utilizar un generador son:

- El uso de memoria frente a una función que genere una lista será menor ya que no tendrá que almacenarse toda la lista, sino solo un elemento.
- Menos tiempo en ejecución.
- Es muy útil solo en casos de que se requiera solo obtener uno de los valores obtenidos.

Ya al entrar en comparar el modo en que se comportan las funciones y los generadores podemos apoyarnos en el ejemplo visto:

```
def generarN(limite):  
    num=1  
    milista=[]  
    while num<limite:  
        milista.append(num*2)  
        num=num+1  
    return milista
```

En esta función vemos como se recibe un entero que será el límite de la lista que se va a crear, se tiene una variable que será iteradora en un ciclo while, la cual generará un incremento en la variable iteradora y se agrega por medio de la sentencia. Append a la lista. Al terminas este bucle se retornará la lista generada.

```
def generaP(limite):  
    num=1  
    while num<limite:  
        yield num*2  
        num=num+1
```

Por otro lado, en un generador seguimos recibiendo un entero limitante, pero con la diferencia que no tendremos una lista. Aquí lo que se hace es por medio del mismo bucle while iremos creando el objeto iterable por medio del yield, que almacena los elementos y los retorna al lugar donde se haya instanciado el generador.

Al momento de acceder a un elemento del objeto iterable lo hacemos de la siguiente forma:



```
devuelvePares=generaP(10)
|
print(next(devuelvePares))

print("Stop sfg")

print(next(devuelvePares))

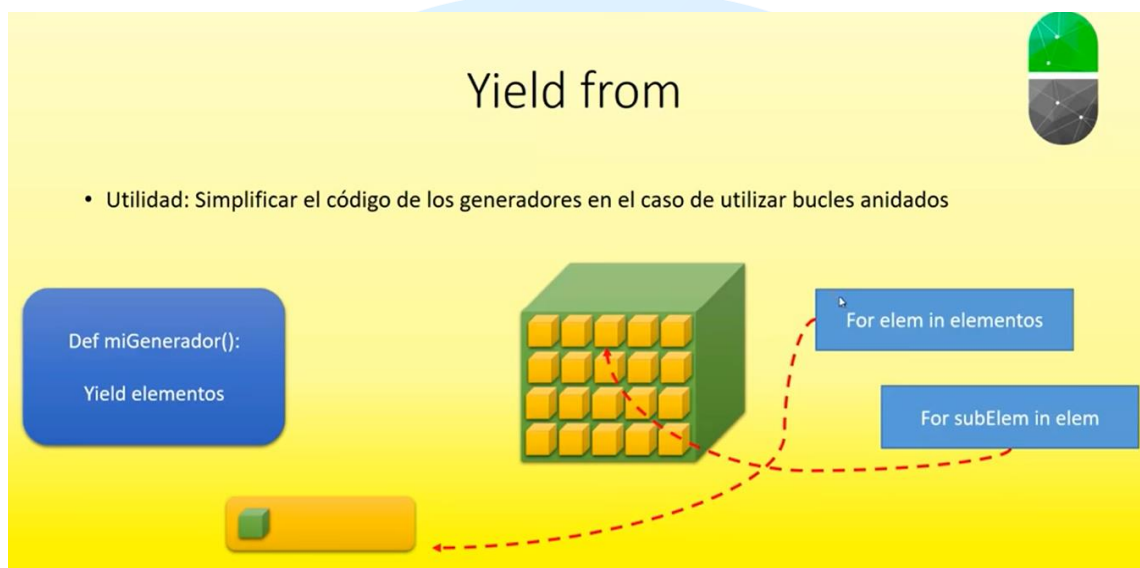
print(next(devuelvePares))
```

Si vemos aquí, se crea el objeto iterable que almacena lo creado por medio del generador, y para acceder a cada elemento usamos la sentencia `next`, la cual dará un paso por cada uno de los elementos generados, de este modo podemos acceder solo a un elemento y no a toda una lista, haciendo que sea más eficiente el uso de memoria y el flujo en un programa.



Video 20:

En este video seguimos trabajando los generadores, y más precisamente en la sentencia YIELD la cual podemos utilizar de otra forma. Cuando se están trabajando ciclos anidados, podemos necesitar extraer algo del producto de este, por lo cual podemos usar la sentencia YIELD FROM, esto nos ayudara a poder extraer valores generados de estos bucles o ciclos.



Aquí se nos explica cómo se puede asemejar el uso de la sentencia YIELD FROM, como la extracción de un elemento que se encuentra en un array bidimensional.

Nota: al momento de definir el parámetro de entrada en un generador o función, si usamos el símbolo asterisco “*” le estamos diciendo a la función que recibirá un número indeterminado de elementos además de que se le indica a la función que recibirá dichos elementos por medio de una tupla.

Después se hizo la comparativa realizando dos generadores:

```
def devuelveciudades2(*ciudades):  
    for elemento in ciudades:  
        yield elemento
```

Es este generador se ingresa una tupla de ciudades, y por medio del ciclo FOR se ira retornando cada uno de los elementos que están contenidos en la tupla.



```
ciudades_devueltas2=devuelveciudades2("Madrid","Barcelona","Bilboa","Valencia")  
print(next(ciudades_devueltas2))  
print(next(ciudades_devueltas2))  
print(next(ciudades_devueltas2))  
print(next(ciudades_devueltas2))
```

Aquí accedemos a cada uno de los elementos de la tupla, pero si queremos acceder a los sub-elementos de la tupla, que en este caso son las letras de cada uno de los elementos debemos realizar lo siguiente.

```
def devuelveciudades(*ciudades):  
    for elemento in ciudades:  
        for subElemento in elemento:  
            yield from subElemento  
  
ciudades_devueltas=devuelveciudades("Madrid","Barcelona","Bilboa","Valencia")  
  
print(next(ciudades_devueltas))  
print(next(ciudades_devueltas))  
print(next(ciudades_devueltas))  
print(next(ciudades_devueltas))  
print(next(ciudades_devueltas))
```

Aquí podemos ver como se ingresa la misma tupla de ciudades al generador, y por medio de un primer ciclo FOR accederemos a cada uno de los elementos de esta tupla, luego, por medio de un segundo ciclo FOR accederemos a los sub-elementos de este y los retronaremos uno a uno.

En conclusión, la sentencia YIELD FROM es útil para acceder a datos obtenidos por medio de ciclos anidados, así de este modo es más fácil mantener un mejor flujo del programa. (nota: si se escribe solo YIELD, el generador funcionara de la misma forma que lo haría con un YIELD FROM)



EXCEPCIONES

En esta sección del curso se explicará todo lo relacionado con las excepciones, como podemos controlarlas y la gran importancia del manejo de estas al momento de programar.

Video 21:

En este video entramos a una nueva unidad de trabajo tan importante como programador como es el control de excepciones. Hay que recordar que el control de excepciones es muy útil para evitar que el flujo de un programa se vea afectado por algún problema en tiempo de ejecución.

Según lo explican en el video:

Excepciones ¿Qué son?

- Las excepciones son errores que ocurren durante la ejecución del programa. La sintaxis del código es correcta pero durante la ejecución ha ocurrido "algo inesperado".
- Este tipo de errores de ejecución se pueden controlar para que la ejecución del programa continúe. Es lo que se conoce como **manejo o control de excepciones**.

¿ ?

www.pildorasinformaticas.com

En este video se hace un control de excepciones doble, y algo que personalmente me interesa mucho es el uso de opciones. Quiero recordar que en Python no contamos con una estructura lógica que nos permita tener casos u opciones, por lo cual debemos apoyarnos en la estructura condicional IF para evaluar todas las opciones que necesitemos.

Nota: el archivo subido por el instructor o la página no compila correctamente, por lo cual me tome la libertad de crear mi propio mini-programa, con ciertas mejoras.

El código subido es el siguiente:



```
1 def suma(num1, num2):
2     return num1+num2
3
4 def resta(num1, num2):
5     return num1-num2
6
7 def multiplica(num1, num2):
8     return num1*num2
9
10 def divide(num1,num2):
11     return num1/num2
12
13
14 op1=(int(input("Introduce el primer número: ")))
15
16 op2=(int(input("Introduce el segundo número: ")))
17
18 operacion=input("Introduce la operación a realizar (suma,resta,multiplica,divide): ")
19
20 if operacion=="suma":
21     print(suma(op1,op2))
22
23 elif operacion=="resta":
24     print(resta(op1,op2))
25
26 elif operacion=="multiplica":
27     print(multiplica(op1,op2))
28
29 elif operacion=="divide":
30     print(divide(op1,op2))
31
32 else:
33     print ("Operación no contemplada")
34
35 print("Operación ejecutada. Continuación de ejecución del programa ")
```

En este bloque de código podemos ver como se crean unas funciones las cuales solo se retornará la solución a una operación aritmética (suma, resta, multiplicación y división), recibiendo como parámetros de entrada dos variables numéricas. Luego vemos como se le pide al usuario ingresar dos valores numéricos, e ingresar la opción que desea, este código no tiene un control de excepciones.



```
def suma (x,y):
    return x+y
def resta (x,y):
    return x-y
def multi (x,y):
    return x*y
def div (x,y):
    try:
        return x/y
    except ZeroDivisionError:
        print("No se puede dividir entre cero.")
        return "Operacion erronea"

while True:
    try:
        opc = int(input( "1. Suma / 2. Resta / 3. Multiplicacion / 4. Divicion / 0. Salida;  "))
        if opc == 0:
            print("A D I O S")
            break
        except ValueError:
            print("Se ingreso un valor no numerico.")
            continue

        try:
            x = int(input("INGRESE EL PRIMER NUMERO : "))
            y = int(input("INGRESE EL SEGUNDO NUMERO : "))
        except ValueError:
            print("Se ingreso un valor no numerico.")
            continue

        if opc == 1:
            print(x,"+",y,"=",suma(x,y))
        elif opc == 2:
            print(x,"-",y,"=",resta(x,y))
        elif opc == 3:
            print(x,"x",y,"=",multi(x,y))
        elif opc == 4:
            print(x,"/",y,"=",div(x,y))
        else:
            print("OPCION INVALIDA")
```

En este bloque de código podemos ver como se controlan dos problemas:

1. Hay que recordar que cualquier división por cero da un numero infinito, lo cual imposibilita que se de un resultado, y si miramos el método DIV veremos que en este se usa un bloque TRY – EXCEPT, en el cual, primero si intenta realizar el envío de la solución de la división, si se encuentra que es imposible realizar, se demarca la excepción *ZeroDivisionError* la cual esta especificada para divisiones en cero (revisar lista de excepciones en Python), y se retorna un mensaje al usuario.
2. En el video se realiza el uso de un menú por medio de cadenas de texto, lo cual es poco eficiente, y ya que también debemos controlar el hecho de que el usuario ingrese algún carácter como alguno de los dígitos operadores, realizamos otro bloque TRY – EXCEPT el cual evitara que el programa caiga y de este modo poder seguir el flujo del programa.
3. Extra: ya que esto se puede considerar como una calculadora, encapsule todo el bloque de código con el que el usuario tendrá contacto en un bucle WHILE, el cual le posibilitara al usuario realizar todas las veces necesarias el uso del programa.



4. Según mi lógica, creo que se debe evaluar primero lo que quiere hacer el usuario, ya que si el usuario no quiere continuar con el programa debe hacerlo antes de que tenga que ingresar los datos, por lo cual creo un TRY – EXCEPT único para el bloque de código en que se valida lo elegido por el usuario.





Video 22:

En este video se sigue trabajando sobre el mismo código del video anterior (La primera parte del video yo por mi cuenta ya la había trabajado así que solo me centrare en las dos ultimas partes).

En este video vemos como se pueden capturar varias excepciones:

```
def divicion ():  
    print("D I V I Ó N")  
    try:  
        x = int(input("INGRESE EL PRIMER NUMERO : "))  
        y = int(input("INGRESE EL SEGUNDO NUMERO : "))  
        return x/y  
    except ZeroDivisionError:  
        print("No se puede dividir entre cero.")  
        return "Operacion erronea"  
    except ValueError:  
        print("Se ingreso un valor no numerico.")
```

Aquí podemos ver como se le pide al usuario que ingrese dos valores numéricos (int or float) y por medio de un bloque TRY – EXCEPT se evalúa si existe una división por cero o, si el valor ingresado no es un valor numérico, en resumidas cuentas, es resumir lo que se hizo en el anterior ejercicio en una sola función.

NOTA: si queremos demarcar una excepción GENERALIZADA



Video 23:

En este video se nos explica cómo es la creación de errores por medio de la sentencia RAISE. Esta sentencia nos permite crear el mensaje que el usuario leerá cuando el programa caiga.

```
import math

def evaluacion (edad):
    if edad < 0:
        raise TypeError("No existen edad negativas")
    if edad < 20:
        return "Eres muy joven"
    elif edad < 40 :
        return "Eres joven"
    elif edad < 65:
        return "Eres mayor"
    elif edad < 100:
        return "See you in heaven"

edad=int(input("Ingresa tu edad: "))

print(evaluacion(edad))
```

Tener en cuenta que, al momento de crear la excepción, debemos tener en cuenta que podemos usar el nombre de cualquier error especificado en Python, no importa cual, ya que estamos haciendo la evaluación de una variable numérica que cumpla un rango, en este caso que sea positiva.

```
Ingresa tu edad: -5
Traceback (most recent call last):
  File "pruebaexx3.py", line 17, in <module>
    print(evaluacion(edad))
  File "pruebaexx3.py", line 5, in evaluacion
    raise TypeError("No existen edad negativas")
TypeError: No existen edad negativas
```

El problema en la implementación de este tipo de excepciones esta en el hecho de que el resto de las líneas de código después de que se capture esta excepción no se ejecutaran.



```
def raices(numero):  
    if numero<0:  
        raise ValueError ("Numero no puede ser negativo")  
    else:  
        return math.sqrt(numero)  
  
opc=(int(input("Introduce valor : ")))  
  
try:  
    print("Raiz cuadrada de " +opc+" es " +raices(opc))  
except ValueError as e:  
    print(e)
```

En este bloque de código podemos ver dos formas de controlar errores en tiempo de ejecución:

- La primera, lanzando un error que, aunque haga caer el programa, indique cual fue el error.
- Y el segundo, que es mas eficaz, es la implementación de un bloque TRY – EXCEPT, en el cual, usamos un error en tiempo ejecución de los especificados en Python y por medio del uso de AS lo asimilamos como el error que necesitemos, sin olvidar la impresión por consola del error. Esta opción es mas factible ya que no provoca el colapso el programa.

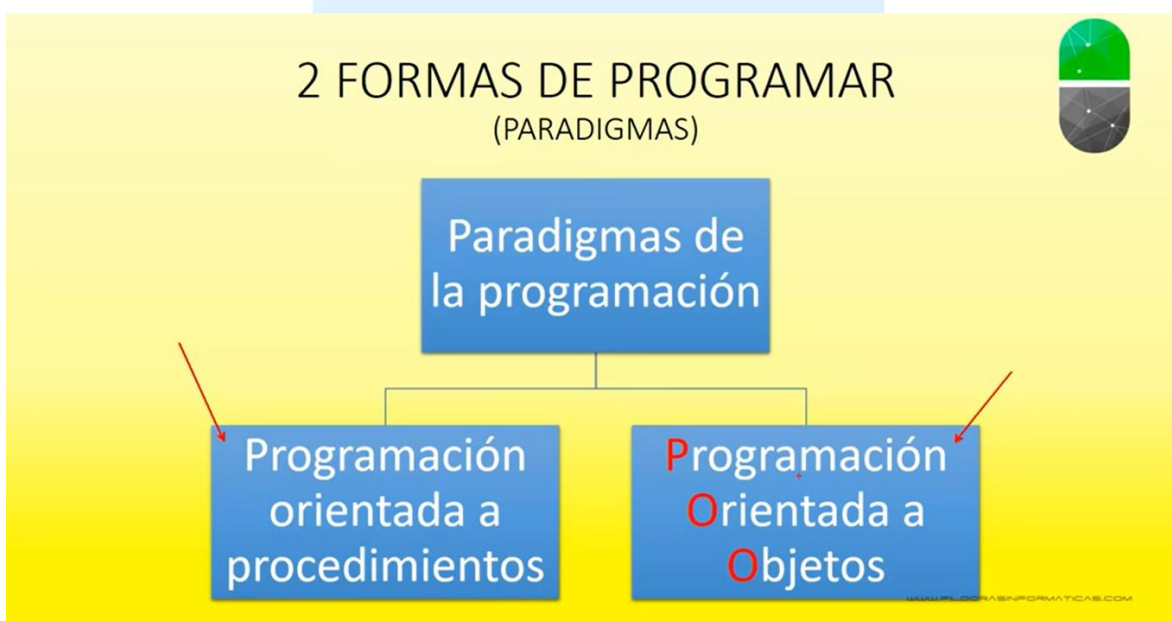


PROGRAMACIÓN ORIENTADA A OBJETOS

En esta sección del curso se explicará todo lo relacionado con la programación orientada a objetos, creación de clases y aplicación de elementos propios de la programación orientada a objetos.

