

RELATORIA1

Presentado por: Jose David Vazco Loaiza

fecha: 24 de agosto de 2022.

clase n.1

hora: 2 a 4 pm.

Título: Entorno python.

Objetivo general: Conocer las diferentes aplicaciones que usaremos para el desarrollo del curso de informática III.

Objetivos específicos:

- Aprender a instalar las aplicaciones a utilizar tales como Visual C, Python y GitHub.
- Crear cuenta o login para la herramienta que usaremos para subir a la nube todos los archivos que tengan que ver con el curso de informática III.

Resumen:

En la siguiente práctica de programación informática aprenderemos a conocer el entorno de trabajo en el que se desarrollará el curso hasta su final, es decir conoceremos las herramientas que utilizaremos de aquí en adelante, por esto aprendimos a instalar Visual Studio Code directamente de su instalador descargado desde google, así también lo hicimos con la herramienta de programación Python el cual nos servirá para abordar diferentes aplicaciones en la vida real es por esto que nos adentraremos en su lenguaje y por último realizamos la descarga e instalación de GitHub una herramienta que nos permite subir archivos del curso a la nube incluyendo los códigos de Python ya que estas se relacionan a su vez con el VSC, por esto creamos una cuenta o login para interconectar las plataformas en cada herramienta y poder tener un uso óptimo de estas aplicaciones.

Marco teórico.

GitHub: Esta aplicación permite compartir proyectos de software con toda la comunidad de ingenieros y programadores de código abierto, actualmente es una de las herramientas más usadas, ya con más de 12 años desde su creación es la aplicación preferida para administrar repositorios de información. Principalmente para su uso se requiere de una cuenta, luego de proceder a crearla en su misma página podemos iniciar nuestro primer repositorio, es decir buscar el recuadro de iniciar repositorio (Initialize Repository) que llevará el nombre de CLASES y allí será la nube al cual subiremos toda la info vista en el curso. Lista nuestra primera parte de la clase. Ahora procedemos a hablar un poco de VSC.

Visual Studio Code: Esta aplicación es desarrollada por Microsoft para editores de código fuente para diferentes sistemas operativos como Windows o Linux. Viene con el control integrado de Git, personalización de pantalla y teclado, programación de código y refactorización de código entre otras aplicaciones. Así pues como lo mencionamos anteriormente logueamos la cuenta creada en git para configurarla y sincronizarla en VSC. De la sgte manera se escribe directamente en la Terminal VSC:

```
git config --global user.name "pepito"  
git config --global user.email "pepito@unal.edu.co"
```

También podemos ver la instalación correcta de Git en VSC escribiendo en la terminal:

```
git --version
```

Una vez verificada la sincronización de datos sabemos que debemos iniciar un repositorio como se mencionó al principio lleva de nombre CLASES, ahora ahí mismo en VSC creamos un nuevo archivo de trabajo y en el panel superior izq. de VSC tenemos que seleccionar la pestaña de Git, luego clicar el botón de “initialize repository” llegados a este punto en nuestro working directory, procedemos a ir al stage área el cual es seleccionar la opción “+”, así podremos pasar a escribir un mensaje que llevará nuestro archivo y luego damos al botón commit área y finalmente subimos al repositorio de la nube pillando el boton donde esta la nube.

Fecha: 26/08/2022

clase N.2

Título: Tipos de datos.

Objetivo general: Describir y conocer los diferentes tipos de datos de Python.

Objetivo específico: Conocer las características y propiedades de los tipos de datos de python.

Resumen: En esta clase vamos a conocer los tipos de datos que maneja Python desde los datos booleanos hasta alfanuméricos. Para ello repasamos la definición de los tipos de datos como aquel conjunto de datos que tienen características o propiedades establecidas o determinadas. Luego vimos que los tipos de datos como los de tipo numéricos trabajan con los números enteros o números de punto flotante lo cual en Py se usa la función “Float()”. Luego existe otro tipo de datos que se trabaja con booleanos y vimos que en Py se representa como “bool” y contiene dos valores (En base 2), es decir True o False. También existen otros tipos de datos como las listas, las tuplas o los diccionarios cuando son datos almacenados en un conjunto o llaves tales como “[1,2,3]”.

Marco teórico:

Los tipos de datos en Py: Los tipos de datos se consideran como un conjunto de datos que poseen una característica o propiedad determinada. por ejemplo: [1,2,3] a esto en python le llamamos tipos de datos.

