30_EJERCICIOS_PYTHON_START_sebassilvap

March 1, 2024

EJERCICIO 01

Presentación de Datos en la Consola de Python

TEMAS: print / variables / tipos de variables / str.format() / f'String' / Formas de usar print() / print con parámetros / print con concatenación de Strings / print con parámetro opcional end / Secuencias de Escape

- Recordar los tipos de variables básicas en Python

Tipo de Dato	Denominación	EJ:
Entero	int	-20 , 0 , 5
Punto Flotante	float	-2.5 , 0.669 , -89.52
Cadena de Texto	str	'Hola' , "Python"
Booleano	bool	True , False , (1 y 0)

- Pero de manera completa y general tenemos la siguiente clasificación de los tipos de datos en Python:

Tipo	Dato en Python
Tipo Texto	str
Tipo Numérico	int / float / complex
Tipo Secuencia	list / tuple / range
Tipo de Mapeo	dict
Tipo de Conjunto	set / frozenset
Tipo Booleano	bool
Tipo Binario	bytes / bytearray / memoryview
Tipo None (Ninguno / Vacío)	NoneType

Fuente: w3schools

0.0.1 *Objetivo del Ejercicio*

- Imprimir en consola el siguiente texto:

```
Nombre : Daniel
Edad : 36
Este año, Daniel ¿es estudiante? : False
El próximo año, Daniel tendrá 37 años, y estudiará una maestría.
El próximo año, Daniel ¿será estudiante? : True
```

- Utilizar las siguientes variables:

```
nombre
edad
es_estudiante
```

- ATENCIÓN:
  - Los espacios, y el texto debe ser igual a como se indica.
  - EJ: espacio entre ¿, ?, o la coma, y el texto.
  - NO imprimir el texto tal y como se muestra.
  - Utilizar las variables.

0.0.2 *Repaso planificado por el instructor*

- Posiblemente este ejercicio es uno de los más sencillos que vamos a tener.
- Pero es la oportunidad perfecta para poder realizar un repaso de lo que son las variables y la función `print()`
- Al momento que se muestre la solución, el instructor va a presentar un repaso JUSTAMENTE NECESARIO de estos temas básicos.
- Debido a que la impresión de datos en la consola es la clave fundamental para la realización no solo de este sino de todos los problemas planteados en este curso de ejercicios.
- Por el momento se pide al estudiante que de todas las maneras que él o ella conozca, trate de dar solución a este ejercicio.

- BUENA SUERTE !!!

[ ]:
```python
# ====================
# NO HACER ESTO !!!!!
# ====================

cadena = """Nombre : Daniel
Edad : 36
Este año, Daniel ¿es estudiante? : False
El próximo año, Daniel tendrá 37 años, y estudiará una maestría.
El próximo año, Daniel ¿será estudiante? : True"""

print(cadena)
```

[ ]:
```python

```

EJERCICIO 02

Función para generar rótulos

TEMAS: print() / Funciones Normales / variables / str.format() / Casting / Métodos de string

- Crear el código Python para la función `generador_rotulo()`
  - `generador_rotulo( nombre , caracter )`
- Esta función recibe 2 parámetros:
  - **nombre** : la variable string con el nombre
  - **caracter** : una variable string con un símbolo
- De manera inicial se dan 3 variables de nombres:

```
# ==========================
# DATOS INICIALES
# => 3 variables de nombres
# ==========================

nombre_1 = 'Diego'
nombre_2 = 'Andrea'
nombre_3 = 'Valentina'
```

- Para verificar el correcto código de esta función, se ha diseñado el siguiente CÓDIGO TEST que el estudiante debe ejecutar:

```
# ============
# CÓDIGO TEST
# ============

generador_rotulo( nombre_1 , '*' )
print()
generador_rotulo( nombre_2 , '|' )
print()
generador_rotulo( nombre_3 , '-' )
```

- El resultado debe ser el siguiente:

```
D * I * E * G * O

A | N | D | R | E | A

V - A - L - E - N - T - I - N - A
```

- BUENA SUERTE !!!