Video 24:

En este video iniciamos un nuevo tema tan importante en el mundo de la programación como lo es los Paradigmas de programación, y uno fundamental en la actualidad: la programación orientada a objetos. Ya que Python es uno de los mejores lenguajes para trabajar en este paradigma, es importante recordar los dos paradigmas de la programación:



Como podemos ver aquí, como existen dos paradigmas:

- Al lado izquierdo encontramos la programación que esta solo dirigida a los procedimientos, la cual se puede considerar como una programación poco eficiente, ya que en esta el código se vuelve muy extenso, líneas de código que cumplan una función deberán ser escritas varias veces, esto desencadenara en que la programación y las líneas de código estén en forma espagueti, es decir, una tras la otra, sobrecargando los procesos y haciendo poco eficiente el uso de esta.



Programación orientada a procedimientos

- Programación Orientada a procedimientos:
 - Algunos ejemplos de lenguajes: Fortran, Cobol, Basic etc.
- **Desventajas:**
 - Unidades de código muy grandes en aplicaciones complejas.
 - En aplicaciones complejas el código resultaba difícil de descifrar.
 - Poco reutilizable.
 - Si existe fallo en alguna línea del código, es muy probable que el programa caiga.
 - Aparición frecuente de código espagueti.
 - Difícil de depurar por otros programadores en caso de necesidad o error.

```
10 LET X=0
20 LET Y=X
30 LET Z=X
40 LET X=X+1
50 GO SUB 1000
60 LET Y=INT (4*RAND)
70 LET Z=INT (6*RAND+1)
80 IF X=31 OR Y=31 THEN PRINT
"BRUJO " ; Z
90 PRINT "JUGADOR " ; Z+1 ; " : "
100 PAUSE 425
110 GO SUB 1000
120 LET I=INT (4*RAND)
130 GO SUB I*(RAND*5)+50+150
140 GO TO 50
150 RETURN
200 LET I=I+1
210 RETURN
220 LET I=I+1
230 RETURN
300 IF NOT J THEN LET X=INT (30*
RAND)
310 IF J THEN LET Y=INT (30*
RAND)
320 RETURN
1000 IF NOT J THEN LET X=X+I*(X+
I*31)
1005 LET Y=Y+J*I*(Y+I*31)
1010 FOR I=0 TO 31
1020 PRINT AT 3, I, I AND I=9, PR
INT AT 3, I:CHR$ (55+I) AND I=9
1030 PRINT INK 1:AT 4, I:CHR$ (50
+I*2) AND I=9
1040 PRINT INK 2:AT 5, I:CHR$ (50
+I*2) AND I=9
1050 NEXT I
1060 RETURN
```

En este tipo de programación se tenía que usar instrucciones como “go to” o “go back” para poder acceder a ciertas regiones de código, lo cual hacía que la ejecución de un programa modelado según este paradigma fuera tediosa. Por último, uno de los mayores problemas que enfrentaba este tipo de programación era el manejo de problemas en tiempo de ejecución, ya que al estar todo el programa en una estructura línea tipo espagueti si se presentaba un problema en cualquier línea de código, esto afectaba a las que seguían después de esta, volviendo imposible el acceso a estas.

- Al lado derecho de la primera imagen de este video, nos encontramos con el paradigma más importante de la programación, el cual busca trasladar los comportamientos y cualidades de objetos del mundo real y tangible a la programación.

Programación Orientada a objetos

- ¿En qué consiste?
 - Trasladar la naturaleza de los objetos de la vida real al código de programación.
- ¿Cuál es la naturaleza de un objeto de la vida real?
 - Los objetos tienen un estado, un comportamiento (¿Qué puede hacer?), y unas propiedades
- Pongamos un ejemplo: El objeto coche.
 - ¿Cuál es el estado de un coche? Un coche puede estar parado, circulando, aparcado etc
 - ¿Qué propiedades tiene un coche? Un coche tiene un color, un peso, un tamaño etc.
 - ¿Qué comportamiento tiene un coche? Un coche puede arrancar, frenar, acelerar, girar etc.





Como podemos ver en la imagen, la programación orientada a objetos consiste en traducir objetos y elementos de la vida real a líneas de código, lo que conlleva a especificar en líneas de código los atributos, es decir, las características que este posee, y el comportamiento, sus actividades o acciones en el mundo real.

En el video se nos muestra el ejemplo más simple y básico que nos ayuda a entender como se trabaja en este paradigma:

OBJETO

¿Qué es?

- Objeto:
 - Tiene propiedades (atributos):
 - Color
 - Peso
 - Alto
 - Largo
 - Tiene un comportamiento (¿Qué es capaz de hacer?):
 - Arrancar
 - Frenar
 - Girar
 - Acelerar



Recordemos que un objeto no solo hace referencia los bienes materiales, hace referencia a cualquier actor (persona, animal, vehículo, artículo, etc.) el cual tenga características TANGIBLES y realice cualquier tipo de actividad (o que se permita realizar una actividad o acción con él).



Basado en esto ya podemos ver como se trabaja en POO:

Programación Orientada a objetos

- Programación Orientada a objetos:
 - Algunos ejemplos de lenguajes: C++, Java, Visual.NET etc.
- **Ventajas:**
 - Programas divididos en "trozos", "partes", "módulos" o "clases". Modularización.
 - Muy reutilizable. Herencia.
 - Si existe fallo en alguna línea del código, el programa continuará con su funcionamiento. Tratamiento de Excepciones.
 - Encapsulamiento.



Este paradigma está presente en la mayoría de los lenguajes que son más usados en la actualidad, y con relación a las características de este podemos ver como hay presencia de reutilización de código, herencia de los atributos o métodos de los objetos, el modularidad de cada acción, lo cual posibilita el uso de métodos y creación de objetos en cualquier parte del programa, y por último, pero menos importante podemos ver la presencia del tratamiento de errores en tiempo de ejecución.

**Video 25:**

En este video vamos a seguir hablando sobre la programación dirigida a objetos y más específicamente de el vocabulario que como programador debemos manejar en este paradigma.

1. Clase

Modelo donde se especifican las características que posee un objeto o un conjunto de objetos. Cuando hablamos de las características nos referimos al conjunto de atributos que componente al objeto y las acciones que este cumple. En el video seguimos trabajando sobre el ejemplo del coche.



Al nosotros crear el modelo de un coche debemos especificar sus atributos básicos como lo son:

- Color
- Tamaño del chasis
- Combustible
- Placa
- Ancho
- Altura
- Puestos

2. Objeto

Hay que recordar que un objeto es el producto de instanciar una clase.

Al hacer la creación de un objeto debemos tener en claro la nomenclatura que debemos seguir, tanto para la creación del objeto con cada uno de sus atributos, como las acciones de este.



Objeto.

Accediendo a propiedades y comportamiento (pseudocódigo)

nomnclatura del punto|

- Objeto:
 - Accediendo a propiedades de objeto desde código (pseudocódigo):
 - `miCoche.color="rojo";`
 - `miCoche.peso=1500;`
 - `miCoche.ancho=2000;`
 - `miCoche.alto=900;`
 - Accediendo a comportamiento de objeto desde código (pseudocódigo):
 - `miCoche.arranca();`
 - `miCoche.frena();`
 - `miCoche.gira();`
 - `miCoche.acelera();`

Aquí podemos ver la forma en que accedemos, tanto a los atributos de una clase, como al comportamiento que esta posee.

3. Instancia de clase

Cuando nos referimos a una instancia es la creación de un objeto perteneciente a una clase, con todas sus atributos y acciones.

INSTANCIA



- Ejemplar perteneciente a una clase



Clase



Objetos



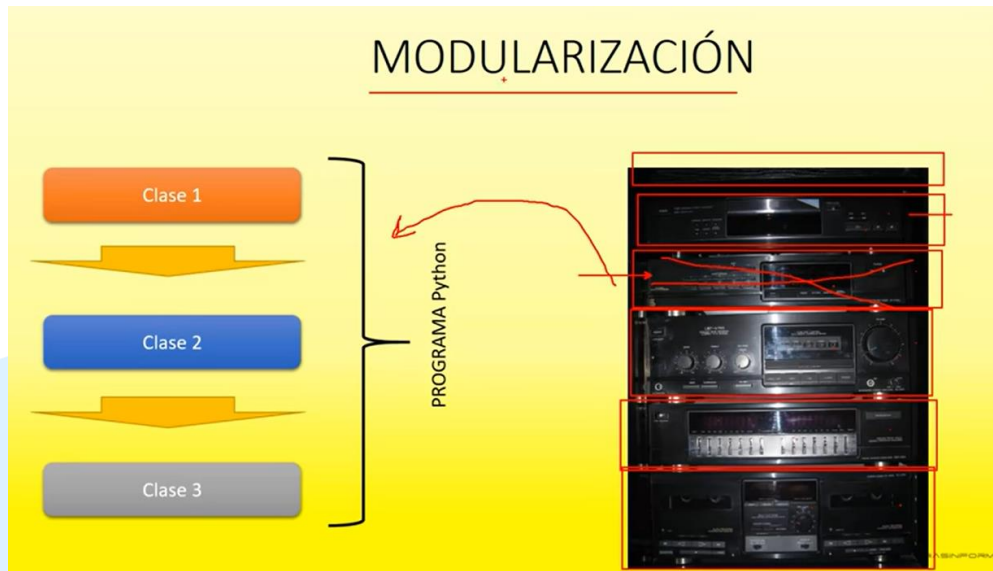
4. Modularización

El concepto de Modularización nace del hecho de que un programa se compone no solo de una clase (u objetos) sino de varios, los cuales interactúan entre sí. En el video hablamos de modularidad ya que permite trabajar desde un centro el cual puede necesitar todas las clases creadas o



no, también se aplica en el hecho de que podemos usar partes de ciertas clases dependiendo de las necesidades que tengamos para programar.

En el video se hace la analogía del uso del concepto de modularidad con los equipos de sonidos de hace ya unos años atrás:



La analogía que se realiza aquí está basada en el hecho de como estos aparatos se componían de varios elementos como radio, lector de casetes, tocadiscos o lectores de CDs, si alguno de estos componentes se dañaba, no era necesario comprar un aparato completo sino solo comprar el componente que está dañado, esto mismo se aplica en los programas, ya que, alguna de las clases de nuestro programa falla, solo necesitamos ir a arreglar el fallo en la clase que presenta error.

5. Encapsulamiento

Al hablar de encapsulamiento seguimos tratándolo sobre el ejemplo del equipo de sonido, cada parte es una capsula, que será ejecutada o usada las veces que el usuario quiera, y en un programa es igual, sus partes están interconectadas, de forma que podamos acceder a elementos o acciones del programa que sean parte de otra clase. Analógicamente en un equipo de sonido es igual: usamos la parte de lectura de casetes, pero podemos acceder también al control del sonido si lo requerimos.

NOTA: Para acceder a estas partes recordar los métodos de acceso demostrados en el apartado [Objeto](#).



Video 26:

En este video entramos a poner en práctica la teoría que vimos en el video anterior, ya trasladando nuestro objeto carro a líneas de código en Python. A continuación, la explicación del programa:

```
class Coche():  
  
    #Atributos del objeto  
    def __init__(self):  
        #Inicialización de valores  
        self.largoChasis= 250  
        self.anchoChasis=120  
        self.ruedas=4  
        self.enMarcha=False  
  
    #Acciones del objeto  
    def arrancar (self):  
        self.enMarcha=True  
  
    def estado(self):  
        if(self.enMarcha):  
            return "Elcoche esta en marcha"  
        else:  
            return "El coche esta quieto"  
  
#Intancia del objeto  
miCoche=Coche()  
#Acceso a los atributos del objeto  
print("El largo del chasis es : ",miCoche.largoChasis)  
print("El coche tiene ",miCoche.ruedas," ruedas")  
#Acceso a las acciones o metodos del objeto  
miCoche.arrancar()  
#Uso de metodo  
print(miCoche.estado())
```

1. Para crear una clase lo iniciamos con la palabra `class` y el nombre de nuestro objeto con su primera letra en mayúscula, con paréntesis. Al dar ENTER se aplicará una sangría que indique que ya podemos comenzar a trabajar en nuestro objeto.
2. Lo primero que hicimos fue la creación o especificación de los atributos que tiene nuestro objeto:
 - a. Para esto iniciamos un método el cual llevará por nombre “`__init__`” y tendrá como parámetro de entrada “`self`”, esta palabra indica que se le darán atributos y valores a estos atributos.
 - b. Para declarar cada atributo, usamos palabra `self`, luego un punto y el nombre de nuestro atributo. La palabra `self` hace referencia a la misma clase, es como decir “`Coche.nombreDeLaVariable`”.



- c. Luego de hacer la declaración del atributo, con el signo igual le damos valor a nuestra variable.
3. Después de ya tender declarados e inicializados los atributos de nuestra clase podemos crear los métodos que le darán “movimiento” a nuestro objeto. Los métodos que están creados en esta clase siempre recibirán como parámetro de entrada la misma clase, ya que será la que será modificada.
4. En estos métodos podemos tener elementos de retorno o no.
5. Cuando ya hayamos terminado de declarar los atributos y métodos de nuestra clase, para crear un objeto solo debemos darle el nombre a una variable y la igualamos con el nombre de la clase, esta es la instancia de un objeto de la clase, con todas las características que le hayamos definido.
6. Recordemos la nomenclatura para acceder a los métodos y atributos:
 - a. Si queremos acceder a un atributo del objeto solo debemos escribir el nombre del objeto que creamos, un punto, y el nombre del atributo al que queremos acceder.
 - b. Si queremos accionar uno de los métodos, escribimos el nombre de nuestro objeto, punto y el nombre de nuestro método seguido de paréntesis.



Video 27:

En este video seguiremos trabajando sobre las mismas líneas de código del video anterior y tocaremos los temas de encapsulamiento y de métodos que reciben atributos de entrada. Recordemos que el encapsulamiento es la posibilidad de acceder a ciertos elementos de una clase.

Revisemos el código:

```
class Coche():

    #Atributos del objeto
    def __init__(self):
        ##Inicializacion de valores
        self.__largoChasis= 250
        self.__anchoChasis=120
        self.__ruedas=4
        self.enMarcha=False

    #Acciones del objeto
    def arrancar (self,arrancamos):
        self.enMarcha=arrancamos
        if(self.enMarcha):
            return "El coche esta en marcha"
        else:
            return "El coche esta quieto"

    def estado(self):
        print ("El vehiculo tiene ",self.__ruedas," ruedas\nUn ancho de ",
        self.__anchoChasis," cm y un alto de ",self.__largoChasis," cm.")

#Intancia del objeto
print("-----")
miCoche=Coche()
print(miCoche.arrancar(True))
miCoche.estado()
print("-----")
miCoche2=Coche()
print(miCoche2.arrancar(False))
miCoche2.__ruedas = 5
miCoche2.estado()
print("-----")
```

El principio de encapsulamiento en lenguaje Python se aplica por medio de la escritura de dos guiones bajos antes de la variable o método que queramos encapsular.