Tipos de datos básicos en Py: Estos datos pueden ser específicamente numéricos, punto flotante (Números reales), complejos y las cadenas de caracteres. También hay otros tipos de datos como lo son las listas, las tuplas, mapas, conjuntos, iteradores, clases, instancias y excepciones.

- **Tipo numérico:** Este tipo de dato pueden ser enteros que lo vamos a representar en Py como "int". Este tipo de dato contiene a los números enteros (no por completo). También podemos expresar estos números en otros sistemas como octal, binario y hexadecimal.

Los números octales se crean anteponiendo "0o" a una secuencia de números octales que van del 0 al 7. Lo mismo ocurre con los hexadecimales, se antepone "0x" a una secuencia de hexadecimales, por ejemplo: "0xA" teniendo en cuenta que van del 0 a 9 y de la A a la F. Para los binarios se antepone "0b" a una secuencia de binarios.

```
>>> diez = 10
>>> diez_binario = 0b1010
>>> diez_octal = 0o12
>>> diez_hex = 0xa
>>> print(diez)
10
>>> print(diez_binario)
10
>>> print(diez_octal)
10
>>> print(diez_hex)
10
```

Otro dato numérico es el punto flotante que en Py lo representamos como "Float", como este tipo de dato puede ser un número con infinitos decimales, en el programa no se va a poder representar por completo, así pues se inventó la notación científica de números de punto flotante (así como funciona con los números reales) para máquinas de 32 bits o 64b. El dato tipo "Float" se puede crear a través de una variable o un resultado de una expresión o resultado de una función. Por ejemplo:

```
>>> un_real = 1.1 # El literal debe incluir el carácter .
```

```
>>> otro_real = 1/2 # El resultado de 1/2 es un float
>>> n = 1.23E3 # float con notación científica (1230.0)
```

El tipo de dato complejo en Py se representa como “complex”. Los números complejos tienen una parte real e imaginaria y se representan como un Float. Por ejemplo:

```
>>> complejo = 1+2j
>>> complejo.real
1.0
>>> complejo.imag
2.0
```

- **Tipo Booleano:** Este tipo de datos se representan en Py como True o False, cualquier objeto que tenga esta categoría puede ser usado en una situación donde se requiera comprobar si algo es verdadero o falso. Por ejemplo se puede usar en estructuras como un If o un While.

Todo objeto es considerado como verdadero excepto:

- Si se usa la expresión “__bool__()” y el objeto es falso.
 - Si usa la expresión “__len__()” es el resultado es 0.
- **Tipo cadena de caracteres:** Este tipo de datos en Py nos referimos a las letras y es conocido como “str”. este tipo de secuencias es inmutable. Para crearlo en Py solo tienes que usar las comillas “” y poner una secuencia string. Por ejemplo:

```
>>> hola = 'Hola "Pythonista"'
>>> hola_2 = 'Hola \'Pythonista\''
>>> hola_3 = "Hola 'Pythonista'"
```

```
>>> print(hola)
Hola "Pythonista"
```

```
>>> print(hola_2)
Hola 'Pythonista'
```

```
>>> print(hola_3)
Hola 'Pythonista'
```

- **Otros tipos:** Existen otros tipos de datos como lo son:

- Listas: Secuencias mutables de valores.
- Tuplas: Secuencias inmutables de valores.
- Los conjuntos: Se usan para representar conjuntos de elementos únicos, es decir, no hay elementos repetidos dentro del conjunto.
- Los diccionarios: Son tipos especiales de contenedores en los que se puede acceder a sus elementos a partir de una clave única.

Por ejemplo tenemos:

```
>>> lista = [1, 2, 3, 8, 9]
>>> tupla = (1, 4, 8, 0, 5)
>>> conjunto = set([1, 3, 1, 4])
>>> diccionario = {'a': 1, 'b': 3, 'z': 8}
>>> print(lista)
[1, 2, 3, 8, 9]
>>> print(tupla)
(1, 4, 8, 0, 5)
>>> print(conjunto)
{1, 3, 4}
>>> print(diccionario)
{'a': 1, 'b': 3, 'z': 8}
```

Comparación con otros lenguajes:

En java, C y C++ podemos observar que manejan una tipología de datos similar entre sí, respecto a python también se puede observar que la estructura de datos es similar, me refiero a sus agrupaciones según sean esas características que hemos determinado para ese conjunto de datos, la única diferencia es que en los programas mencionado anteriormente denominamos tipos de datos simples o primitivos que pueden ser numéricos, booleanos o caracteres, otra diferencia más respecto a Py es que estos programas pueden tener tipos de datos primitivos o clases mucho más especiales que permiten modificar tipos de datos primitivos.