[ ]:
```
# ==========================
# DATOS INICIALES
# => 3 variables de nombres
# ==========================

nombre_1 = 'Diego'
nombre_2 = 'Andrea'
nombre_3 = 'Valentina'
```

[ ]:

EJERCICIO 03

Programa de Calculadora Básica en Python

TEMAS: input() / variables / eval() / Funciones con parámetros y retorno / str.format()

- Generar un programa de calculadora en Python
- Interactuando con la consola
- Se pide al usuario 2 números
- Se presenta las operaciones básicas entre 2 números CON UN MÁXIMO DE 2 DECIMALES
- Considerando que el primer número es un 5 y el segundo número es un 3 al ejecutar la función `iniciar_calculadora()`, se deben tener los siguientes resultados:

```
iniciar_calculadora()
```

- Resultados en la consola:

```
Bienvenidos a nuestra Calculadora Básica en Python
Ingrese un primer número  : 5
Ingrese un segundo número : 3

Los resultados son los siguientes:
La suma de 5.0 + 3.0 = 8.0
La resta de 5.0 - 3.0 = 2.0
El producto de 5.0 * 3.0 = 15.0
La división de 5.0 / 3.0 = 1.67
5.0 elevado a la 3.0 = 125.0
El residuo de dividir 5.0 para 3.0 = 2.0

Gracias por utilizar nuestro programa...
```

- NOTAS IMPORTANTES:
  - El camino para realizar este programa es COMPLETAMENTE LIBRE
  - El estudiante puede crear todas las funciones que considere necesarias
  - NO es necesario evaluar el error en la división para CERO
  - NO es necesario realizar el programa en un bucle, es decir, el usuario proporciona los 2 números, los cálculos básicos se presentan y el programa finaliza.
  - Como se indicó arriba: LOS RESULTADOS SE PRESENTAN EN PUNTO FLOTANTE DE MÁXIMO 2 DECIMALES.
- BUENA SUERTE !!!

[ ]:

EJERCICIO 04

Función en Python para Encriptar una Contraseña

TEMAS: string y métodos / Funciones Normales / Condicional IF / Bucle FOR / Diccionarios / Métodos Diccionarios

- Para este ejercicio se da como dato inicial 2 CLAVES o PASSWORDS a manera de string, de las típicas que se usan para una cuenta de red social, para un correo electrónico, es decir, algún tipo de cuenta que manejamos en el internet.

```
# ========================
# DATO: clave a encriptar
# ========================

clave_1 = 'suPER123passWORD'
clave_2 = '89esUNaCLave78DificiL35'
```

- Se pide desarrollar el código en Python para la crear la función `encriptar_password()` que reciba como parámetro una variable `password` (las que se dan de dato)
- Y su objetivo es tomar la clave original y crear una clave "encriptada" que la haga aún más difícil de leer.
- La lógica de como encriptar la clave se explica a continuación, pero antes, veamos como quedarían las claves encriptadas al aplicar el método. Esto se lo hace ejecutando el siguiente CÓDIGO TEST:

```
# ============
# CÓDIGO TEST
# ============

clave_encriptada_1 = encriptar_password( clave_1 )
clave_encriptada_2 = encriptar_password( clave_2 )

# RESULTADOS
print( 'CLAVE 1' )
print( 'CLAVE ORIGINAL   :  ' , clave_1 )
print( 'CLAVE ENCRIPTADA :  ' , clave_encriptada_1 )

print( '\nCLAVE 2' )
print( 'CLAVE ORIGINAL   :  ' , clave_2 )
print( 'CLAVE ENCRIPTADA :  ' , clave_encriptada_2 )
```

- El resultado sería el siguiente:

```
CLAVE 1
CLAVE ORIGINAL   :   suPER123passWORD
CLAVE ENCRIPTADA :   S&&&P###RBBBCCCDDDP!!!SSW%%%RD

CLAVE 2
CLAVE ORIGINAL   :   89esUNaCLave78DificiL35
CLAVE ENCRIPTADA :   IIIJJJ###S&&&N!!!CL!!!V###HHHIIID$$$F$$$C$$$LDDDFFF
```

0.0.3 *LÓGICA DE LA ENCRIPTACIÓN*

- Como se puede observar, al final todo el string queda en mayúsculas.