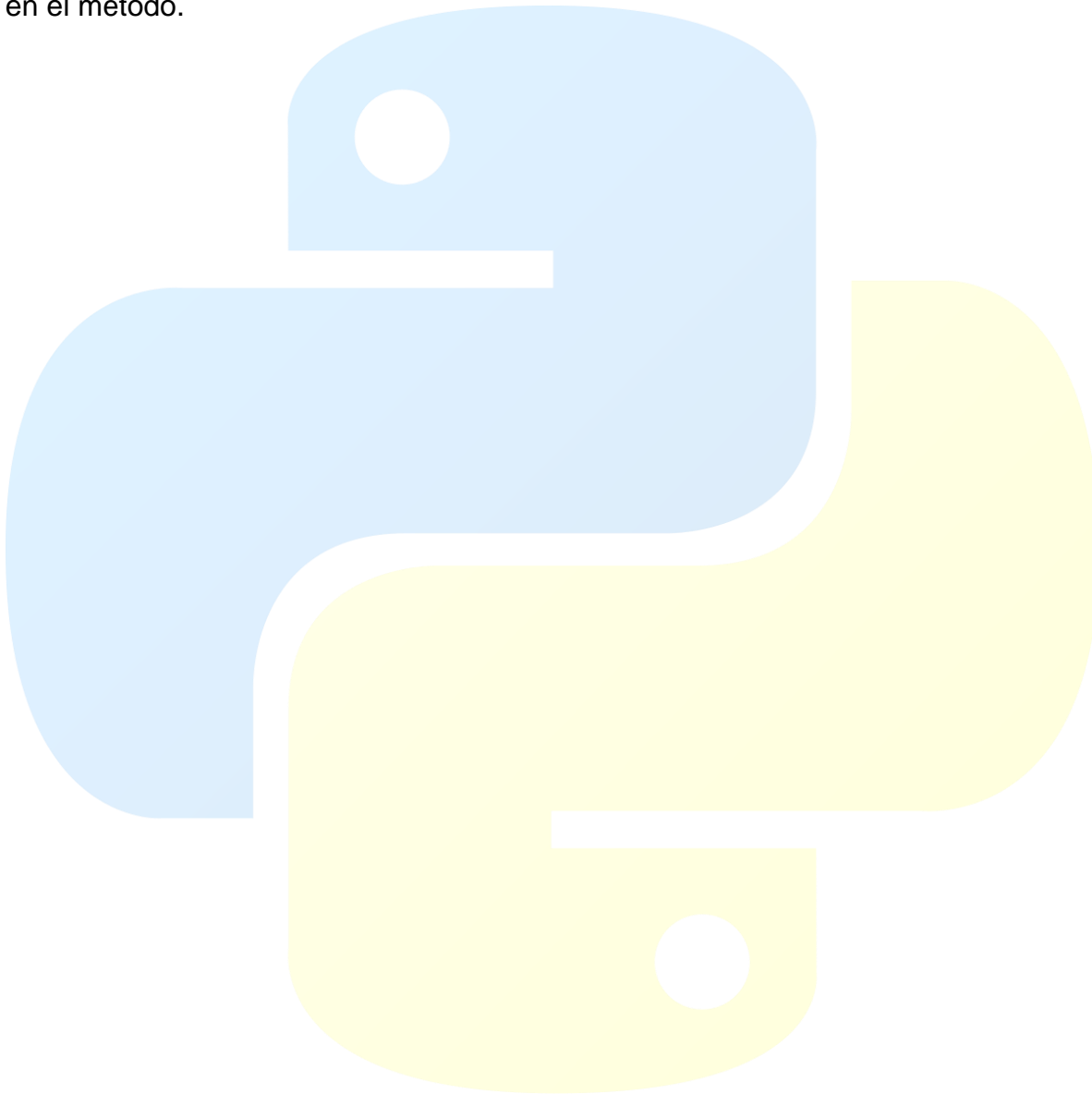
Si por alguna razón se intenta cambiar el valor de un atributo de un objeto, el programa no colapsará, solo ignorará dicha sentencia y seguirá el flujo normal.

El cambio del valor de una variable encapsulada solo se puede hacer desde adentro de la clase.



Por otro lado, cuando queremos darle un parámetro de entrada a un método de una clase, solo debemos ponerle un nombre a esta en los parámetros de entrada y ya adentro del método operaremos con dicho parámetro.

Al momento de llamar el método que recibe un parámetro para su funcionamiento, hay que tener en cuenta que debemos ingresar el mismo tipo de atributo definido en el método.





Video 28:

En este video seguimos trabajando sobre el código y seguimos viendo como podemos usar el encapsulamiento en métodos que pertenecen a una clase.

En este video creamos un método al cual solo tendrá acceso la misma clase cuando lo requiera en otro proceso, es decir que por fuera de la clase es imposible usarla, en este método se crean atributos a los cuales solo se podrá tener acceso al momento de usarlo, y, por otro lado, en el método ARRANCAR se usa el método de CHEQUEO para verificar que todo esté bien. En este video no se hace mayor profundización en el encapsulamiento, solo vemos cómo se puede emplear este en métodos.

```
class Coche():
    #Bloque de atributos
    ##Constructora
    def __init__(self):
        #Inicializacion de valores
        self.largoChasis= 250
        self.anchoChasis=120
        ##Variable encapsulada
        self.__ruedas=4
        ##
        self.enMarcha=False
    #Bloque de acciones
    def arrancar(self,arrancamos):
        self.enMarcha=arrancamos

        if(self.enMarcha == True):
            chequeoI=self.__chequeo()

            if(self.enMarcha and chequeoI == True):
                return "El vehiculo esta en marcha"

            elif (self.enMarcha and chequeoI == False):
                return "Algo ha ido mal"

            else:
                return "El vehiculo esta quieto"

    def __chequeo(self):
        print("Inicio chequeo interno")
        self.gasolina="OK"
        self.aceite="mal"
        self.puertas="cerradas"

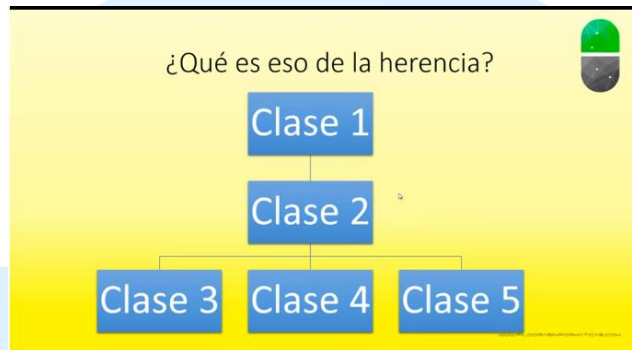
        if(self.gasolina == "OK" and self.aceite == "OK" and self.puertas == "cerradas"):
            return True
        else:
            return False

    def caracterizticas(self):
        print("El vehiculo tiene ",self.__ruedas," ruedas\nUn ancho de ",
            self.anchoChasis," cm y un alto de ",self.largoChasis," cm.")
```

**Video 29:**

En este video trabajaremos lo que es la herencia, que es, para que sirve y cómo podemos implementarla en Python.

Se conoce como herencia el hecho de delegar atributos y métodos de una clase super o padre a otras hijas, estas clases hijas pueden tener atributos y métodos propios, pero compartirán los que contenga su clase padre.



En el video se nos sigue planteando el ejemplo con los vehículos como carros, motos, buses, camiones, bicicletas, etc.



```
class Vehiculos():
    ##Recibe como datos de entrada una marca y un modelo
    def __init__(self,marca,modelo):
        self.marca=marca
        self.modelo=marca
        ##Atributos de condicionamiento
        self.enMarcha=False
        self.acelera=False
        self.frena=False

    def arrancar(self):
        self.enMarcha=True

    def acelerar(self):
        self.acelera=True

    def frenar(self):
        self.frena=True

    def estado(self):
        if self.enMarcha == True:
            em = " Si"
        else:
            em = "No"
        if self.acelera == True:
            ac = " Si"
        else:
            ac = "No"
        if self.frena == True:
            fr = " Si"
        else:
            fr = "No"
        print("*****\nMarca : ",self.marca,
            "\nModelo : ",self.modelo,
            "\n¿En marcha? ",em,"\n¿Acelerando? ",ac,
            "\n¿Frenando? ",fr,"\n*****")
```

Aquí podemos ver como creamos una super-clase “Vehículo” la cual guardara los atributos y métodos que compartirán sus clases hijas. En su constructor vemos como recibirá datos de entrada y tendrá variables de control booleanas. Estas variables serán modificadas por medio de los métodos declarados.

Todo este conjunto de características será heredado por cualquier clase que lo requiera.

```
class Moto(Vehiculos):
    pass

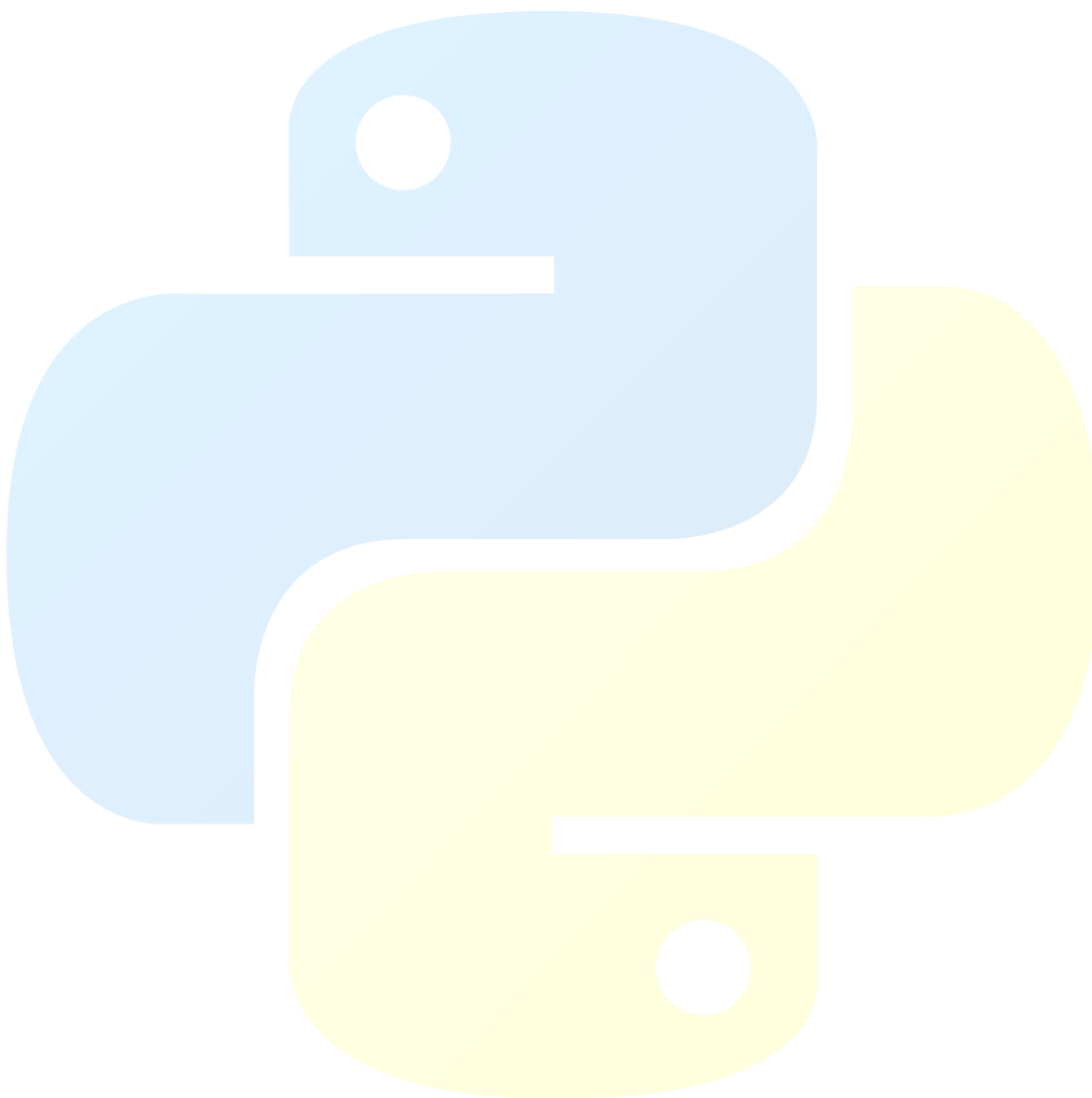
miMoto = Moto("AKT","5G")

miMoto.estado()
```

Como vemos aquí, la clase “Moto” recibe como parámetro de entrada la clase “Vehículos”, esto indica que recibe todos los atributos y métodos que en esta están especificados.



```
*****  
Marca : AKT  
Modelo : AKT  
¿En marcha? No  
¿Acelerando? No  
¿Frenando? No  
*****
```





Video 30:

En este video seguimos trabajando en el tema tan importante como lo es la herencia y entramos a aplicar la condición de como una clase que hereda características de otra puede poseer unas propias. Recordemos como en el video anterior aplicamos una herencia a la clase "Moto" la cual recibía los atributos y métodos de la clase vehículo, ahora aplicamos a la clase "Moto" ciertos cambios.

```
class Moto(Vehiculos):
    hcaballito=""

    def caballito(self):
        hcaballito="*Haciendo caballito*"

    def estado(self):
        if self.enMarcha == True:
            em = " Si"
        else:
            em = "No"
        if self.acelera == True:
            ac = " Si"
        else:
            ac = "No"
        if self.frena == True:
            fr = " Si"
        else:
            fr = "No"
        if self.hcaballito == "":
            self.hcaballito = "No esta haciendo caballito"
        else:
            self.hcaballito = "*Haciendo caballito*"

    print("*****\nMarca : ",self.marca,
          "\nModelo : ",self.modelo,
          "\n¿En marcha? ",em,"\n¿Acelerando? ",ac,
          "\n¿Frenando? ",fr,"\n¿Haciendo caballito?",self.hcaballito,
          "\n*****")
```

Como vemos, a esta clase le creamos un atributo propio de ella, el cual será modificado por un método que solo pertenecerá a esta. La mayor particularidad aquí es la aparición de la sobre-escritura de métodos, la cual es la creación de métodos con el mismo nombre, pero pertenecientes a diferentes clases. Cada método será independiente del otro y tendrá una función distinta.

```
class Furgon(Vehiculos):

    def cargar(self, cargar):
        self.cargado=cargar
        if (self.cargado):
            return "El furgon esta cargado"
        else:
            return "El furgon no esta cargado"
```



Esta clase también se recibe herencia, y además crea un método constructor el cual recibe un parámetro de entrada de tipo booleano.

A continuación, podemos ver el comportamiento de las clases “Furgón” y “Moto”.

```
MOTO
*****
Marca : AKT
Modelo : 5G
¿En marcha? No
¿Acelerando? No
¿Frenando? No
¿Haciendo caballito? No esta haciendo caballito
*****

Marca : AKT
Modelo : 5G
¿En marcha? No
¿Acelerando? No
¿Frenando? No
¿Haciendo caballito? *Haciendo caballito*
*****

FURGON
*****

Marca : Chevrolet
Modelo : xxxxxxxx
¿En marcha? Si
¿Acelerando? No
¿Frenando? No
*****
El furgon esta cargado
>>>
```

Por último, realizamos dos últimas clases para poder demostrar cómo se puede aplicar la herencia doble, la cual se define como el hecho de que una clase reciba herencia de dos clases.

NOTA: al realizar este tipo de herencia, se le dará prioridad a la primera clase especificada. Por lo cual, si la segunda clase requiere atributos de entrada, no podrán ser ingresados y se lanzara un error.

```
class Velectric():
    def __init__(self):
        self.autonomia=100

    def cargarBat(self):
        self.cargando=True

class BiciElect(Velectric,Vehiculos):
    pass
```




Se crea otra super-clase, la cual no recibe ninguna herencia, y posee un método constructor y un comportamiento propio. Luego podemos ver una clase hija la cual recibe a las dos super-clases. Y si al momento de correr queremos usar los métodos de alguna de estas, debemos tener en cuenta el orden en que las especificamos. En el orden en que se ven, se les dará prioridad a las características de la primera super-clase.

```
miBici = BiciElect()
print("BICICLETA")
miBici.estado()

miMoto = Moto("AKT", "5G")
print("MOTO")
miMoto.estado()

miMoto.caballito()

miMoto.estado()

miFurgon = Furgon("Chevrolet", "xxxxxxx")

miFurgon.arrancar()
print("FURGON")
miFurgon.estado()

print(miFurgon.cargar(True))
```

Al momento de ejecutar nuestro código y querer ver su comportamiento en pantalla nos encontraremos de que en las líneas en donde se requieren ciertos atributos, pues que estas están vacías.

```
BICICLETA
Traceback (most recent call last):
  File "herencia1", line 91, in <module>
    miBici.estado()
  File "herencia1", line 22, in estado
    if self.enMarcha == True:
AttributeError: 'BiciElect' object has no attribute 'enMarcha'
```



Video 31:

En este video se va a continuar con el tema de herencia y el uso de la sentencia super (). Para esto vamos a trabajar con un nuevo ejemplo, debido a que el anterior ya se encuentra bien explicado y especificado. En este ejercicio vamos a trabajar con dos clases, persona y empleado, por medio de las cuales una heredara de la otra.

Primero que todo planteamos la clase Persona

```
class Persona():
    def __init__(self,nombre, edad, lugar_residencia):
        self.nombre = nombre

        self.edad = edad

        self.lugar_residencia = lugar_residencia

    def descripcion(self):
        print ("Nombre : ",self.nombre," Edad: ",
              self.edad," Residencia: ",self.lugar_residencia)
```

En esta clase vamos a crear la constructora y un método de impresión de los datos de cada persona.