Ejemplos de repaso vistos en clase:

TIPOS DE DATOS EN python

* BOOLEANOS: True, False

* STRINGS: "", ' ', "casa", "123", "","", "12.23132"

* ENTEROS: -1000, 1000, 0,

* FLOTANTES: 12.5, -100.0, 5.

* LISTAS: [], [1,2,3], ["A", "B", "C"], [1, "A", 2, "B", 3, "C"] ==>

MUTABLES

* TUPLAS: (), (1,2,3), ("A", "B", "C"), (1, "A", 2, "B", 3, "C") ==>

INMUTABLES

* CONJUNTOS: {"A", "B", "C"}

* DICCIONARIOS: {"CRISTIAN": 5, "DANIELA": 0, "SEBASTIAN": 3}

Referencias bibliográficas:

- <https://j2logo.com/python/tutorial/>
- https://cruzado.info/tutojava/l_4.htm

Clase N.3

Fecha: 26/08/2022 y 31/08/2022.

Título: Operadores en Py.

Objetivo general: Describir y conocer los diferentes tipos de operadores de Python.

Objetivo específico: Conocer las características y propiedades de los tipos de operadores de python.

Resumen: En esta práctica vamos a ver los diferentes tipos de operadores que usaremos en Py que van desde los tipos de operadores booleanos hasta tipo los tipos String, para estos operadores hay unos símbolos que son propios del lenguaje en cuestión el cual se usan para operar uno, dos o más elementos, también analizaremos la estructura jerárquica que poseen estos operadores según Py y su estructura lógica para que funcionen correctamente en la redacción de los códigos de programación.

Marco teórico:

- **Tipo de operaciones booleanas:** Son denominadas de la siguiente manera, +, And, Or y Not.

Hay que recalcar que las operaciones no devuelven True o False, devuelve otro operador.

A or B, si A se evalúa como falso, entonces devuelve B, sino devuelve A. Es decir, evalúa el segundo término si A es falso.

A and B, si A se evalúa a falso, entonces devuelve A sino devuelve B. Es decir, solo evalúa el segundo operando si el primero o sea A es verdadero.

Por ejemplo:

```
x = True
y = False
x or y
True
```

```
x and y
False
```

```
not x
False
```

```
x = 0
y = 10
x or y
10
```

```
x and y
0
```

```
not x
True
```

- **Operadores aritméticos:** Estos operadores me permiten realizar operaciones tales como la suma, resta, multiplicación, división etc.

Por ejemplo tenemos esta tabla:

Operador	Descripción
+	Suma dos operandos.
-	Resta al operando de la izquierda el valor del operando de la derecha. Utilizado sobre un único operando, le cambia el signo.
*	Producto/Multiplicación de dos operandos.
/	Divide el operando de la izquierda por el de la derecha (el resultado siempre es un <code>float</code>).
%	Operador módulo. Obtiene el resto de dividir el operando de la izquierda por el de la derecha.
//	Obtiene el cociente entero de dividir el operando de la izquierda por el de la derecha.
**	Potencia. El resultado es el operando de la izquierda elevado a la potencia del operando de la derecha.

Por ejemplo:

```
x = 7
y = 2
x + y # Suma
9
x - y # Resta
5
x * y # Producto
14
x / y # División
3.5
x % y # Resto
1
x // y # Cociente
3
x ** y # Potencia
49
```


- **Operadores de comparación:** Estos operadores como su nombre lo indica se usan para comparar 2 o más valores. Su respuesta es un True o False.

En la siguiente tabla vemos condensados estos operadores:

Operador	Descripción
>	Mayor que. <code>True</code> si el operando de la izquierda es estrictamente mayor que el de la derecha; <code>False</code> en caso contrario.
>=	Mayor o igual que. <code>True</code> si el operando de la izquierda es mayor o igual que el de la derecha; <code>False</code> en caso contrario.
<	Menor que. <code>True</code> si el operando de la izquierda es estrictamente menor que el de la derecha; <code>False</code> en caso contrario.
<=	Menor o igual que. <code>True</code> si el operando de la izquierda es menor o igual que el de la derecha; <code>False</code> en caso contrario.
==	Igual. <code>True</code> si el operando de la izquierda es igual que el de la derecha; <code>False</code> en caso contrario.
!=	Distinto. <code>True</code> si los operandos son distintos; <code>False</code> en caso contrario.