- Las vocales de la contraseña original son reemplazadas por los siguientes caracteres
  - a = !!!
  - e = ###
  - i = $$$
  - o = %%%
  - u = &&&
- Los números son reemplazados por las primera letra del abecedario, repetida 3 veces, de la siguiente manera (empezamos desde el dígito 0 y terminamos con el 9):
  - 0 = AAA
  - 1 = BBB
  - 2 = CCC
  - 3 = DDD
  - 4 = EEE
  - 5 = FFF
  - 6 = GGG
  - 7 = HHH
  - 8 = III
  - 9 = JJJ
- Con esta explicación, se pide crear la función que permita realizar esta encriptación.
- RECUERDEN: Puede haber más de una manera de llegar a la solución
- BUENA SUERTE !!!

[ ]:
```
# ========================
# DATO: clave a encriptar
# ========================

clave_1 = 'suPER123passWORD'
clave_2 = '89esUNaCLave78DificiL35'
```

[ ]:
```

```

EJERCICIO 05

Función en Python para convertir números enteros en binarios

TEMAS: Función Normal / str.format() / bucle WHILE / métodos de string / str / Listas / Aritmética en Python / Módulo %

- En este ejercicio se pide generar una función en Python para generar el equivalente en binario a partir de un número entero.
- Para entender la teoría de los números binarios se presenta el siguiente ejemplo:

0.0.4 *Lógica de los números binarios*

- Como se puede observar de los ejemplos mostrados.
  - 19 en número binario = 10011

  - 43 en número binario = 101011
- La formación del número binario es dividiendo el número entero para 2, de manera continua hasta que el cociente final sea 1
- Este cociente final es el punto de partida del equivalente binario
- Y a partir de este cociente se comienzan a añadir los residuos de cada operación de división hasta llegar al cociente final.
- Obsérvese la dirección de la flecha en los ejemplos de arriba, con lo cual se indica el orden en el cual establecemos la formación del número binario.
- **RECORDAR :** Para obtener el residuo directamente de una operación de división podemos utilizar el operador de módulo `%` en Python.

0.0.5 *Recordatorio sobre el operador módulo*

- Básicamente el módulo `%` es un operador que nos devuelve el residuo de una división.
- La división en Python es con `/`
- De manera adicional también tenemos la división + floor que se obtiene con un doble operador `//`
- A continuación se presenta un código preparado para ejemplificar el funcionamiento básico del `%`, `/` y `//`

```
# ================================================
# DATO IMPORTANTE 1
# RECORDAR EL FUNCIONAMIENTO DEL OPERADOR MÓDULO %
# ================================================

# % => devuelve el residuo de una división
# EJ:
"""
9   |____5
4        1
"""

# Las partes de la división serían:
# Dividendo = 9
# Divisor = 5
# Cociente = 1
# Residuo (Módulo) = 4

# Esto lo podemos obtener con %
print( 9 / 5 )  # división normal => resultado float
print( 9 // 5 ) # división + floor
print( 9 % 5 )  # módulo ó residuo
```

- La ejecución de este código nos daría el siguiente resultado:

```
1.8
1
4
```

- **NOTA IMPORTANTE :** Cuando el DIVIDENDO es mayor al DIVISOR, el operador

módulo % nos devuelve el DIVIDENDO

```
# ================================
# DATO IMPORTANTE 2
# GENERALIDADES SOBRE EL OPERADOR %
# ================================

# ¿qué pasaría si intentamos 3 % 5?

print( 3 / 5 ) # 0.6
print( 3 % 5 ) # 3

# => cuando el dividendo es menor que el divisor
# => el módulo es igual al dividendo
```