Luego, independiente de esta clase, creamos la clase Empleado:

```
class Empleado():
    def __init__(self,nombre,edad,lugar_residencia,
                 salario,antigüedad):
        ##super().__init__(nombre,edad,lugar_residencia)
        self.salario = salario
        self.antigüedad = antigüedad

    def descripcion (self):
        ##super().descripcion()
        print ("Salario: ",self.antigüedad," Antigüedad: ",
              self.antigüedad," años")
```

En esta clase vemos como un empleado tiene unos datos que lo hacen diferente a la clase Persona, pero que, a su vez, un empleado tiene datos contenidos en la clase Persona, por lo cual se requiere hacer uso de la constructora y otros métodos de esta clase padre.



```
class Empleado(Persona):
    def __init__(self,nombre,edad,lugar_residencia,
                salario,antigüedad):
        super().__init__(nombre,edad,lugar_residencia)
        self.salario = salario
        self.antigüedad = antigüedad

    def descripcion (self):
        super().descripcion()
        print ("Salario: ",self.antigüedad," Antigüedad: ",
              self.antigüedad," años")
```

Aquí vemos como se hace uso de la sentencia **super()** por medio de la cual tendremos acceso a los métodos de la clase ingresada como padre, la cual es Persona. Con esta sentencia hacemos uso de la constructora la cual nos posibilitara crear empleados con sus datos personales y con los pertenecientes a la empresa. Por otro lado, se llama al método de descripción definido en la clase persona para mostrar los datos ingresados.

```
28 antonio = Empleado ("Antonio",25,"España",1500,15)
29 print ("-----Datos Empleado-----")
30 antonio.descripcion()
```

```
-----Datos Empleado-----
Nombre : Antonio Edad: 25 Residencia: España
Salario: 15 Antigüedad: 15 años
[Finished in 0.5s]
```

Luego se nos plantea el escenario de la herencia múltiple y aparece el termino de **principio de sustitución**, el cual define que una clase hija es siempre una parte de una clase padre. En este caso que un empleado siempre será una persona.

Aquí haremos uso de la sentencia **isinstance()** para comprobar este principio. Esta sentencia retorna un valor booleano que comprobara si pertenece o no a una super-clase.

A continuación probamos el principio de sustitución con el método **isinstance()**:

Validando la instancia de cada uno de los objetos y su conexión con una clase padre, como en el caso del objeto Antonio con su clase super que es persona, y comprobando el principio de sustitución con los objetos creados



```
29 antonio = Empleado ("Antonio",25,"España",1500,15)
30 print ("-----Datos Empleado-----")
31 antonio.descripcion()
32 ##Comprobacion de que el objeto creado si es una instancia
33 print(isinstance(antonio,Persona))
34 print(isinstance(antonio,Empleado))
35 manuel = Persona("Manuel",30,"Colombia")
36 print(isinstance(manuel,Persona))
37 print(isinstance(manuel,Empleado))

-----Datos Empleado-----
Nombre : Antonio Edad: 25 Residencia: España
Salario: 15 Antigüedad: 15 años
True
True
True
False
[Finished in 1.6s]
```



Video 32:

En este video se va a ahondar en un término tan importante como es el polimorfismo, el cual es la habilidad de un objeto de poder cambiar de formas a lo largo de la ejecución del código, explicándonos con el siguiente código

```
▼ class Coche():  
    def desplazamiento(self):  
        print("Me desplazo utilizando cuatro ruedas")  
▼ class Moto():  
    def desplazamiento(self):  
        print("Me desplazo utilizando dos ruedas")  
▼ class Camnion():  
    def desplazamiento(self):  
        print("Me desplazo utilizando seis ruedas")
```

Vemos que creamos tres clases, las cuales representan a tres tipos de vehículos, cada uno con la capacidad de desplazarse con sus recursos:

```
20  
21 miVehiculo = Moto()  
22 miVehiculo.desplazamiento()  
23 miVehiculo2 = Coche()  
24 miVehiculo2.desplazamiento()  
25 miVehiculo3 = Camnion()  
26 miVehiculo3.desplazamiento()  
27  
Me desplazo utilizando dos ruedas  
Me desplazo utilizando cuatro ruedas  
Me desplazo utilizando seis ruedas  
[Finished in 0.3s]
```



Al realizar una instancia de cada una de estas clases vemos el comportamiento de cada uno. Pero, ya que cada una de las clases comparte un comportamiento o actividad que cada uno realiza de una forma diferente, podemos aplicar el principio de polimórficos con un simple método:

```
def desplazamientoVehiculo(vehiculo):  
    vehiculo.desplazamiento()
```



Este método recibe un objeto sea el que sea de las clases creadas anteriormente, y hace llamado al método de desplazamiento que tiene este, esto dándonos la libertad de no tener que llamar a cada uno de los métodos de cada clase, sino que este método, sin importar el tipo de “vehículo” llame al método.

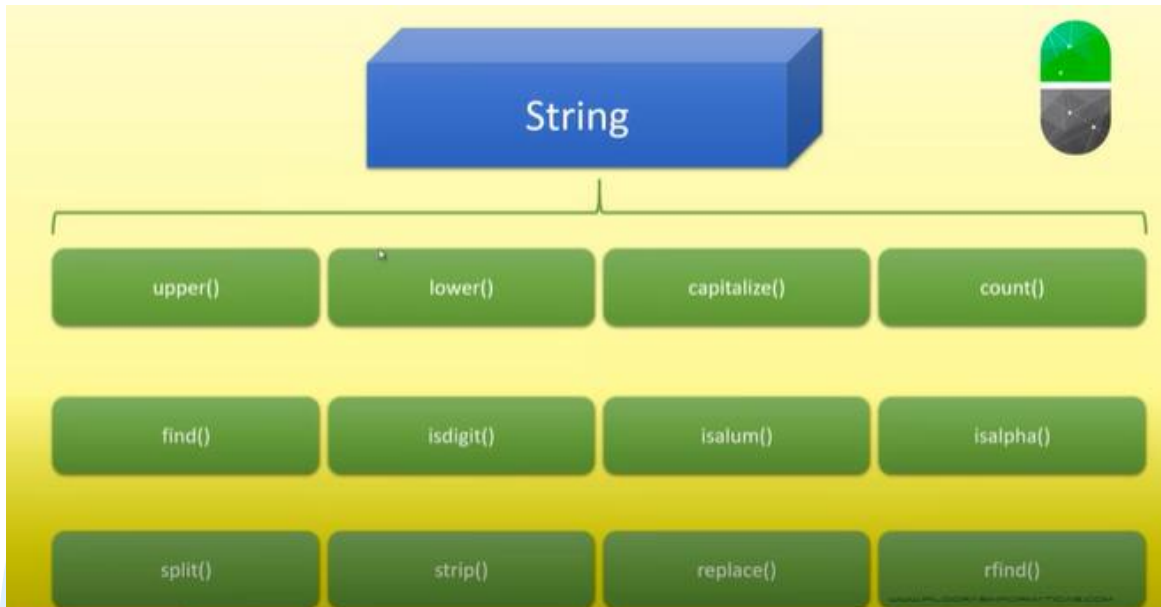
Se nos menciona la flexibilidad que nos brinda Python al ser un lenguaje débilmente tipado

	
Al aplicar el principio de polimorfismo en Python, no es necesario definir el tipo de objeto con el cual se esta realizando, dándonos una libertad y simplicidad al aplicar el principio	Al querer aplicar el principio de polimorfismo en Java, tendremos que contar con el hecho de tener que definir a las clases que vamos a usar como hijas de alguna clase padre, para de este modo, pedir como parámetro un objeto que sea del tipo de la clase padre, o de una forma incorrecta, trabajar con objetos Object, los cuales son ineficientes y una mala práctica al programar.
NOTA: como consejo propio, al usar este principio es recomendado manejar un control de excepciones dado sea el caso de que el objeto ingresado no posea dicho método o comportamiento requerido.	



Video 33:

Como última parte de la programación orientada a objetos, en este video se va a explicar los métodos de manipulación de cadenas, las cuales se consideran como objetos manipulables, lo cual se va a realizar por medio de los siguientes métodos:



- Upper(): este método se encargara de volver en **mayúscula** cada uno de los caracteres alfabéticos de una cadena.
- Lower(): este método se encargara de volver en **minúscula** cada uno de los caracteres alfabéticos de una cadena.
- Capitalize(): este método se encargara de volver en **mayúscula la primera letra de una cadena**.
- Count(): este método se encargara de contar la cantidad de veces que se repite un carácter o una cadena dentro de otra cadena.
- Find(): este método se encargara de retornar el índice o posición en el que se encuentra un carácter o una subcadena dentro de una cadena.
- isdigit(): este método booleano nos retornara si una cadena es o no un valor numérico.
- isalnum(): este método booleano nos retornara si una cadena se compone o no de valores alfanuméricos.
- isalpha(): este método booleano nos retornara si una cadena se compone solamente de caracteres alfabéticos (no cuenta los espacios).
- Split(): este método se encarga de dividir una cadena por medio de los espacios que tenga (es decir, generando subcadenas sin espacios).
- Strip(): método que elimina los espacios del principio y el final de una cadena.
- Replace(): este método se encargara de reemplazar una subcadena por otro.
- Rfind(): Este método es inverso al método find(), ya que retorna el índice que ocupa un carácter dentro de una cadena pero de la última posición hacia atrás.



Para más información sobre los métodos de manipulación de Python, revisar (van Rossum, 2000) en el apartado "4.1 string -- Common string operations", en donde encontrara otros métodos para la manipulación de cadenas en este lenguaje. También puede visitar <http://pyspanishdoc.sourceforge.net/lib/string-methods.html>

A continuación, algunos ejemplos del uso de los métodos de manipulación de cadenas:

```
nombreUsuario = input("Introduce tu nombre de usuario: ")
print ("El nombre es : ",nombreUsuario.upper())
nombreUsuario = input("Introduce tu nombre de usuario: ")
print ("El nombre es : ",nombreUsuario.lower())
nombreUsuario = input("Introduce tu nombre de usuario: ")
print ("El nombre es : ",nombreUsuario.capitalize())
```

Como vemos, se hace uso de métodos como lower() o upper() para mostrar la cadena del nombre del usuario, y también el uso de capitalize() para el nombre del usuario, como se ve a continuación:

```
Introduce tu nombre de usuario: lfvelascot
El nombre es : LFVELASCOT
Introduce tu nombre de usuario: LFEVLASCOT
El nombre es : lfevlascot
Introduce tu nombre de usuario: luis Felipe
El nombre es : Luis felipe
```

Finalmente, se nos plantea un pequeño programa que evalúe la edad de un usuario la cual será ingresada en una cadena de texto, por lo tanto, tenemos que validar que se nos dé un valor numérico y que cumpla con el límite de edad estipulado:

```
edad = input("Ingresa la edad : ")
while (edad.isdigit() == False):
    print ("Por favor introducir un valor numerico")
    edad = input("Ingresa la edad : ")

if (int (edad) < 18):
    print ("no puede pasar")
else :
    print ("si puede pasar")
```

En este código se valida que hasta que el usuario no ingrese un valor numérico no se podrá validar su edad, esto por medio del uso de un bucle while que valida la condición de que si se ingresa un valor numérico con el método isdigit(), al momento de que se cumpla dicha condición, se evaluará el valor ingresado por medio de un casteo del valor ingresado.



```
Ingrese la edad : dfdf
Por favor introducir un valor numerico
Ingrese la edad : dssf
Por favor introducir un valor numerico
Ingrese la edad : cji
Por favor introducir un valor numerico
Ingrese la edad : 19
si puede pasar
```

Ejercicio del video 33:

Crea un programa que pida introducir una dirección de email por teclado. El programa debe imprimir en consola si la dirección de email es correcta o no en función de si esta tiene el símbolo '@'. Si tiene una '@' la dirección será correcta. Si tiene más de una o ninguna '@' la dirección será errónea. Si la '@' está al comienzo de la dirección de email o al final, la dirección también será errónea.

```
correo = input ("Introducir el correo electronico : ")
if (correo.count("@") == 1 and
    (correo.find("@") != 0 and correo.find("@") != len(correo)-1))
    print ("Direccion de correo valida")
else :
    print ("Direccion de correo invalida")
```



Módulos y paquetes

Video 34:

En este video se nos explica todo lo relacionado con los módulos:

MÓDULOS

- ¿Qué son?
 - Un archivo con extensión .py .pyc (Python compilado) o archivo escrito en C para CPython, que posee su propio espacio de nombres y que puede contener variables, funciones, clases e incluso otros módulos.
- ¿Para qué sirven?
 - Para organizar y reutilizar el código (modularización y reutilización) poo

Es resumen, los módulos son el conjunto de archivos .py que permiten la utilización y conexión entre sí del código que cada módulo contiene, como lo vemos en java en donde las distintas clases pueden hacer uso de sus componentes entre sí.

Para esto, creamos en la carpeta donde hemos almacenado todos los ejercicios realizados una nueva carpeta



En esta carpeta tendremos todos los módulos que trabajaremos en esta unidad.

Luego creamos el módulo/archivo funciones_matematicas.py con el siguiente código en su interior

```
def sumar(op1,op2):  
    print ("El resultado de la suma : ",op1," + ",op2," = ",op1+op2)  
  
def restar(op1,op2):  
    print ("El resultado de la suma : ",op1," - ",op2," = ",op1-op2)  
  
def multiplicar(op1,op2):  
    print ("El resultado de la suma : ",op1," x ",op2," = ",op1*op2)
```

Como vemos creamos 3 métodos que usaremos en otro modulo



Luego crearemos un módulo llamado `us_funciones.py` en donde haremos uso de los métodos

```
#import funciones_matematicas

#funciones_matematicas.sumar(1,5)

#funciones_matematicas.restar(1,5)

#funciones_matematicas.multiplicar(10,5)

from funciones_matematicas import *

sumar(1,5)

restar(1,5)

multiplicar(10,5)
```

Veremos dos formas de cómo usar los métodos del otro modulo:

1. En la parte comentada del código vemos el uso de la sentencia **import** la cual nos posibilita hacer uso de TODOS los métodos que se encuentren dentro del módulo, cabe resaltar que es un poco tedioso estar escribiendo la clase y el método que haremos uso.
2. En la parte no comentada, vemos una forma más sencilla y que es la más indicada al momento de trabajar con programas de gran tamaño, en donde por medio de **from nombre:modulo import * o los métodos que usaremos** limitamos el acceso a las funcionalidades del módulo, reduciendo el peso de nuestros programas.

Recordemos el módulo que creamos en donde se explicó la herencia, ese mismo código lo guardamos en un nuevo módulo llamado `modulo_vehiculos.py` donde encapsularemos las funciones del método.

```
class Vehiculos():

    def __init__(self,marca,modelo):
        #Datos ingresados
        self.marca=marca
        self.modelo=modelo
        #Datos por defecto
        self.enMarcha=False
        self.acelera=False
        self.frena=False
```



```
def arrancar(self):
    self.enMarcha=True

def acelerar(self):
    self.acelera=True;

def frenar (self):
    self.frena = True

def estado(self):
    print("Marca : ",self.marca,"\nModelo : ",self.modelo,"\n¿En
Marcha?      ",self.enMarcha,"\n¿Acelera?      ",self.acelera,"\n¿Frena?
",self.frena)

class Moto(Vehiculos):
    hcaballito="*No hace caballito*"
    def caballito(self):
        self.hcaballito="*Haciendo caballito*"

    def estado(self):
        print("Marca : ",self.marca,"\nModelo : ",self.modelo,"\n¿En
Marcha?      ",self.enMarcha,"\n¿Acelera?      ",self.acelera,"\n¿Frena?
",self.frena,"\n ",self.hcaballito)

class Furgoneta (Vehiculos):

    def carga(self, cargar):
        self.cargada=cargar
        if(self.cargada):
            return "La Furgoneta está cargada"
        else:
            return "La furgoneta no está cargada"

class vElectrico():

    def __init__(self):
        self.autonomia =100

    def cargaE(self):
        self.cargado=True

class biciElectrica(vElectrico,Vehiculos):
    pass
```



Luego crearemos el módulo `uso_vehiculos.py` en donde podremos hacer uso de todas las funcionalidades de las clases que definimos previamente

```
1  from modulo_vehiculos import *
2
3  miCoche = Vehiculos("Mazda","MX5")
4  miCoche.estado()
```

El problema es cuando tenemos que hacer uso de funcionalidades en módulos externos a la carpeta creada previamente, lo cual veremos en el próximo video.