Por ejemplo tenemos:

```
>>> x = 9
>>> y = 1
>>> x < y
False
>>> x > y
True
>>> x == y
False
```

- **Operadores de concatenación o tipo string:** Una de las operaciones básicas en Py cuando se trabaja con cadenas de caracteres lo llamamos concatenación. La forma mas simple de concatenar dos cadenas es usando el operador "+". Por ejemplo:

```
hola = 'Hola'
python = 'Pythonista'
hola_python = hola + ' ' + python # concatenamos 3 strings
print(hola_python)
Hola Pythonista
```

Ejemplos de jerarquización:

En este ejemplo vemos que sin usar paréntesis para separar las operaciones la potenciación tiene prioridad sobre la división:

<pre>x=2 y=9 x/y**x</pre>	<pre>x=2 y=9 x/(y**x)</pre>
0.024691358024691357	0.024691358024691357

En este ejemplo vemos cómo Py primero opera la suma y luego eleva a la potencia porque tiene paréntesis pero de lo contrario sino separamos por paréntesis la prioridad la tiene la potenciación:

<pre>x=2 y=9 (x+y)**y</pre>	<pre>x=2 y=9 x+y**y</pre>
2357947691	387420491

Otro ejemplo claro en el que vemos como Py primero realiza la resta dentro del paréntesis y luego la división, de lo contrario la división tiene prioridad sobre la resta:

<pre>x=2 y=9 (x-y)/y</pre>	<pre>x=2 y=9 x-y/y</pre>
-0.7777777777777778	1.0

Un ejemplo más claro en el que primero Py hace la división de los números dentro del paréntesis y luego realiza la potenciación:

<pre>x=2 y=9 (x/y)**x</pre>
0.04938271604938271

Existen también otros tipos de operadores tales como:

Operadores de pertenencia: “in”, “not in” se usan para comparar tipos de datos como listas o tuplas.

Operador	Descripción
in	Devuelve True si el valor se encuentra en una secuencia; False en caso contrario.
not in	Devuelve True si el valor no se encuentra en una secuencia; False en caso contrario.

Operadores de conjuntos: Estos operadores son “|”, “&”, “-”

Referencias bibliograficas:

- <https://j2logo.com/python/tutorial/operadores-en-python/#operadores-logicos>

Clase N.4

Fecha: 02/09/2022

Título: Funciones integradas en Py.

Objetivo general: Conocer las diferentes funciones integradas de Python.

Objetivo específico: Conocer las características y propiedades de los tipos de funciones integradas de python.

Resumen: En esta clase vimos las diferentes funciones integradas de Python que van desde funciones de entrada y salida hasta funciones de operaciones con secuencia. Describimos cada una e hicimos unos breves ejemplos de su funcionamiento, hasta el momento podemos ver como funciona cada una de ellas y a partir de el aprendizaje de la sintaxis de Py podemos darle un mejor uso para optimizar y sentirnos más cómodos con el lenguaje de programación.

Marco teórico:

Veremos los diferentes tipos de funciones integradas que se manejan en Py:

- **Funciones de entrada y salida:** Este tipo de funciones las usamos para introducir y reproducir datos, como muy bien se menciona estas funciones manejan un dominio numérico y de tipo string, estas funciones son: input(), print() y format().
print(): Imprimir por la salida estándar la representación en string de cualquier objeto.

```
▶ print("Hola mundo, estoy listo para programar")  
Hola mundo, estoy listo para programar
```

input(): Me permite crear una box para ingresar datos por parte del usuario, podemos ingresar datos y normalmente esta caja lleva un mensaje especificando qué dato o respuesta debe ingresar un usuario.

```
input("introduzca el valor de la temperatura: ")  
  
introduzca el valor de la temperatura: 478K  
'478K'
```

format(): esta función me permite formatear un valor de tipo numérico (número real) o string para imprimir una parte de la cadena o número que se quiere mostrar, para ejecutar esta acción debemos poner "3.f " para indicar el número o la parte de la cadena que queremos recortar y el 3 es el número de caracteres que se muestran :

<pre>format(1.98276458736, ".3f") '1.983'</pre>	<pre>x=1.345678 format(x, ".1f") '1.3'</pre>
--	---