- El anterior código nos daría el siguiente resultado:

```
0.6
3
```

0.0.6 *Objetivo del ejercicio*

- Crear la función en Python: **`generar_binario(numero)`**
- Esta función acepta un número entero como argumento de entrada y su función consiste en devolver el valor binario
- **NOTA IMPORTANTE :** El valor binario también se lo puede obtener con la función interna de Python `bin(numero_entero)`
- De esta manera podemos comprobar si la lógica de nuestra función es la correcta.
- Para comprobación de nuestro programa, podemos ejecutar el siguiente código de prueba:

```
# ==================
# CÓDIGO TEST FINAL
# ==================

print('Binario de 19 : {} | {}'.format( bin(19) , generar_binario(19) ))
print('Binario de 43 : {} | {}'.format( bin(43) , generar_binario(43) ))
print('Binario de 17 : {} | {}'.format( bin(17) , generar_binario(17) ))
print('Binario de 9 : {} | {}'.format( bin(9) , generar_binario(9) ))
print('Binario de 5 : {} | {}'.format( bin(5) , generar_binario(5) ))
print('Binario de 99 : {} | {}'.format( bin(99) , generar_binario(99) ))
```

- Lo cual nos debe dar el siguiente resultado:

```
Binario de 19 : 0b10011 | 10011
Binario de 43 : 0b101011 | 101011
Binario de 17 : 0b10001 | 10001
Binario de 9 : 0b1001 | 1001
Binario de 5 : 0b101 | 101
Binario de 99 : 0b1100011 | 1100011
```

- BUENA SUERTE !!!

[ ]:
```
# ================================================
# DATO IMPORTANTE 1
# RECORDAR EL FUNCIONAMIENTO DEL OPERADOR MÓDULO %
# ================================================

# % => devuelve el residuo de una división
# EJ:
"""
9   |____5
4        1
"""

# Las partes de la división serían:
# Dividendo = 9
# Divisor = 5
# Cociente = 1
# Residuo (Módulo) = 4

# Esto lo podemos obtener con %
print( 9 / 5 )  # división normal => resultado float
print( 9 // 5 ) # división + floor
print( 9 % 5 )  # módulo ó residuo
```

```
1.8
1
4
```

[ ]:
```
# ==================================
# DATO IMPORTANTE 2
# GENERALIDADES SOBRE EL OPERADOR %
# ==================================

# ¿qué pasaría si intentamos 3 % 5?

print( 3 / 5 ) # 0.6
print( 3 % 5 ) # 3

# => cuando el dividendo es menor que el divisor
# => el módulo es igual al dividendo
```

```
0.6
3
```

[ ]:
```

```

EJERCICIO 06

Interacción Inteligente con la computadora - Mini bot para preguntar nombre y edad

TEMAS: bucle while / variables / condicional if / métodos string / try-except / casting de tipos básicos / operadores de comparación / operadores lógicos / print / string.format() / input

- Preguntar al usuario su nombre y su edad
- Si el usuario ingresa un valor que no sea un número entero para la edad, presentar un error en la consola y volver a repetir la verificación de la edad
- Una vez ingresado correctamente un nombre y una edad presentar el siguiente mensaje en la terminal:

```
¿Cuál es su nombre? :     seBastiÁn silvA
¿Cuál es su edad? : 17
Su nombre es Sebastián Silva y usted tiene 17 años.
```

- NOTA:
  - El nombre debe presentarse sin espacios a la izquierda o derecha y en formato de título
- BUENA SUERTE !!!