Video 35:

En este video se nos explica el concepto y uso de los paquetes en Python y cómo podemos usarlos en nuestros proyectos. Se nos da una breve explicación de que, para que y como se crean los paquetes:

PAQUETES

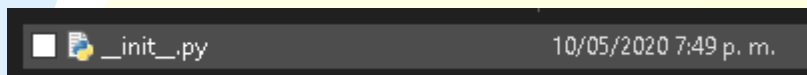
- ¿Qué son?
 - Directorios donde se almacenarán módulos relacionados entre sí.
- ¿Para qué sirven?
 - Para organizar y reutilizar los módulos
- ¿Cómo se crea un paquete?
 - Tan sencillo como crear una carpeta con un archivo `__init__.py`

Después se nos explica cómo crear paquetes:

1. Creamos una carpeta con el nombre del paquete, en este caso se llamará cálculos



2. En el interior de esta carpeta crearemos un archivo con el nombre `__init__.py`



Cada vez que creemos un nuevo paquete o subpaquete, debemos incluir este archivo.

3. Ahora solo es necesario crear los módulos con los que queramos conformar cada uno de los paquetes.

Paquete general

En este paquete tendremos el archivo `__init__.py`, el cual, en el podremos crear un modulo en donde usos los módulos de los subpaquetes o cualquier modulo que necesitemos.

Subpaquete 1 – cálculos básicos

En este paquete crearemos el archivo `__init__.py` y el módulo `operaciones_basicas.py` con el siguiente código:



```
def sumar(op1,op2):  
    print ("El resultado de la suma : ",op1," + ",op2," = ",op1+op2)  
  
def restar(op1,op2):  
    print ("El resultado de la resta : ",op1," - ",op2," = ",op1-op2)  
  
def multiplicar(op1,op2):  
    print ("El resultado de la multiplicacion : ",op1," x ",op2," = ",op1*op2)  
  
def dividir(op1,op2):  
    print ("El resultado de la divicion : ",op1," / ",op2," = ",op1/op2)
```

Estos métodos serán los que usemos en el módulo externo

Subpaquete 2 – redondeo

En este paquete crearemos el archivo `__init__.py` y el módulo `redonde_potencia.py` con el siguiente código

```
def potencia(op1,op2):  
    print ("El resultado de la potencia : ",op1," ^ ",op2," = ",op1**op2)  
  
def redondeo(op1):  
    print ("El resultado de del redondeo de : ",op1," = ",round(op1))
```

Para hacer uso de los módulos creados previamente, en la carpeta donde hemos guardados todos los archivos, crearemos el módulo `uso_paquetes.py` en donde probaremos que todo funcione correctamente:

```
from calculos.basicos.operaciones_basicas import *  
from calculos.redonde_potencia.redondea_potencia import *  
  
dividir (4,6)  
##  
redondeo(1.9)
```

Aquí cabe resaltar que tenemos que conocer bien la dirección en donde se ubica cada módulo, para hacer uso de este, y además saber si vamos a hacer uso de todas sus funcionalidades o solo de algunas.



Video 36:

En este video se nos explicara como hacer uso de los paquetes distribuibles, los cuales son aquellos paquetes que podemos enviar por cualquier medio para ser usado en otros equipos.



Para esto debemos crear el paquete e instalarlo.

Al distribuir un paquete podremos hacer uso de este sin importar en donde se encuentre en nuestro Equipo, sea en la carpeta raíz donde fue creado o en cualquier otro directorio o computador.

Para esto hay que seguir los siguientes pasos:

1. En el directorio raíz o donde hemos almacenado todos nuestros ejercicios vamos a crear el archivo setup.py el cual contendrá el siguiente código:

```
from setuptools import setup
setup(

    name="paquetecalculos",
    version="1.0",
    description="Paquete de redondeo y potencia",
    author="Felipe",
    author_email="luisfelipevelascotao@gmail.com",
    packages=["calculos", "calculos.redonde_potencia"]
)
```

Aquí tendremos que poner:

- Name: el nombre del paquete sin espacios



- Versión: si nosotros vamos actualizando o modificando nuestros paquetes, podremos ir actualizando este valor
- Author: aquí ponemos el nombre de quien haya creado el paquete.
- Author_email: aquí podemos añadir el email de quien creo el paquete.
- Packages: debemos poner el directorio donde se encuentra el paquete que vamos a distribuir.

Luego de haber definido estos datos, guardamos.

2. Nos dirigimos a la ventana de comandos de nuestro sistema operativo, y nos desplazaremos hasta llegar al directorio donde creamos el archivo setup.py

```
C:\Users>cd..  
  
C:\>E:  
  
E:\>cd USBF  
  
E:\USBF>cd PYTHON LOG
```

3. Al llegar al directorio donde tenemos el archivo setup.py ejecutaremos la siguiente línea:

```
E:\USBF\PYTHON LOG>python setup.py sdist  
  
running sdist  
running egg_info  
writing paquetecalculos.egg-info\PKG-INFO  
writing          dependency_links          to          paquetecalculos.egg-  
info\dependency_links.txt  
writing top-level names to paquetecalculos.egg-info\top_level.txt  
reading manifest file 'paquetecalculos.egg-info\SOURCES.txt'  
writing manifest file 'paquetecalculos.egg-info\SOURCES.txt'  
warning: sdist: standard file not found: should have one of README,  
README.rst, README.txt, README.md  
  
running check  
warning: check: missing required meta-data: url  
  
creating paquetecalculos-1.0  
creating paquetecalculos-1.0\calculos  
creating paquetecalculos-1.0\calculos\redonde_potencia  
creating paquetecalculos-1.0\paquetecalculos.egg-info  
copying files to paquetecalculos-1.0...
```



```
copying setup.py -> paquetecalculos-1.0
copying calculos\__init__.py -> paquetecalculos-1.0\calculos
copying calculos\redonde_potencia\__init__.py -> paquetecalculos-
1.0\calculos\redonde_potencia
copying      calculos\redonde_potencia\redondea_potencia.py      ->
paquetecalculos-1.0\calculos\redonde_potencia
copying      paquetecalculos.egg-info\PKG-INFO      ->      paquetecalculos-
1.0\paquetecalculos.egg-info
copying      paquetecalculos.egg-info\SOURCES.txt      ->      paquetecalculos-
1.0\paquetecalculos.egg-info
copying      paquetecalculos.egg-info\dependency_links.txt      ->
paquetecalculos-1.0\paquetecalculos.egg-info
copying      paquetecalculos.egg-info\top_level.txt      ->      paquetecalculos-
1.0\paquetecalculos.egg-info
Writing paquetecalculos-1.0\setup.cfg
creating dist
Creating tar archive
removing 'paquetecalculos-1.0' (and everything under it)
```

4. Después de que aparezca que se ha generado el paquete distribuible encontraremos en nuestro directorio raíz las siguientes carpetas (las que están seleccionadas):

<input checked="" type="checkbox"/>	dist	11/05/2020 1:02 p. m.	Carpeta de archivos
<input type="checkbox"/>	modulos	10/05/2020 5:43 p. m.	Carpeta de archivos
<input checked="" type="checkbox"/>	paquetecalculos.egg-info	11/05/2020 1:02 p. m.	Carpeta de archivos

- En la carpeta dist encontraremos un archivo comprimido que podremos compartir

	paquetecalculos-1.0.tar.gz	11/05/2020 1:02 p. m.	Archivo WinRAR	2 KB
--	----------------------------	-----------------------	----------------	------

- En la segunda carpeta se encuentra la información del paquete creado.
5. Si queremos que ese paquete se pueda usar en nuestro equipo o recibimos un paquete que requerimos instalar debemos ir hasta la ruta donde se encuentra el comprimido:

```
E:\USBF\PYTHON LOG>cd dist
```

Luego ejecutaremos la siguiente línea:

```
E:\USBF\PYTHON LOG\dist>pip3 install paquetecalculos-1.0.tar.gz
```

```
Processing e:\usbf\python log\dist\paquetecalculos-1.0.tar.gz
```



```
Installing collected packages: paquetecalculos
  Running setup.py install for paquetecalculos ... done
Successfully installed paquetecalculos-1.0
WARNING: You are using pip version 19.2.3, however version 20.1 is
available.
You should consider upgrading via the 'python -m pip install --
upgrade pip' command.
```

Después de esto, podremos ya hacer uso del paquete, especificando la ruta requerida del módulo a usar:

```
from calculos.redonde_potencia.redondea_potencia import *
redondeo(1.9)
```

6. Si requerimos desinstalar un paquete solo debemos ejecutar la siguiente línea de código en la ventana de comandos (no es necesario ir al directorio raíz)

- 7.

```
E:\USBF\PYTHON LOG\dist>pip3 uninstall paquetecalculos

Uninstalling paquetecalculos-1.0:
  Would remove:
    c:\users\compured\appdata\local\programs\python\python38-
32\lib\site-packages\calculos\*
    c:\users\compured\appdata\local\programs\python\python38-
32\lib\site-packages\paquetecalculos-1.0-py3.8.egg-info
Proceed (y/n)? y
  Successfully uninstalled paquetecalculos-1.0
```



Archivos externos

Video 37:

En este video se nos introduce al manejo de archivos externos, los cuales posibilitan el almacenamiento de la información y que esta pueda ser reutilizada en un futuro en que volvamos a abrir un programa podamos acceder a la información previamente indexada. Los archivos que podemos manipular son los archivos generales como los archivos binarios, planos y sus subcategorías y el acceso a las bases de datos:

ARCHIVOS EXTERNOS

- Objetivo: persistencia de datos
- Dos alternativas :
 - Manejo de archivos externos
 - Trabajo con BBDD

The diagram illustrates the concept of external files. A central laptop icon is connected by blue arrows to a file icon labeled 'LOG' and a MySQL database icon. The file icon shows a sample log entry with fields like 'id', 'fecha', 'hora', 'usuario', 'password', 'telefono', 'email', 'direccion', 'comentarios', and 'estado'.

En esta sección del curso se hace enfoque en la manipulación de los archivos externos y las etapas que se cumplen en este procedimiento:

MANEJO ARCHIVOS EXTERNOS

- Fases necesarias para guardar información en archivos externos

The flowchart shows four orange rounded rectangular boxes arranged horizontally, connected by a large light blue arrow pointing from left to right. The boxes are labeled 'Creación', 'Apertura', 'Manipulación', and 'Cierre'. Red arrows point to each box from above.



Las fases de la manipulación de archivos se ejemplifican en el ejemplo trabajado:

1. Creación – Apertura: estas dos primeras etapas se pueden configurar como uno solo ya que esto depende de la existencia del archivo en el directorio, esto lo realizamos de la siguiente forma:

```
from io import open  
  
archivo_texto = open ("archivo.txt","w")
```

Para la manipulación de archivos se requiere importar de la librería **io**, la cual nos posibilita el acceso a archivos externos, el método **open** para el acceso a archivos planos.

Creamos un objeto con el nombre del archivo, en este almacenaremos el archivo el cual abriremos con el método **open**, que recibe como parámetros el nombre del archivo (si el archivo no existe, el método se encargara de crearlo) y el modo en que se abre el archivo (cada uno en minúscula):

- W: modo de solo escritura.
 - R: modo de solo lectura del archivo.
 - A: modo de adición al archivo.
2. Manipulación: en esta fase es en la cual se realiza alguna acción sobre el archivo, algunas de las acciones que podemos realizar según el modo de apertura son:
 - a. Modo de lectura
 - Read(): este método permite una lectura total de archivo, tal cual como se encuentre estructurado.
 - Readlines(): este método nos retornara el archivo en forma de una lista, la cual almacenara en cada posición una línea del archivo.
 - b. Modo de escritura o adición:
 - Write("texto"): este método añadirá el texto que se le dé como parámetro al archivo
 3. Cierre: En esta fase se hace un cierre del fichero, esto se realiza con el siguiente comando:

```
archivo_texto.close()
```

Si se van a realizar diferentes acciones sobre el archivo, se recomienda cerrar el archivo después de realizar cada una



```
from io import open
archivo_texto = open ("archivo.txt","w")
frase ="Estupendo día para estudiar Python \n H O Y"
archivo_texto.write(frase);
archivo_texto.close()
archivo_texto = open ("archivo.txt","r")
texto = archivo_texto.read ()
archivo_texto.close()
print(texto)
archivo_texto = open ("archivo.txt","r")
texto = archivo_texto.readlines ()
archivo_texto.close()
print(texto[0])
archivo_texto = open ("archivo.txt","a")
archivo_texto.write ("\n siempre es una buena ocacion para estudiar python")
archivo_texto.close()
```

Como se observa, se abre el archivo en sus tres modalidades. Recordar que es muy importante corroborar la existencia del archivo en el fichero raíz y el nombre de este, así no se generaran duplicados o errores en la manipulación del archivo

Video 38:

En este video veremos el posicionamiento del curso sobre un archivo de testo, esto viendo el uso del método **seek(i)** el cual nos ayuda a ubicar el cursor que recorrerá el cursor en una posición de partida i:

```
from io import open
archivo_texto = open ("archivo.txt","r")
archivo_texto.seek(11)
print(archivo_texto.read ())
archivo_texto.close()
```

Aquí le damos la orden para que el cursor inicie la lectura desde el punto $i = 11$, cabe resaltar que, si en nuestro código deseamos realizar dos veces lectura del mismo fichero, lo recomendable es, antes siempre de la lectura, indicar la posición del cursor en el inicio:

```
archivo_texto.seek(0)
```

Se nos especifica el uso del método **read(i)** en donde el valor de i será el limitante o la posición final que tendrá el cursor al realizar la lectura del archivo:

```
archivo_texto = open ("archivo.txt","r")
#Se detiene en la posicion indicada
print(archivo_texto.read (11))
archivo_texto.close()
```

Aquí delimitamos como posición final del curso $i = 11$, lo cual solo nos dará los 11 primeros caracteres de nuestro código.



Ya al momento se conocer el manejo de estos dos métodos, podemos comenzar a manipular los archivos de la forma que necesitemos:

```
archivo_texto = open ("archivo.txt","r")
archivo_texto.seek(len(archivo_texto.read())/2)
print("\n",archivo_texto.read ())
archivo_texto.close()
```

Aquí eliminamos el inicio de lectura del archivo desde la mitad de su extensión haciendo uso del método **len("archivo")** en donde le damos como parámetro de entrada el archivo de texto leído, y el valor de este lo dividimos en 2.

```
archivo_texto = open ("archivo.txt","r+") #r+ = lectura y escritura
lista_texto = archivo_texto.readlines()
lista_texto[2] = "Esta línea ha sido incluida desde el exterior\n"
archivo_texto.seek(0)
archivo_texto.writelines(lista_texto);
archivo_texto.seek(0)
print(archivo_texto.read ())
archivo_texto.close()
```

En este último ejemplo podemos ver un nuevo parámetro en la apertura de nuestro archivo

```
archivo_texto = open ("archivo.txt","r+") #r+ = lectura y escritura
```

El parámetro de modo de acceso **"r+"** abre el archivo en modo de lectura y escritura, esto posibilitando realizar ambas acciones, sin requerir abrir y cerrar el archivo para realizarlas independientemente.

NOTA: la importancia de cada uno de estos modos de acceso a los archivos cobra importancia en el momento en que queremos delimitar la actividad que queramos realizar sobre el archivo, esto con fines de seguridad

Después de entrar en este modo, capturamos el contenido de nuestro archivo en una lista por medio del método **readlines()** el cual, como indicamos previamente, nos retorna una lista con las líneas del archivo. Después de esto, insertamos una línea de texto por medio, como lo vemos en la línea 3, después de esto, posicionamos el cursor de inicio en la posición 0 con el método **seek(0)**, luego, haremos uso de un nuevo método, el cual es **writelines(lista)** el cual recibe como parámetro de entrada una lista de Strings, esto posibilitando la actualización de cada una de las líneas del archivo.

Finalmente, hacemos una verificación de la modificación leyendo el archivo y cerrándolo.



Video 39:

En este video se habla acerca de la serialización de archivos, lo cual es el proceso de creación y lectura de archivos codificados, en los cuales se almacena la información de forma binaria, todo esto dándonos varias ventajas como lo es el acceso a la información y la facilidad de distribución que tiene estos archivos:

The screenshot shows a video player interface. At the top, the title "¿Qué es la serialización?" is displayed. Below the title, there is a code editor window on the left showing Python code for two classes, `Persona` and `Empleado`, with methods for serialization and deserialization. To the right of the code editor, the text "Código binario" is shown above a binary string: `111001010010101011110110101011010101010`. A small icon of a computer monitor is visible in the top right corner of the video frame.