- **Funciones de ayuda para programar:**

- help(): muestra la ayuda integrada de cualquier componente de Py, si se deja libre la función sin ningún componente que queramos comprender entonces abre una box de ayuda interactiva.

```
>help()  
  
Welcome to Python 3.7's help utility!  
  
If this is your first time using Python, you should definitely check out  
the tutorial on the Internet at https://docs.python.org/3.7/tutorial/.  
  
Enter the name of any module, keyword, or topic to get help on writing  
Python programs and using Python modules. To quit this help utility and  
return to the interpreter, just type "quit".  
  
To get a list of available modules, keywords, symbols, or topics, type  
"modules", "keywords", "symbols", or "topics". Each module also comes  
with a one-line summary of what it does; to list the modules whose name  
or summary contain a given string such as "spam", type "modules spam".  
  
help> 
```

- `dir()`: Obtenemos una lista cuyos elementos son los métodos de objeto que se muestran ordenados alfabéticamente.

```
dir(list)
['_add_',
 '_class_',
 '_contains_',
 '_delattr_',
 '_delitem_',
 '_getitem_',
 '_len_',
 '_pop_',
 '_remove_',
 '_reverse_',
 '_setitem_',
 '_sort_',
 '_str_',
 '_subclasshook_',
 '_update_']

dir([1,2,3])
['_add_',
 '_class_',
 '_contains_',
 '_delattr_',
 '_delitem_',
 '_getitem_',
 '_len_',
 '_pop_',
 '_remove_',
 '_reverse_',
 '_setitem_',
 '_sort_',
 '_str_',
 '_subclasshook_',
 '_update_']
```

efectivamente encontramos todas las funciones que se pueden usar con "list".

- `type()`: Tipea o clasifica en tipo de dato cualquier objeto que se ingresa.

```
[29] type("Hola mundo")
str
```

```
[30] type(12.34)
float
```

```
type(12345)
int
```

- **Funciones matemáticas:** Estas funciones me permiten trabajar con números para importar valores que deben pasar por cierta operación matemática, es decir:

- `abs()`: Me permite sacar el valor absoluto de un número:

```
abs(-12.3)
12.3
```

- `round()`: Se usa para redondear un número a su entero mas proximo:

```
round(-12.3)
-12
```

- `pow()`: calcula la potenciación definiendo la base y la potencia, o sea:

```
pow(2,3)
8
```

- **Funciones de conversión:** Este tipo de funciones las usamos para convertir valores de entrada de una misma característica en otra forma, por ejemplo si son números podemos trabajar en conversión a sistemas numéricos entre sí. Estas funciones son:
- `int()` se usa para transformar un número a un entero:

```
int(0b100)
4
```

También tenemos la función `float()` que me permite convertir un número cualquiera a un número entero. igualmente tenemos las funciones `str()`, `bool()`, `bin()`, `complex()`, `set()`, esta función `set` me permite crear la clase `set` de cualquier objeto, es decir:

```
set("Hola mundo")
{' ', 'H', 'a', 'd', 'l', 'm', 'n', 'o', 'u'}
```

Otros ejemplos:

```
[63] str(0b100)
'4'
```

En el ejemplo anterior podemos observar una curiosidad como con la función de conversión `str()` cuando se ingresan números binarios los transforma en enteros de una vez.

- **Funciones de tipo secuencia:**
- `range()`: la función `range(número fin de la secuencia)` genera una secuencia de números enteros que podemos usar para iterar o contar en un bucle. De manera que podemos tomar el primer número y con un determinado paso hasta un número final pero este no será tomado en cuenta en la iteración.

```
list(range(1,9))
[1, 2, 3, 4, 5, 6, 7, 8]
```

```
list(range(1,9,2))
[1, 3, 5, 7]
```

- `enumerate()`: La función `enumerate()` toma una secuencia o iterador y lo retorna a un objeto tipo `enumerate`. Este objeto es una secuencia iterable de tuplas con su respectivo índice. Es decir:

```
lenguajes = ["Python", "Java", "JavaScript", "C++"]
list(enumerate(lenguajes))
```

[(0, 'Python'), (1, 'Java'), (2, 'JavaScript'), (3, 'C++')]

- len(): esta función retorna el número de elementos que contiene un objeto.

```
len("python")
```

6

- max(), min(), sum(): son funciones que actúan como como len() pero retornan el número máximo o más grande de una secuencia iterable, el número mínimo o más pequeño de una secuencia iterable o la suma de estos objetos de la secuencia iterable respectivamente.