[ ]:

EJERCICIO 07

Movimiento de un Robot con funciones de Python

TEMAS: Condicional IF / string.format() / funciones con parámetros por defecto

- Se pide crear la función de movimiento de un robot
- La función se llama `move(x,y)`
- Recibe 2 parámetros: `x`, `y`
  - `x` => representa el movimiento en centímetros a izquierda o derecha (dependiendo si es positivo o negativo)
  - `y` => representa el movimiento en centímetros arriba o abajo (dependiendo si es positivo o negativo)
- El objetivo de este ejercicio es definir esta función, de tal manera que al ejecutar el siguiente test:

```
# => TEST
move(5,-8)
print()
move(-15.4589,20)
print()
move(0,80.7562)
print()
move(0,0)
print()
move()
```

- Tengamos el siguiente resultado en consola:

```
Movimiento de 5.00 cm a la DERECHA
Movimiento de -8.00 cm hacia ABAJO
```

```
Movimiento de -15.46 cm a la IZQUIERDA
Movimiento de 20.00 cm hacia ARRIBA

No hay movimiento HORIZONTAL
Movimiento de 80.76 cm hacia ARRIBA

No hay movimiento HORIZONTAL
No hay movimiento VERTICAL

Movimiento de 1.00 cm a la DERECHA
Movimiento de 1.00 cm hacia ARRIBA
```

- NOTA IMPORTANTE: Como se puede observar, la función puede no aceptar ningún parámetro, en tal caso, pues el movimiento siempre debe ser de 1 cm tanto, a la derecha y arriba respectivamente
- BUENA SUERTE !!!

[ ]:

EJERCICIO 08

Función para dibujar una rampa con símbolos

TEMAS: Funciones / Condicional IF / Bucle FOR / Listas / str.format() / range / Funciones con parámetros por defecto

- Desarrollar un código en Python para una función llamada `formar_rampa(caracter,size)`
- Básicamente, esta función recibe 2 parámetros:
  - **`caracter`** = un caracter de tipo String para dibujar la silueta de una rampa.
  - **`size`** = el tamaño (número de pisos) que va a tener esta rampa.
- Esta rampa se dibuja de manera creciente, es decir crece hacia abajo.

0.0.7 *NOTAS IMPORTANTES:*

- El usuario no puede pasar un size de CERO, o menor a CERO.
- Si el size no se pasa como parámetro, por defecto será de 5

0.0.8 *Código Test*

- Para explicar mejor el funcionamiento de esta rampa, tomar en consideración el siguiente código test:

```
# ============
# CÓDIGO TEST
# ============

formar_rampa( '*' , 8 )
formar_rampa( 'O' )
formar_rampa( '#' , 3 )
```

```
formar_rampa( 'X' , 0 )
```

- La funcionalidad de nuestra rampa debe ser tal, que al ejecutar este código test, debemos ver el siguiente resultado en la terminal de Python:

```
*
* *
* * *
* * * *
* * * * *
* * * * * *
* * * * * * *
* * * * * * * *
* * * * * * * * *
---------------------------------

O
O O
O O O
O O O O
O O O O O
---------------------------------

#
# #
# # #
---------------------------------

ERROR - El tamaño debe ser al menos de 1 !!!
---------------------------------
```

0.0.9 *OJO !!!*

- Al final de dibujar la rampa, se coloca de manera opcional esta separación con guiones para indicar al usuario nada más, un dibujo de rampa separado de otro.
- Esto es algo opcional, pero solo implica una pequeña línea de código que se debe poner en algún lugar de nuestra función.
- BUENA SUERTE !!!

[ ]:

EJERCICIO 09

Función: dibujar_rampa() / Versión Completa / Dibujando una rampa normal y una rampa inversa

TEMAS: Funciones / Condicional IF / Bucle FOR / Listas / str.format() / range / Funciones con parámetros por defecto

- Tomando en consideración el anterior ejercicio planteado (Función para dibujar una rampa con símbolos)