Recordemos que en los videos previos hemos manejados archivos planos, en los cuales se almacenan datos en formato de texto, en este caso, vamos a generar también archivos que contendrán la información en forma de binarios, la cual solo podrá ser extraída o manipulada por módulos o programas que creemos.

Para la serialización de archivos, debemos trabajar con la librería **pickle**, la cual nos permite realizar este proceso, de esta en este video trabajamos los siguientes métodos:

- **Dump():** este método lo que hace es el volcado o escritura de los archivos binarios
- **Load():** este método permite la extracción o carga del fichero externo

A continuación, el código de explicación:

```
import pickle

lista_nombres = ["Pedro", "Ana", "Maria", "Isabel"]
fichero_binario = open("lista_nombres", "wb")
pickle.dump(lista_nombres, fichero_binario)
fichero_binario.close()
del(fichero_binario)
```

En este ejemplo realizamos la escritura de un archivo serializado, en donde hacemos uso de un método de escritura binario ("**wb**"), denotado en el segundo parámetro del método **open()**, el cual recibe como parámetros el nombre del archivo y la modalidad de apertura. Previamente se creó una lista de que servirá como objeto a ser serializado, esta lista será



usada en la línea 5, donde se hace uso de la librería para llamar al método **dump(object, fichero)** el cual recibiera como parámetros de entrada el objeto a ser volcado en el fichero y el fichero sobre el cual se realizara esta acción, finalmente cerramos el fichero y eliminamos el fichero temporal que creamos.

```
import pickle

fichero = open("lista_nombres","rb")
lista = pickle.load(fichero)
print(lista)
fichero.close()
```

En este último ejemplo realizaremos la lectura del archivo previamente creado, aquí abriremos el archivo en modo de lectura binaria ("**rb**"), para almacenar el contenido creamos una lista, en la cual almacenaremos el resultado del método **load (fichero)** el cual retornara una lista con el contenido de las filas serializadas en el código, finalmente para comprobar el resultado imprimiremos la lista y cerramos el fichero.



Video 40:

En el anterior video trabajamos la serialización de colecciones las cuales contienen cadenas, en este video ahora realizaremos la serialización de objetos, por lo cual haremos uso de la clase Vehículos previamente creada:

```
class Vehiculo():
    def __init__(self,marca,modelo):
        self.marca=marca
        self.modelo=modelo
        self.enMarcha=False
        self.acelera=False
        self.frena=False

    def arrancar(self):
        self.enMarcha=True

    def acelerar(self):
        self.acelera=True;

    def frenar (self):
        self.frena = True

    def estado(self):
        print("Marca : ",self.marca,"\nModelo : ",self.modelo,
              "\n¿En Marcha? ",self.enMarcha,"\n¿Acelera? ",
              self.acelera,"\n¿Frena? ",self.frena,"\n")
```

Creemos tres instancias de esta clase con los siguientes valores:

```
coche1 = Vehiculo("Mazda","MXS")
coche2 = Vehiculo("Seat","Leon")
coche3 = Vehiculo("Renault","Magane")
```

Estos objetos los incluimos en una lista:

```
coches = [coche1,coche2,coche3]
```

Después de esto, realizaremos la serialización de esta lista como lo hicimos anteriormente:

```
import pickle
fichero = open ("losCoches","wb")
pickle.dump(coches,fichero)
fichero.close()
del(fichero)
```



Como vemos, se realiza la carga de la lista de objetos, de forma regular, luego de esto, podemos comprobar el contenido del fichero previamente creado de la siguiente forma:

```
fichero = open ("losCoches", "rb")
lista = pickle.load(fichero)
fichero.close()
for i in lista:
    i.estado()
```

Aquí creamos una lista, en donde se realizara el volcado de una colección de objetos serializados, luego, recorreremos esta lista por medio de un ciclo For y revisando el estado de los objetos por medio del metodo estado() de la clase vehiculos.

NOTA:

Para realizar la lectura del archivo se requiere tener la clase en donde se especifica el modelo del objeto, ya que, sin este no se podra generar el correcto volcado de la lista y su posterior lectura.



Video 41:

En este último video de la sección de archivos externos se trabajo un ejemplo en donde se recopila todo lo visto en la sección, haciendo uso de una nueva clase y creando en estos métodos los cuales utilicen los métodos de la librería **pickle** para el indexado de datos en los archivos:

```
import pickle

class Persona ():

    def __init__ (self,nombre, genero, edad):
        self.nombre = nombre
        self.genero = genero;
        self.edad = edad
        print("Se ha creado una persona nueva con el nombre :",self.nombre)

    def __str__ (self):
        return "{} - {} - {}".format(self.nombre,self.genero,self.edad)

p = Persona("Sandra","Femenino",29)
p1 = Persona("Andres","Masculino",28)
p2= Persona("Felipe","Maculino",47)
```

Primero que todo, importamos la librería pickle para la manipulación de los archivos, luego creamos la clase que nos da la estructura de los objetos para almacenar en el archivo, acompañados del método de escritura de objetos.

Luego realizamos 3 instancias de la clase, con datos aleatorios de personas para ser usados en el ejemplo.



```
class ListaPersonas:
    personas = []

    def __init__(self):
        listaDePersonas = open("ficheroExterno", "ab+")
        listaDePersonas.seek(0)
        try:
            self.personas = pickle.load(listaDePersonas)
            print("Se han cargado {} personas del fichero externo".format(len(self.personas)))
        except:
            print("El fichero esta vacio")
        finally:
            listaDePersonas.close()
            del(listaDePersonas)

    def agregarPersonas(self, p):
        self.personas.append(p)
        self.guardarPersonasEnFicheroExterno()

    def mostrarPersonas(self):
        for i in self.personas:
            print(p)

    def guardarPersonasEnFicheroExterno(self):
        listaDePersonas = open("ficheroExterno", "wb")
        pickle.dump(self.personas, listaDePersonas)
        listaDePersonas.close()
        del(listaDePersonas)
```

Ahora, creamos una clase, en la cual se generará, como estructura de datos, una lista. En la constructora de esta clase tendremos la creación – apertura del archivo, aquí realizaremos el volcado de los objetos contenidos en el archivo en la lista, y le mostramos al usuario la cantidad de registros en el fichero.

Aparte, tenemos el método de agregar personas, el cual añade una persona a la lista, y luego llama al método de guardar personas en el archivo, en donde se reescribe el contenido del fichero para preservar los cambios realizados en la lista, finalmente encontramos el método que muestra la lista.

```
mislista = ListaPersonas()
#mislista.agregarPersonas(p)
#mislista.agregarPersonas(p1)
#mislista.agregarPersonas(p2)
mislista.mostrarPersonas()
```

Por último, probamos todo creando un objeto de tipo **ListaPersonas()**, y como vemos en la parte comentada, insertamos objetos de tipo persona, como ya se realizó previamente, solo hacemos lectura de los objetos insertados previamente.



Interfaces graficas

Video 42:

En este video iniciamos una nueva y la última sección del curso, en donde se hará manejo de interfaces graficas. Para la creación de interfaces graficas nos podemos apoyar en distintas librerías como las siguientes:

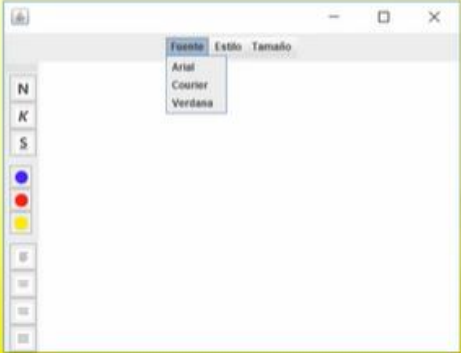
- Tkinter
- WxPython
- PyQt
- PyGTK



Esta librería viene integrada con Python en su instalación, por lo que no es necesario realizar nada para verificar su existencia en el sistema.

Interfaces gráficas

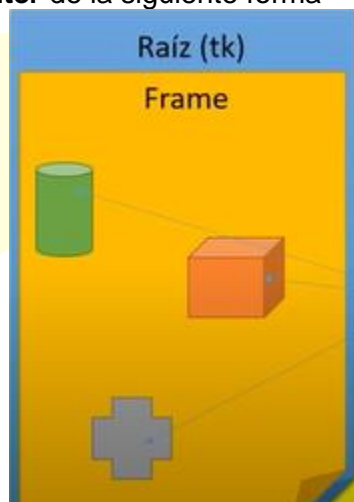
- También denominadas GUI, son intermediarios entre el programa y el usuario. Formadas por un conjunto de gráficos como ventanas, botones, menús, casillas de verificación etc.
- Librerías con las que trabajar para crear GUI:
 - Tkinter
 - WxPython
 - PyQt
 - PyGTK
- Tkinter es un "puente" entre Python y la librería TCL/Tk



Se nos explica la composición de una ventana creada con **Tkinter** de la siguiente forma

1. Raíz (tk): componente base o ventana en donde se van a posicionar los componentes de la ventana
2. Frame: Panel en donde se van a posicionar los componentes de la ventana, este también se configura como un Widget.
3. Widgets: componentes con los que el usuario tendrá interacción.

En este video se nos explica algunos métodos de manipulación de la raíz:





```
from tkinter import *

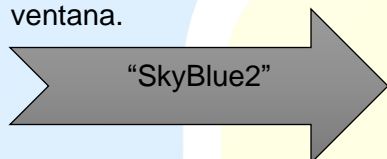
raiz = Tk()

raiz.title("Ventana de pruebas")
raiz.resizable(1,1)
raiz.iconbitmap("logo.ico")
raiz.geometry("650x450")
raiz.config(bg="blue")

raiz.mainloop()
```

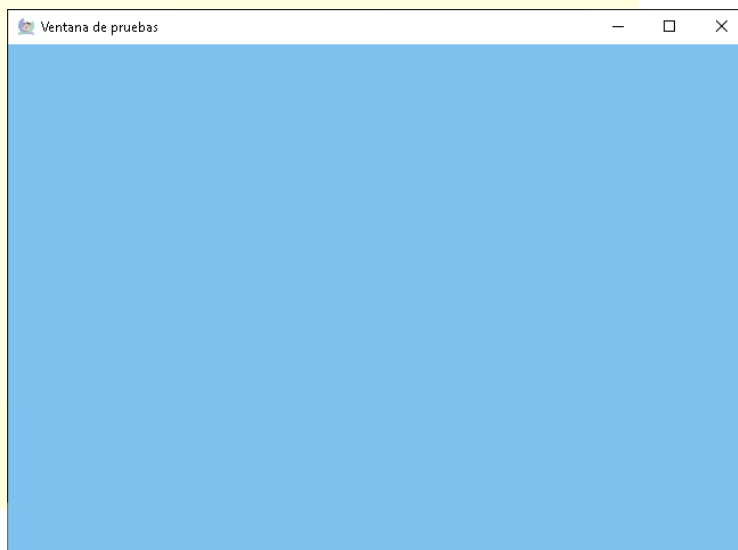
Primero que todo importamos todos los componentes de la librería Tkinter, luego creamos una raíz como lo vemos en la línea 3. De la línea 5 a la 9 se encuentran métodos de manipulación del aspecto de la raíz:

- **r.title("xxxxx")**: con este método le asignamos un título a la ventana.
- **r.resizable(boolean,boolean)**: este método determina la capacidad de que la ventana pueda ser redimensionada, siendo el primer parámetro para el ancho y el segundo para la altura de la ventana.
- **r.iconbitmap("logo.ico")**: este método requiere que, en el directorio en que creamos nuestro modulo tengamos un archivo .ico, el cual corresponde al icono de la parte superior de la ventana.
- **r.geometry("anchoxalto")**: este método le dará el tamaño inicial a la ventana, por lo que requiere una cadena en formato "**anchoxalto**" con valores numéricos para darle el tamaño a la ventana.
- **r.config(bg = "color en inglés / RBG")**: este método en específico (ya que el método .config() permite realizar varias operaciones) le da un color al fondo de la ventana.



NOTA: el uso de colores en Tkinter puede realizarse con los colores en inglés, como los que encontrara en la siguiente imagen, o haciendo uso del código de colores RGB el cual puede consultar en la siguiente página: <https://htmlcolorcodes.com/es/selector-de-color/> (Moe, 2020)

NOTA: para no abrir la consola al ejecutar una ventana, guardar el módulo con extensión. **pyw**.



Para más información sobre Tkinter ir a <https://guia-tkinter.readthedocs.io/es/develop/> (Alvarez, 2015)

snow	deep sky blue	gold	seashell3	SlateBlue2	LightBlue3	SpringGreen2	DarkGoldenrod1	brown4	pink3	purple1	gray26	gray64
ghost white	sky blue	light goldenrod	seashell4	SlateBlue3	LightBlue4	SpringGreen3	DarkGoldenrod2	salmon1	pink4	purple2	gray27	gray65
white smoke	light sky blue	goldenrod	AntiqueWhite1	SlateBlue4	LightCyan2	SpringGreen4	DarkGoldenrod3	salmon2	LightPink1	purple3	gray28	gray66
gainsboro	steel blue	dark goldenrod	AntiqueWhite2	RoyalBlue1	LightCyan3	green2	DarkGoldenrod4	salmon3	LightPink2	purple4	gray29	gray67
floral white	light steel blue	rosy brown	AntiqueWhite3	RoyalBlue2	LightCyan4	green3	RosyBrown1	salmon4	LightPink3	MediumPurple1	gray30	gray68
old lace	light blue	indian red	AntiqueWhite4	RoyalBlue3	PaleTurquoise1	green4	RosyBrown2	LightSalmon2	LightPink4	MediumPurple2	gray31	gray69
linen	powder blue	saddle brown	bisque2	RoyalBlue4	PaleTurquoise2	chartreuse2	RosyBrown3	LightSalmon3	PaleVioletRed1	MediumPurple3	gray32	gray70
antique white	pale turquoise	sandy brown	bisque3	blue2	PaleTurquoise3	chartreuse3	RosyBrown4	LightSalmon4	PaleVioletRed2	MediumPurple4	gray33	gray71
papaya whip	dark turquoise	dark salmon	bisque4	blue3	PaleTurquoise4	chartreuse4	IndianRed1	orange2	PaleVioletRed3	thistle1	gray34	gray72
blanched almond	medium turquoise	salmon	PeachPuff2	DodgerBlue2	CadetBlue1	OliveDrab1	IndianRed2	orange3	PaleVioletRed4	thistle2	gray35	gray73
bisque	turquoise	light salmon	PeachPuff3	DodgerBlue3	CadetBlue2	OliveDrab2	IndianRed3	orange4	maroon1	thistle3	gray36	gray74
peach puff	cyan	orange	PeachPuff4	DodgerBlue4	CadetBlue3	OliveDrab4	IndianRed4	DarkOrange1	maroon2	thistle4	gray37	gray75
navajo white	light cyan	dark orange	NavajoWhite2	SteelBlue1	CadetBlue4	DarkOliveGreen1	sienna1	DarkOrange2	maroon3		gray38	gray76
lemon chiffon	cadet blue	coral	NavajoWhite3	SteelBlue2	turquoise1	DarkOliveGreen2	sienna2	DarkOrange3	maroon4		gray39	gray77
mint cream	medium aquamarine	light coral	NavajoWhite4	SteelBlue3	turquoise2	DarkOliveGreen3	sienna3	DarkOrange4	VioletRed1	gray40	gray78	
azure	aquamarine	tomato	LemonChiffon2	SteelBlue4	turquoise3	DarkOliveGreen4	sienna4	coral1	VioletRed2	gray41	gray79	
alice blue	dark green	orange red	LemonChiffon3	DeepSkyBlue2	turquoise4	khaki1	burlywood1	coral2	VioletRed3	gray42	gray80	
lavender	dark olive green	red	LemonChiffon4	DeepSkyBlue3	cyan2	khaki2	burlywood2	coral3	VioletRed4	gray43	gray81	
lavender blush	dark sea green	hot pink	cornsilk2	DeepSkyBlue4	cyan3	khaki3	burlywood3	coral4	magenta2	gray44	gray82	
misty rose	sea green	deep pink	cornsilk3	SkyBlue1	cyan4	khaki4	burlywood4	tomato2	magenta3	gray45	gray83	
dark slate gray	medium sea green	pink	cornsilk4	SkyBlue2	DarkSlateGray1	LightGoldenrod1	wheat1	tomato3	magenta4	gray46	gray84	
dim gray	light sea green	light pink	ivory2	SkyBlue3	DarkSlateGray2	LightGoldenrod2	wheat2	tomato4	orchid1	gray47	gray85	
slate gray	pale green	pale violet red	ivory3	SkyBlue4	DarkSlateGray3	LightGoldenrod3	wheat3	OrangeRed2	orchid2	gray48	gray86	
light slate gray	spring green	maroon	ivory4	LightSkyBlue1	DarkSlateGray4	LightGoldenrod4	wheat4	OrangeRed3	orchid3	gray49	gray87	
gray	lawn green	medium violet red	honeydew2	LightSkyBlue2	aquamarine2	LightYellow2	tan1	OrangeRed4	orchid4	gray50	gray88	
light grey	medium spring green	violet red	honeydew3	LightSkyBlue3	aquamarine4	LightYellow3	tan2	red2	plum1	gray51	gray89	
midnight blue	green yellow	medium orchid	honeydew4	LightSkyBlue4	DarkSeaGreen1	LightYellow4	tan4	red3	plum2	gray52	gray90	
navy	lime green	dark orchid	LavenderBlush2	SlateGray1	DarkSeaGreen2	yellow2	chocolate1	red4	plum3	gray53	gray91	
cornflower blue	yellow green	dark violet	LavenderBlush3	SlateGray2	DarkSeaGreen3	yellow3	chocolate2	DeepPink2	plum4	gray54	gray92	
dark slate blue	forest green	blue violet	LavenderBlush4	SlateGray3	DarkSeaGreen4	yellow4	chocolate3	DeepPink3	MediumOrchid1	gray55	gray93	
slate blue	olive drab	purple	MistyRose2	SlateGray4	SeaGreen1	gold2	firebrick1	DeepPink4	MediumOrchid2	gray56	gray94	
medium slate blue	dark khaki	medium purple	MistyRose3	LightSteelBlue1	SeaGreen2	gold3	firebrick2	HotPink1	MediumOrchid3	gray57	gray95	
light slate blue	khaki	thistle	MistyRose4	LightSteelBlue2	SeaGreen3	gold4	firebrick3	HotPink2	MediumOrchid4	gray58	gray96	
medium blue	pale goldenrod	snow2	azure2	LightSteelBlue3	PaleGreen1	goldenrod1	firebrick4	HotPink3	DarkOrchid1	gray59	gray97	
royal blue	light goldenrod yellow	snow3	azure3	LightSteelBlue4	PaleGreen2	goldenrod2	brown1	HotPink4	DarkOrchid2	gray60	gray98	
blue	light yellow	snow4	azure4	LightBlue1	PaleGreen3	goldenrod3	brown2	pink1	DarkOrchid3	gray61	gray99	
dodger blue	yellow	seashell2	SlateBlue1	LightBlue2	PaleGreen4	goldenrod4	brown3	pink2	DarkOrchid4	gray62	gray63	