Ejemplo:

<pre>sum([1,3,5])</pre> <p>9</p>	<pre>max([1,3,5])</pre> <p>5</p>	<pre>min([1,3,5])</pre> <p>1</p>
----------------------------------	----------------------------------	----------------------------------

- **Funciones aplicadas a iterables:**
- map(): Esta función me permite aplicar una función a cada uno de los objetos iterables, o sea map(función, iterable):

```
mi_lista = [1, 2, 3, 4]
def cuadrado(x):
    return x * x

lista_map = map(cuadrado, mi_lista)
type(lista_map)

list(lista_map)
```

[1, 4, 9, 16]

- filter(): Esta función extrae todos los elementos de una secuencia iterable para los cuales la función siempre retorna en true. El resultado es un objeto filter que se puede convertir en tuplas o listas. Tenemos que:

```
mi_lista = [1, 2, 3, 4]
def es_par(x):
    if x % 2 == 0:
        return True
    return False
iterador_num_pares = filter(es_par, mi_lista)
type(iterador_num_pares)

list(iterador_num_pares)

[2, 4]
```

Bibliografía:

<https://www.programaenpython.com/miscelanea/funciones-integradas/>

<https://j2logo.com/python/tutorial/python-if-sentencia/#python-if-basico>

Clase N.5

Fecha: 02/09/2022

Título: Sentencia If-else en Py.

Objetivo general: Descripción de la sentencia if-else en Python.

Objetivo específico: Conocer las características y propiedades de la sentencia if en python.

Resumen: En esta clase pudimos describir la sentencia if-else, conocer su sintaxis en python y aplicamos diferentes casos en los que se puede aplicar y las ventajas que tiene este condicional.

Marco teórico:

Sentencia if: Llegados a este punto en el conocimiento de nuestro lenguaje en el que ya podemos trabajar con condicionales y variables bien definidas para este caso tenemos la sentencia "if" el cual se utiliza para ejecutar un código si, y sólo si se cumple determinada condición. la estructura de este condicional es:

```
if sentencia o condicion:
    codigo de programacion
    .
    .
    .
```


es decir si solo se cumple la condición o está al evaluarla es un True entonces el código puede ejecutarse. En caso contrario, si la sentencia es False, no se evalúa ningún código. Para nuestro caso la condición puede ser literal, un valor numérico o una variable. Para este tipo de casos es muy común usar como sentencias operadores booleanos y de comparación.

```
x=2
if x<20:
    print("x es menor que 20")
```

x es menor que 20

Como podemos observar nuestro código se ejecutó porque la sentencia fue evaluada como True. Otro ejemplo de True o False. :

```
[3] valores=[1,2,3,4,5]
    if 5 in valores:
        print("Esta en valores")
```

Esta en valores

```
valores=[1,2,3,4]
if 5 in valores:
    print("Esta en valores")
print("No pertenece Fin")
```

No pertenece Fin

Mensaje importante: Recordar que en Py todas sentencias u objetos son evaluados a True excepto todas las condiciones vacías, 0, none, se evalúan false.

Sentencia if...else:

Debido a que la sentencia if es insuficiente porque sólo evalúa objetos a True, muchas veces vamos a requerir de evaluar objetos a False para ejecutar aplicaciones de nuestro código.

Por ejemplo:

```
x=10
y=2
if y !=0:
    R=x/y
    print("El resultado es: ",R)
else :
    print("No se puede hacer divisiones entre 0")
```

El resultado es: 5.0

```

▶ x=10
  y=0
  if y !=0:
      R=x/y
      print("El resultado es: ",R)
  else :
      print("No se puede hacer divisiones entre 0")

```

No se puede hacer divisiones entre 0

Sentencia if...elif...else: Debido a que en muchas ocasiones una decisión en un código depende de varias condiciones entonces se usa la sentencia “elif”: Por ejemplo:

```

▶ x = 28
  if x < 0:
      print("x es menor que 0 ")
  elif x > 0:
      print("x es mayor que 0 ")
  else:
      print('x es 0')

```

x es mayor que 0

Así vemos que al no cumplirse la sentencia del if entonces el programa ejecuta la sentencia 2 que pertenece al elif, si esta es True entonces pasa a ejecutar la línea de código que está bajo el elif, luego sino se cumplen ninguna en true entonces analiza el else e imprime la única sentencia evaluada false.