- En ese ejercicio nosotros creamos la función en Python `formar_rampa()`
- Que como recordaremos, aceptaba 2 parámetros:
  - **`caracter`** = (símbolo string para dibujar la rampa)
  - **`size`** (tamaño, número de pisos de la rampa, por defecto = 5)
- Recordar que esta rampa anterior se dibujaba de manera creciente, es decir empezaba con un símbolo y aumentaba en cada línea en función del tamaño o size puesto como parámetro
- Este ejercicio, de manera completa pide 2 cosas:
  - **[1]** Desarrollar una función en Python **`dibujar_rampa_inversa()`** para dibujar una rampa decreciente, es decir que empiece con un número máximo de caracteres y en cada línea vaya disminuyendo hasta llegar a 1.
  - **[2]** Desarrollar una segunda función en Python llamada **`dibujar_rampa()`**, cuyo objetivo es unir la experiencia del ejercicio anterior y la primera parte de este ejercicio con el objetivo de tener una función general para dibujar rampa.

0.0.10 *Función: `dibujar_rampa( caracter , normal_inversa , size )`*

- Con el fin de generalizar el dibujo de la rampa, se plantea esta función en general **`dibujar_rampa()`**
- NOTA ESPECIAL: Esta segunda parte tiene más sentido hacerla cuando hemos completado el ejercicio anterior y la primera parte de este ejercicio. Es decir, nosotros debemos tener ya presente la funcionalidad de:
  - `formar_rampa()`, y
  - `dibujar_rampa_inversa()`
- POR FAVOR, completar estas 2 primeras partes antes de entrar a esta segunda parte.
- Tal y como se describe en el título, para esta función se plantean 3 parámetros:
  - **`caracter`** = de igual manera representa el caracter en forma de String para dibujar la rampa
  - **`normal_inversa`** = este parámetro expresa el deseo de dibujar una rampa normal ó creciente (igual a lo que hace la función "formar_rampa" del ejercicio anterior), para lo cual el parámetro vale 1. En caso de que deseemos dibujar una rampa inversa como lo plantea la primera parte de este ejercicio, entonces este parámetro vale -1. Como NOTA IMPORTANTE, por defecto este parámetro vale 1 positivo, es decir, por defecto nosotros dibujamos una rampa creciente.
  - **`size`** = de igual manera que las 2 anteriores funciones, este parámetro representa el tamaño o número de pisos que tendrá la rampa.

0.0.11 *Código Test*

- Tal y como se mencionó, este ejercicio tendrá 2 partes.
- La primera parte y el ejercicio anterior se complementan para la segunda parte de este ejercicio.
- Necesitamos entonces consolidar 2 funciones en Python:
  - 1) **`dibujar_rampa_inversa()`**, y
  - 2) **`dibujar_rampa()`**
- Una vez que se tenga el código de estas funciones podemos plantear los siguientes códigos test:

- **A) Para la función: `dibujar_rampa_inversa()`**

```
# ==============
# CÓDIGO TEST
# PARTE 1
# ==============

dibujar_rampa_inversa( '*' , 8 )
dibujar_rampa_inversa( 'O' )
dibujar_rampa_inversa( '#' , 3 )
dibujar_rampa_inversa( 'X' , 0 )
```

El resultado será el siguiente:

```
* * * * * * * *
* * * * * * *
* * * * * *
* * * * *
* * * *
* * *
* *
*
---------------------------------

O O O O O
O O O O
O O O
O O
O
---------------------------------

# # #
# #
#
---------------------------------

ERROR - El tamaño debe ser al menos de 1 !!!
---------------------------------
```

- **B) Para la función: `dibujar_rampa()`**

```
# ==============
# CÓDIGO TEST
# PARTE 2
# ==============

# dibujar_rampa( caracter , normal_inversa , size )

dibujar_rampa( '@' , -1 , 8 )
dibujar_rampa( '*' , 1 , 8 )
dibujar_rampa( '&' )
dibujar_rampa( '$', -1 )
```

```
dibujar_rampa( 'A' , 1 , 3 )
dibujar_rampa( 'Z' , -1 , 3 )
dibujar_rampa( 'X' , 0 , 0 )
dibujar_rampa( 'X' , 10 , 20 )
dibujar_rampa( 'X' , 1 , 0 )
```