Video 43:

En este video se nos explica el uso de Frames, el uso del método **pack()** y **config()** para la configuración de diferentes parámetros de aspecto, para esto trabajamos sobre la raíz creada previamente:

```
1  from tkinter import *
2
3  raiz = Tk()
4  raiz.title("Ventana de pruebas")
5  raiz.resizable(1,1)
6  raiz.iconbitmap("logo.ico")
7  raiz.geometry("650x450")
8  raiz.config(bd = 15)
9  raiz.config(relief = "groove")
10 raiz.config(cursor = "hand2")
11 raiz.config(bg="SkyBlue2")
12
13 miFrame = Frame()
14 ##miFrame.pack(side = "right", anchor = "n")
15 #miFrame.pack(fill = "both", expand = "True")
16 miFrame.pack()
17 miFrame.config(bg="Pink")
18 miFrame.config(width = "550", height = "400")
19 miFrame.config(bd = 15)
20 ##Revisar tipos de bordes
21 miFrame.config(relief = "raised")
22 miFrame.config(cursor = "pirate")
23 raiz.mainloop()
```

Nos vamos a concentrar en las líneas 13 -23 en donde hacemos manipulación del Frame:

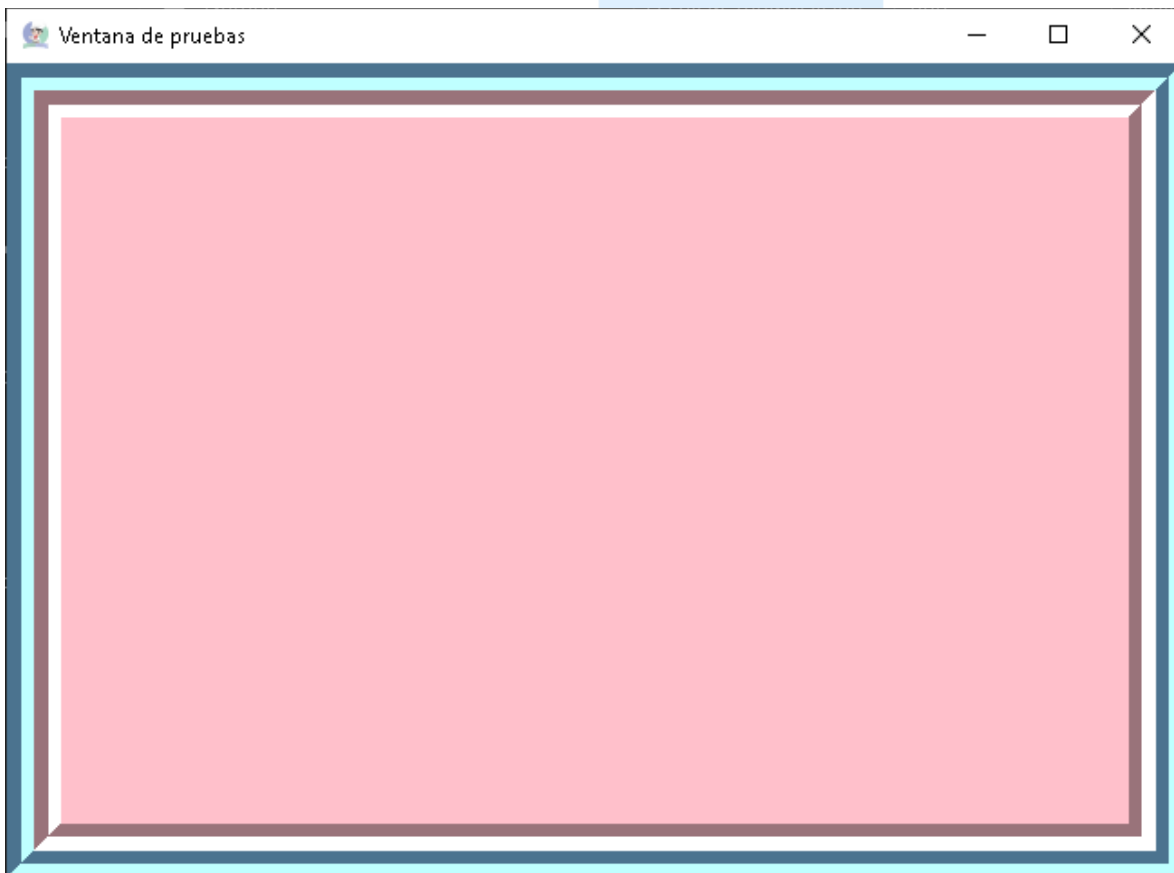
- Para la creación de un Frame creamos un objeto **Frame()** como lo vemos en la línea 13.
- Entre la línea 14 a 16 se hace uso del método **pack ()** el cual se encarga de introducir el Frame dentro de la raíz:
 - o En la línea 16 hacemos un empaquetado básico, el cual posicionara el Frame de forma predeterminada en la parte superior de la ventana.
 - o En la línea 14 se hace uso de las instrucciones **side** y **anchor**, la primera de estas servirá para posicionar en cualquier lateral de la ventana (**right, left, top o bottom**) y la segunda servirá para ubicar el frame en cualquiera de los puntos cardinales (**n, s, w, o, nw, no, sw**, so que corresponden a norte, sur, este, oeste, noreste, noroeste, sureste, suroeste)
 - o En la línea 15 encontramos el uso de las instrucciones **fill** y **expand**, el primero encargado rellenar el espacio de la raíz con el Frame dependiendo



del eje en que se le indique (x = horizontal, y = vertical o both = en horizontal y vertical)

- En la línea 17 vemos el uso del parámetro **bg**, para indicar el color del Frame.
- En la línea 18 se indica el ancho y alto del Frame por medio de los parámetros **width** y **height**
- En la línea 19 se hace uso del parámetro **bd**, el cual sirve para indicar el grosor del borde del Frame.
- En la línea 20 se hace uso de parámetro **relief**, el cual le da estilo a l borde del Frame (flat, groove, raised, ridge, solid, o sunken)
- En la línea 21 se hace uso del parámetro **cursor** el cual cambiara el aspecto del cursor al estar en su zona.

Las configuraciones especificadas para el borde, el grosor de este y el cursor pueden ser aplicados también a la raíz, como lo observamos en el resultado final.



Para mejor documentación ir a <https://docs.python.org/3/library/tk.html> (Foundation, 2020)

**Video 44:**

En este video se explica el uso de los Labels, los cuales son espacios de texto estático, el cual puede reaccionar frente acciones de los componentes o demás widgets. Los Labels son objetos los cuales podemos manipular con diferentes parámetros en el método **.config()** como los siguiente, para mas parámetros buscar en la documentación que se encuentra en la parte final del documento:



The screenshot shows a presentation slide titled "Label: Sintaxis". At the top right is a small image of a computer mouse. Below the title is a code snippet: `variableLabel = Label(contenedor, opciones)`. Below this is a table with two columns: "Opción" and "Descripción".

Opción	Descripción
Text	Texto que se muestra en el Label
Anchor	Controla la posición del texto si hay espacio suficiente para él (center por defecto)
Bg	Color de fondo
Bitmap	Mapa de bits que se mostrará como gráfico
Bd	Grosor del borde (2 px por defecto)
Font	Tipo de fuente a mostrar
Fg	Color de la fuente
Width	Ancho de Label en caracteres (no en píxeles)
Height	Altura de Label en caracteres (no en píxeles)
Image	Muestra imagen en el Label en lugar de texto
justify	Justificación del texto del Label

Para practicar el uso de **Labels**, realizamos el siguiente ejemplo:

```
1 from tkinter import *
2
3 root = Tk()
4 miFrame = Frame(root, width = 400, height = 400)
5 miFrame.pack()
6 Label(miFrame, text = "Este es un label", fg = "Red", font = ("Arial Black" ,18)).place(x= 10, y = 10)
7 imagen= PhotoImage(file="mundo.gif", format="gif -index 2")
8 Label(miFrame, image=imagen).place(x = 10, y = 40)
9 root.mainloop()
```

En la línea 6 hacemos instancia de un objeto de tipo Label el cual tiene los siguientes parámetros:

- miFrame: componente (Frame o Raíz) en el que será incluido.
- Text : Cadena de texto que contendrá el Label
- Fg: color del texto.
- Font: aquí se incluirá la descripción de la fuente a usar, para esto, busque la lista de fuentes que pueden usarse en Tkinter

En esta misma línea de damos una posición al Label dentro del Frame con el método. **place(x, y).**



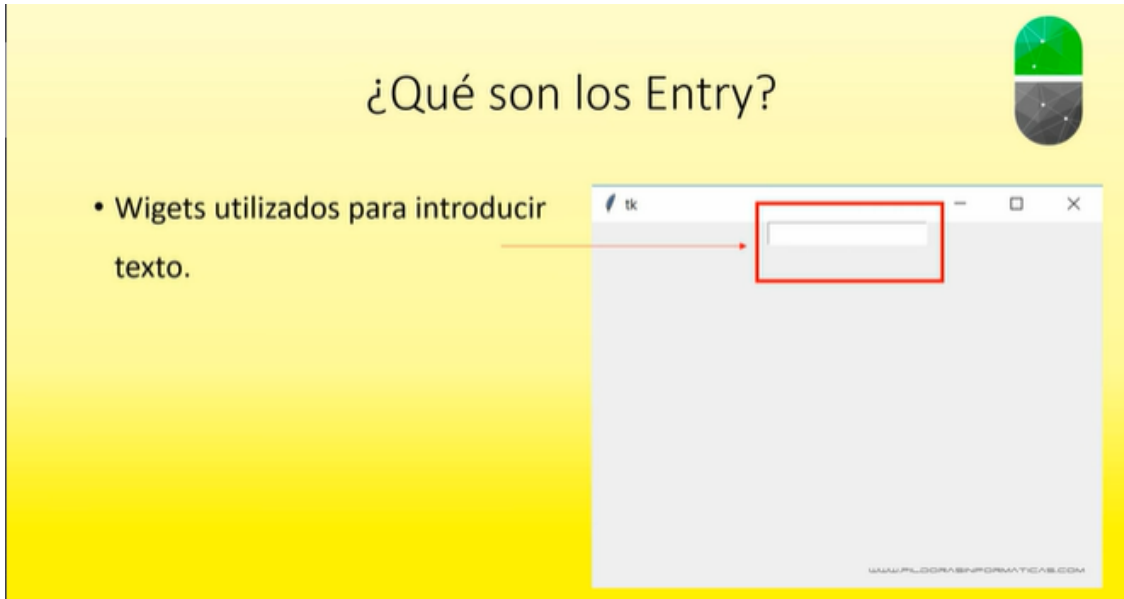
Luego hacemos uso de un objeto tipo **PhotoImage**, el cual trabaja con imágenes en formato PNG o GIF, el cual almacena una imagen para ser usada. Esta imagen luego la usamos en otro Label, en donde su segundo parámetro será la imagen que queremos incluir.

A continuación el resultado del ejercicio:



**Video 45:**

En este video se nos explica el uso de los campos de entrada de texto o Entry, los cuales son aquellas casillas en las cuales podemos escribir cadenas de texto para su posterior manipulación o visualización. Este tipo de objetos usa parámetros similares a los existentes en los Labels.



Como parte importante, se nos enseña la implementación del método **Grid(irow = ,columna =jj)** el cual genera una malla o matriz sobre el componente en que posicionen los elementos, en donde podremos usar como coordenadas de posicionamiento la columna y la fila en donde se encontrara el objeto:

Se nos explica que la cuadrilla en la que trabajaremos sobre el Frame tendrá **n** y **m** filias y columnas, y mediante las coordenadas de esta podremos posicionar los componentes que requiramos, además de poder configurar el comportamiento de los componentes dentro de cada casilla.

En el ejemplo a continuación se nos ejemplifica el uso de Labels, Entrys y del método Grid para el posicionamiento de estos componentes en el Frame.

	0	1	2	3	4
0					
1					
2					
3					
4					



```
1  from tkinter import *
2
3  raiz = Tk()
4  frame1 = Frame(raiz,width = 400, height = 400)
5  frame1.pack()
6  cuadro1 = Entry(frame1)
7  cuadro1.grid(row = 0,column =1,padx=10, pady = 10)
8  cuadro1.config(fg = "red", justify = "center")
9  label1 = Label(frame1, text = "Nombre:")
10 label1.grid(row =0,column =0,sticky = "w",padx=10, pady = 10)
11
12
13 labelp = Label (frame1,text = "Contraseña:")
14 labelp.grid(row =1,column =0,sticky = "w",padx=10, pady = 10)
15 cuadropassword = Entry(frame1)
16 cuadropassword.grid(row =1,column =1,sticky = "w",padx=10, pady = 10)
17 cuadropassword.config(show = "*")
18
19 cuadro2 = Entry(frame1)
20 cuadro2.grid(row = 2,column =1,padx=10, pady = 10)
21 label2 = Label(frame1, text = "Apellido:")
22 label2.grid(row =2,column =0,sticky = "w",padx=10, pady = 10)
23
24 cuadro3 = Entry(frame1)
25 cuadro3.grid(row = 3,column =1,padx=10, pady = 10)
26 label3 = Label(frame1, text = "Dirección de residencia:")
27 label3.grid(row =3,column =0, sticky = "w",padx=10, pady = 10)
28
29 raiz.mainloop()
```

Como podemos ver, alas instancias de **Entry** tienen los siguientes parámetros de configuración:

- Frame1: componente en el que se va a posicionar el campo.