Ahora el siguiente caso es cuando se puede anidar un if a un else o un elif, por ejemplo:

```

▶ x = 28
  if x < 0:
      print(f'{x} es menor que 0')
  else:
      if x > 0:
          print(f'{x} es mayor que 0')
      else:
          print('x es 0')

```

28 es mayor que 0

Clase N.6

Fecha: 07/09/2022

Título: Bucle While en Py.

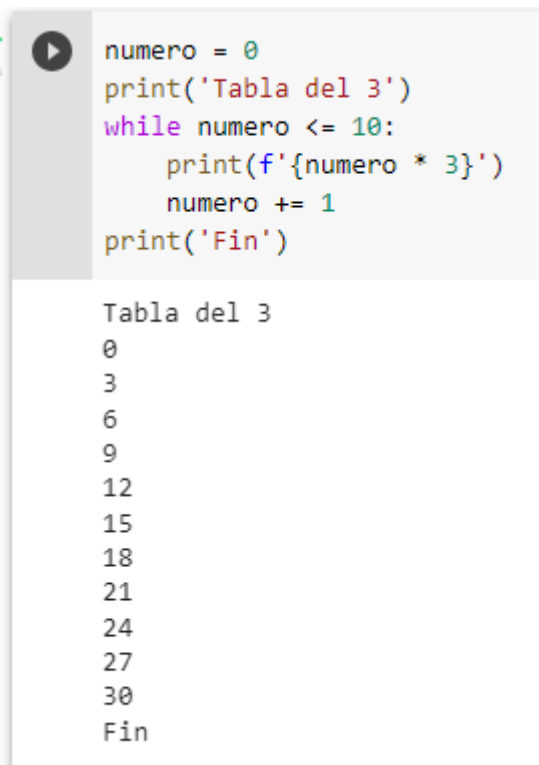
Objetivo general: Descripción del bucle while en Python.

Objetivo específico: Conocer las características y propiedades del bucle while en python.

Resumen: Para esta práctica hicimos una descripción breve del bucle while en python, teniendo en cuenta los casos en los que se puede aplicar y como romper un ciclo infinito de estos hicimos algunos ejemplos para comprender este tipo de bucle.

Marco teórico:

Bucle While: Este tipo de bucle maneja una estructura condicional similar al if ya que de ser cumplida una sentencia en True entonces el código se ejecutará en bloque como sigue después del while.



```
numero = 0
print('Tabla del 3')
while numero <= 10:
    print(f'{numero * 3}')
    numero += 1
print('Fin')
```

Tabla del 3
0
3
6
9
12
15
18
21
24
27
30
Fin

En el ejemplo anterior pudimos ver como al guardar el valor de 0 en la variable numero y como en la linea de codigo donde esta el while se cumple la condición impuesta por nuestra variable número que en este caso tiene el valor de cero, entonces se ejecuta el bloque de código hasta que número como al estar guardada por un contador entonces tenderá a aumentar hasta que rompa el código naturalmente justo cuando no se cumple la condición del while.

```
▶ valores = [5, 1, 9, 2, 7, 4]
eureka = False
indice = 0
long= len(valores)
while not eureka and indice < long:
    valor = valores[indice]
    if valor == 2:
        eureka = True
    else:
        indice += 1
if eureka:
    print(f'El número 2 ha sido encontrado en el índice {indice}')
else:
    print('El número 2 no se encuentra en la lista de valores')

El número 2 ha sido encontrado en el índice 3
```

Como puedes observar, en el ejemplo, se utilizan 3 variables de control:

- eureka: Indica si el número 2 ha sido encontrado en la lista.
- índice: Contiene el índice del elemento de la lista valores que va a ser evaluado.
- long: Indica el número de elementos de la lista de valores.

En esta ocasión, la condición de continuación del bucle *while* es que no se haya encontrado el número 2 y que el índice del elemento a evaluar sea menor que la longitud de la lista (es posible acceder a un elemento de una lista a partir del índice de su posición, comenzando por 0). Por tanto, el bucle finaliza bien cuando se haya encontrado el elemento, bien cuando se haya evaluado el último elemento de la lista. Si se ha encontrado el número 2, se muestra un mensaje indicando el índice en el que está. En caso contrario, se muestra un mensaje indicando que el número 2 no se encuentra en la lista.