El resultado será el siguiente:

```
@ @ @ @ @ @ @ @
@ @ @ @ @ @ @
@ @ @ @ @ @
@ @ @ @ @
@ @ @ @
@ @ @
@ @
@
---------------------------------

*
* *
* * *
* * * *
* * * * *
* * * * * *
* * * * * * *
* * * * * * * *
---------------------------------

&
& &
& & &
& & & &
& & & & &
---------------------------------

$ $ $ $ $
$ $ $ $
$ $ $
$ $
$
---------------------------------

A
A A
A A A
---------------------------------

Z Z Z
Z Z
```

```
Z
--------------------------------

ERROR - El tamaño debe ser al menos de 1 !!!
ERROR - el parámetro normal_inversa puede ser 1 (rampa creciente) ó -1 (rampa decreciente)
--------------------------------

ERROR - el parámetro normal_inversa puede ser 1 (rampa creciente) ó -1 (rampa decreciente)
--------------------------------

ERROR - El tamaño debe ser al menos de 1 !!!
--------------------------------
```

0.0.12 *OJO !!!*

- Observar en el segundo test, al ejecutar:
- `dibujar_rampa( 'X' , 0 , 0 )`
- Los 2 errores se muestran de manera simultánea:
  - Cuando el size es puesto como CERO
  - Cuando normal_inversa es DIFERENTE de 1 y -1
- Tomar esto en cuenta al momento de diseñar el código para `dibujar_rampa()`
- BUENA SUERTE !!!

[ ]:

EJERCICIO 10

Cálculo de las Soluciones / Raíces de una Ecuación Cuadrática por medio de Funciones en Python

TEMAS: Funciones Normales / Casting / str.format() / Diccionarios

- Crear una función que permita al usuario calcular las raíces / soluciones de una ecuación cuadrática.
- Una ecuación cuadrática es por ejemplo:

  0.0.13 `5 * x^2 - 2 * x - 3 = 0`

- Que se puede también poner de la siguiente manera:

  0.0.14 `a * x^2 + b * x + c = 0`

- La fórmula de la ecuación cuadrática, nos indica y sugiere que existen dos valores de x que pueden satisfacer esta ecuación:

  0.0.15 `X1 = ( -b + sqrt(b^2 - 4*a*c) ) / 2*a`

  0.0.16 `X2 = ( -b - sqrt(b^2 - 4*a*c) ) / 2*a`

- En la siguiente imagen podemos observar la teoría de resolución de las ecuaciones cuadráticas mediante esta fórmula general presentada:

- `X1` y `X2` son las soluciones o también llamadas las raíces de la ecuación cuadrática que satisfacen la ecuación
- OBJETIVO: Básicamente se pide crear una función en python `calculo_x1_x2()` que reciba como parámetros `a`, `b` y `c`
- Al final se pide retornar la solución de estas raíces en forma de diccionario.
- Se puede ejecutar el siguiente test como referencia:

```
# TEST
print( calculo_x1_x2(5,-2,-3) )
print( calculo_x1_x2(-6,-5,10) )
```

- La solución en consola debería ser:

```
{'x1': 1.0, 'x2': -0.6}
{'x1': -1.7732, 'x2': 0.9399}
```

- NOTAS IMPORTANTES:
  - Retornar la solución con máximo 4 decimales y de tipo float (los valores de las claves del diccionario)
  - Las claves del diccionario deben ser `x1` y `x2` respectivamente
- BUENA SUERTE !!!