Al momento de hacer uso del método Grid se hace uso de los siguientes parámetros:

- Row: Fila en la que se va a ubicar el componente
- Column: Columna en la que se va a posicionar el componente
- Sticky: ubicación del componente en el espacio/cuadro por medio de puntos cardiales (n, s, w, o, nw, no, sw, so que corresponden a norte, sur, este, oeste, noreste, noroeste, sureste, suroeste)
- Padx: espacio en pixeles de los bordes laterales del componente con el siguiente y anterior

tk

Nombre:

Contraseña:

Apellido:

Dirección:

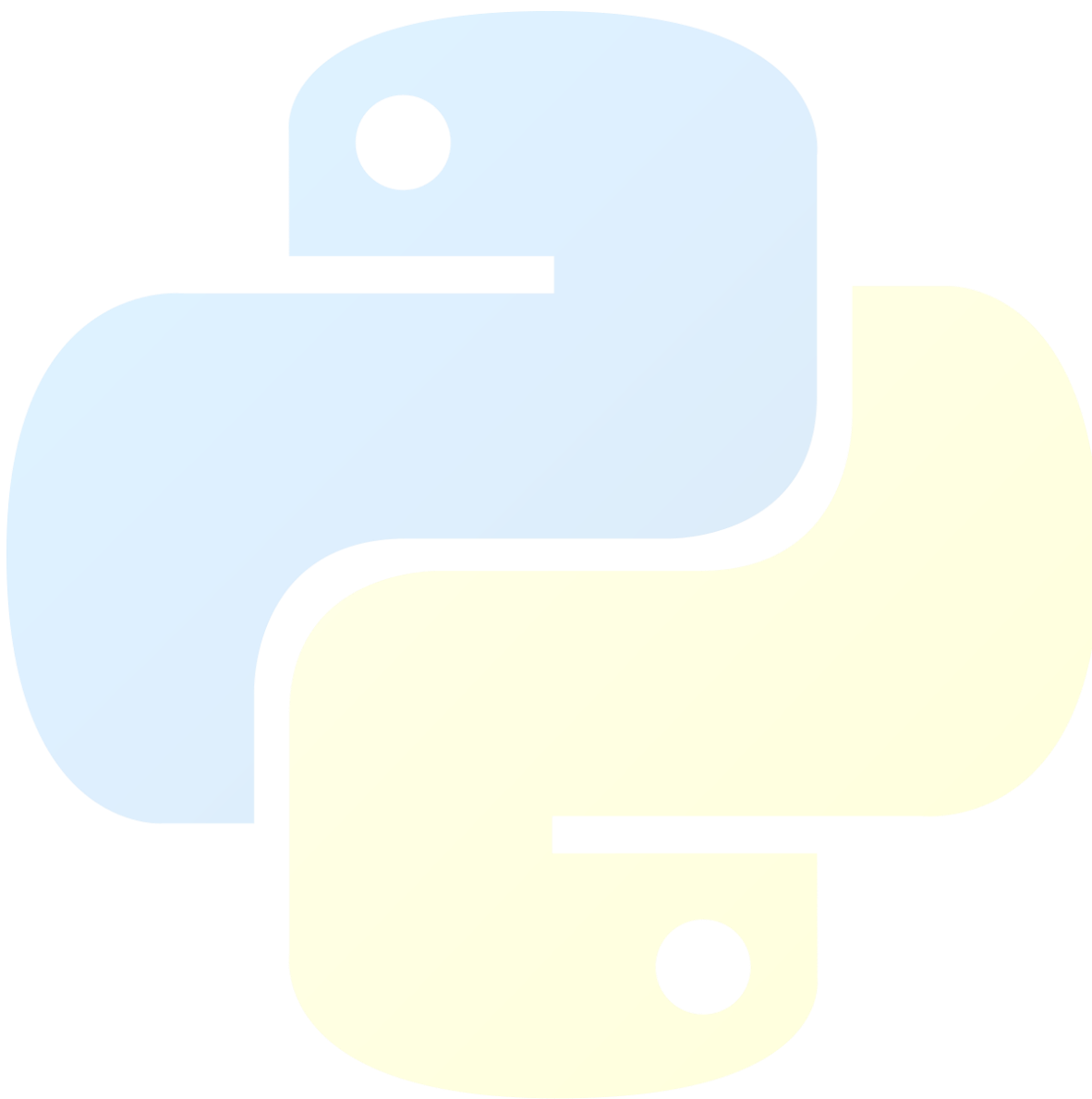
Comentario:

Enviar



- Pady: espacio en pixeles de los bordes frontales del componente con el superior e inferior.

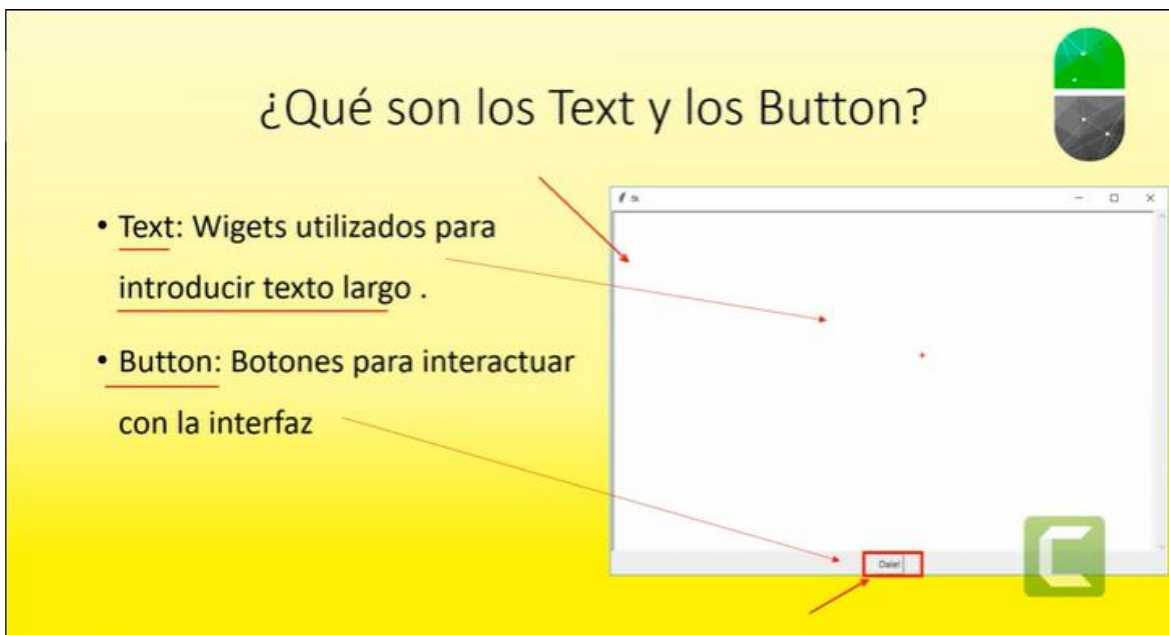
En la línea 17 se hace uso de método **config()** con el parámetro show el cual servirá para ocultar la información ingresada al campo.





Video 46:

En este video se trabaja con dos componentes gráficos importantes en las interfaces gráficas, como lo son los campos de Texto extendido y los Botones, los cuales nos permiten interactuar con el usuario de forma más cercana.



Para dar un ejemplo del uso de estos componentes trabajamos sobre el ejemplo del video pasado:

```
35 label4 = Label(frame1, text = "Comentario:")
36 label4.grid(row = 4, column = 0, sticky = "w", padx = 10, pady = 10)
37 textoComent = Text(frame1, width = 16, height = 5)
38 textoComent.grid(row = 4, column = 1, padx = 10, pady = 10)
39 #Barra desplazamiento
40 scrollbar = Scrollbar(frame1, command = textoComent.yview)
41 scrollbar.grid(row = 4, column = 2, sticky = "nsew")
42 textoComent.config(yscrollcommand = scrollbar.set)
43
44 def codigoBoton ():
45     minombre.set("Felipe")
46
47 boton1 = Button(raiz, text = "Enviar", command = codigoBoton)
48 boton1.pack()
49
50 raiz.mainloop()
51
```

Como vemos en la línea 37 creamos un objeto de tipo Text, en donde lo ubicamos dentro del Frame y le damos las dimensiones del tamaño con **Width** y **Height**, además de darle una posición en el Frame con el método Grid.



En la línea 40 creamos un objeto que no se había mencionado antes pero que es importante porque permite interactuar mejor con la información de un Text, como lo es un **Scroll Bar**, lo cual es una barra que permite desplazar el contenido del **Text** para ser visualizado. Dentro de sus parámetros de creación vemos el parámetro **command** el cual le da la habilidad de generar una vista vertical del campo Text. Posicionamos este objeto Scroll Bar al lado del campo Text con el método Grid, en donde su tercer parámetro **Sticky**, le da la propiedad de que el tamaño del Scroll Bar se apegue al del campo Text.

Finalmente, le damos al campo Text la habilidad de sincronizar el desplazamiento del texto dentro de el con el movimiento de la Barra lateral con el método **Config ()** en donde tiene como parámetro **yscrollcommand** que define como barra de desplazamiento la previamente creada.

Por otro lado, en la línea 47 creamos un botón, el cual como parte especial trabaja con el parámetro **command**, el cual recibe como método de actividad el método **codigoBoton**, el cual le imprimirá al cuadro de texto 1 (en la imagen inferior), el valor del Objeto – Cadena de caracteres **miNombre**.

```
7  minombre = StringVar()

9  cuadro1 = Entry(frame1, textvariable=minombre)
10 cuadro1.grid(row = 0, column = 1, padx=10, pady = 10)
11 cuadro1.config(fg = "red", justify = "center")
12
```

A continuación, se muestra el resultado final del ejercicio:

tk

Nombre: Felipe

Contraseña:

Apellido:

Dirección:

Comentario:

Enviar



Video 47 – 48 – 49

En estos videos trabajamos un ejercicio práctico de la creación de una calculadora, en donde pusimos en practica los conocimientos aprendidos del apartado de interfaces graficas:

```
from tkinter import *

raiz = Tk()
raiz.config(bg = "black")
frame = Frame(raiz)
frame.config(bg = "gray40")
frame.pack()

#-----PANTALLA
numeroPantalla = StringVar()
operacion = ""
resultado = 0

pantalla = Entry(frame, textvariable = numeroPantalla)
pantalla.grid(row = 1, column = 1, padx = 10, pady = 10, columnspan = 4)
pantalla.config(bg = "Black", fg = "Green1", justify = "right")

#----- METODOS
def numerPulsado (num):
    global operacion
    m = numeroPantalla.get()
    coma = m.count('.')
    if (operacion != ""):
        numeroPantalla.set(num)
        operacion= ""
    else :
        numeroPantalla.set(numeroPantalla.get()+num)
```



```
def borrarContenido():
    numeroPantalla.set(numeroPantalla.get()[:-1])

def suma(num):
    global operacion
    global resultado
    m = numeroPantalla.get()
    if (len(m) != 0):
        resultado+= int (num)
        operacion = "suma"
        numeroPantalla.set(resultado)

def total ():
    global resultado
    numeroPantalla.set(resultado+float(numeroPantalla.get()))
    resultado = 0

#----- FILA 1
boton7 = Button (frame, text = "7", width = 3,
command=lambda: numerPulsado("7"))
boton8 = Button (frame, text = "8", width = 3,
command=lambda: numerPulsado("8"))
boton9 = Button (frame, text = "9", width = 3,
command=lambda: numerPulsado("9"))
botondiv = Button (frame, text = "/", width = 3)
boton7.grid(row = 2, column = 1)
boton8.grid(row = 2, column = 2)
```



```
boton9.grid(row = 2, column = 3)
botondiv.grid(row = 2, column = 4)
#----- FILA 2
boton4 = Button (frame, text = "4", width = 3,
command=lambda:numerPulsado("4"))
boton5 = Button (frame, text = "5", width = 3,
command=lambda:numerPulsado("5"))
boton6 = Button (frame, text = "6", width = 3,
command=lambda:numerPulsado("6"))
botonx = Button (frame, text = "x", width = 3)
boton4.grid(row = 3, column = 1)
boton5.grid(row = 3, column = 2)
boton6.grid(row = 3, column = 3)
botonx.grid(row = 3, column = 4)
#----- FILA 3
boton1 = Button (frame, text = "1", width = 3,
command=lambda:numerPulsado("1"))
boton2 = Button (frame, text = "2", width = 3,
command=lambda:numerPulsado("2"))
boton3 = Button (frame, text = "3", width = 3,
command=lambda:numerPulsado("3"))
botonres = Button (frame, text = "-", width = 3)
boton1.grid(row = 4, column = 1)
boton2.grid(row = 4, column = 2)
boton3.grid(row = 4, column = 3)
botonres.grid(row = 4, column = 4)
#----- FILA 4
boton0 = Button (frame, text = "0", width = 3,
command=lambda:numerPulsado("0"))
botoncoma = Button (frame, text = ",", width = 3,
command=lambda:numerPulsado(","))
```



```
botonigual = Button (frame, text = "=", width = 3,command
=lambda:total())

botonsuma = Button (frame, text = "+", width = 3,command =
lambda:suma(numeroPantalla.get()))

boton0.grid(row = 5, column = 1)

botoncoma.grid(row = 5, column = 2)

botonigual.grid(row = 5, column = 3)

botonsuma.grid(row = 5, column = 4)


botonBorrar = Button(frame,text = "DEL",width = 4,
command=borrarContenido)

botonBorrar.grid(row = 6 , column= 3, padx = 10, pady = 10, columnspan =
2)

raiz.mainloop()
```

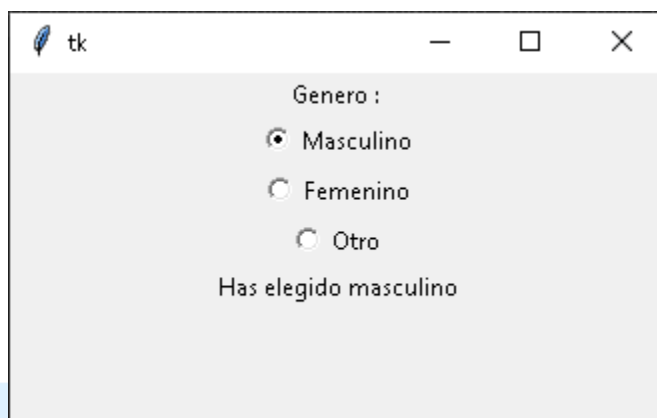
A lo largo del texto se trabajan con conceptos relacionados con la obtención de valores al actuar sobre los botones de la calculadora, como lo son la muestra de los datos obtenidos, la operación de los valores numéricos, y la interacción de los valores.:

- **Lambda:método():** el uso de funciones lambda facilita la interacción del usuarios con los componentes, siendo un método que reaccione solo cual se presione los botones.
- **Columnspan:** este parámetro lo vemos al momento de posicionar el objeto Entry de la pantalla de la calculadora, por medio de este se indica la cantidad de columnas que ocupara el objeto



Video 50:

En este ultimo video del primer modulo del curso de Python, vamos a trabajar con otro componente grafico como lo son los **RadioButtons**, los cuales se usan en menús de selección múltiple.



Para esto, trabajamos con el siguiente ejemplo:

```
1 from tkinter import *
2 r = Tk()
3 varOption = IntVar()
4 def imprimir():
5     if(varOption.get() == 1):
6         etiqueta.config(text = "Has elegido masculino")
7     elif (varOption.get() == 2):
8         etiqueta.config(text = "Has elegido Femenino")
9     else:
10        etiqueta.config(text = "Has elegido femenino")
11 Label(r, text="Genero : ").pack()
12 Radiobutton(r, text="Masculino",variable = varOption, value = 1,command = imprimir).pack()
13 Radiobutton(r, text="Femenino",variable = varOption, value = 2,command = imprimir).pack()
14 Radiobutton(r, text="Otro",variable = varOption, value = 3,command = imprimir).pack()
15 etiqueta =Label(r)
16 etiqueta.pack()
17 r.mainloop()
```

Como podemos observar, los componentes se posicionan sobre la Raíz.

Primero que todo, creamos un Objeto para almacenar valores enteros, en donde se almacenara el valor del Radiobutton seleccionado. Con este valor trabaja el método imprimir el cual le dará un texto a un Label. Finalmente vemos el uso de los RadioButton los cuales se les dará parámetros de texto, variable la cual reaccionará con este, valor numérico y el método que se ejecutará al presionarlo.



Bibliografía

Alvarez, A. (2015). *Guia Tkinter*. Recuperado el 14 de 05 de 2020, de <https://guia-tkinter.readthedocs.io/es/develop/>

Foundation, P. S. (14 de Mayo de 2020). *Graphical User Interfaces with Tk*. Recuperado el 14 de Mayo de 2020, de <https://docs.python.org/3/library/tk.html>

Moe, D. &. (2020). *Codigo de color de HTML*. Recuperado el 14 de 05 de 2020, de Selector de color: <https://htmlcolorcodes.com/es/selector-de-color/>

van Rossum, G. (16 de 10 de 2000). *Referencia de la Biblioteca de Python*. (J. e. Fred L. Drake, Ed.) Recuperado el 10 de 05 de 2020, de <http://pyspanishdoc.sourceforge.net/lib/lib.html>