[ ]:

EJERCICIO 11

Función en Python para operaciones aritméticas de arreglos

0.1 Función para Aritmética de Arreglos

TEMAS: Funciones / Condicional IF / Bucle FOR / Listas y Métodos / str.format() / eval()

- El objetivo de este ejercicio es realizar una función en Python para realizar aritmética de arreglos.
- Arreglo = Lista unidimensional de valores numéricos
- EJ: `arreglo = [10, -5, 9.8, 2]`
- Es decir si tenemos, por ejemplo los siguientes 2 arreglos sencillos:
  - `arr_1 = [1, 2, 3]`
  - `arr_2 = [4, 5, 6]`
- El resultado de una SUMA por ejemplo sería:
  - `resultado = [5, 7, 9]`

0.1.1 *ANTES QUE NADA: NO usar numpy !!*

- Como deben de saber, esto es posible y sencillo usando numpy y transformando la lista de valores numéricos a un array de numpy, pero este ejercicio consiste en desarrollar la lógica de programación que permita a nuestra función realizar estas operaciones aritméticas entre arreglos.

- Al inicio del ejercicio se proporcionan los siguientes datos iniciales:

```
# =================
# DATOS INICIALES
# =================

arreglo_1 = [-5, 2, 4, -1]
arreglo_2 = [3, 2.5, -1.8, -6.7]
arreglo_3 = [-5, 4.66, 2]
```

- El objetivo es desarrollar la función en Python `aritmetica_de_arreglos()` que permita ejecutar el siguiente código test:

```
# ===============
# TEST DE CÓDIGO
# ===============

aritmetica_de_arreglos( arreglo_1, arreglo_2, '+' )
aritmetica_de_arreglos( arreglo_1, arreglo_2, '-' )
aritmetica_de_arreglos( arreglo_1, arreglo_2, '*' )
aritmetica_de_arreglos( arreglo_1, arreglo_2, '/' )
aritmetica_de_arreglos( arreglo_1, arreglo_2, '**' )
aritmetica_de_arreglos( arreglo_1, arreglo_2, '%' )
aritmetica_de_arreglos( arreglo_1, arreglo_2, '$' )
aritmetica_de_arreglos( arreglo_1, arreglo_3, '+' )
```

- Su código debe ser tal, que esperamos los siguientes resultados en la terminal:
  - **NOTA IMPORTANTE:** Los resultados se muestran como float de MÁXIMO 2 decimales !!!

```
El resultado de la operación "+"
Entre array_1 = [-5, 2, 4, -1]
Y el  array_2 = [3, 2.5, -1.8, -6.7]
Es igual a = [-2.0, 4.5, 2.2, -7.7]
---------------------------------------

El resultado de la operación "-"
Entre array_1 = [-5, 2, 4, -1]
Y el  array_2 = [3, 2.5, -1.8, -6.7]
Es igual a = [-8.0, -0.5, 5.8, 5.7]
---------------------------------------

El resultado de la operación "*"
Entre array_1 = [-5, 2, 4, -1]
Y el  array_2 = [3, 2.5, -1.8, -6.7]
Es igual a = [-15.0, 5.0, -7.2, 6.7]
---------------------------------------

El resultado de la operación "/"
Entre array_1 = [-5, 2, 4, -1]
```

```
Y el  array_2 = [3, 2.5, -1.8, -6.7]
Es igual a = [-1.67, 0.8, -2.22, 0.15]
---------------------------------------

El resultado de la operación "**"
Entre array_1 = [-5, 2, 4, -1]
Y el  array_2 = [3, 2.5, -1.8, -6.7]
Es igual a = [-125.0, 5.66, 0.08, -1.0]
---------------------------------------

El resultado de la operación "%"
Entre array_1 = [-5, 2, 4, -1]
Y el  array_2 = [3, 2.5, -1.8, -6.7]
Es igual a = [1.0, 2.0, -1.4, -1.0]
---------------------------------------

ERROR - Operador No Existe !!
----------------------------

ERROR - Los 2 arreglos deben tener el MISMO tamaño!
--------------------------------------------------
```

- BUENA SUERTE !!!

[ ]:
```
# =================
# DATOS INICIALES
# =================

arreglo_1 = [-5, 2, 4, -1]
arreglo_2 = [3, 2.5, -1.8, -6.7]
arreglo_3 = [-5, 4.66, 2]
```

[ ]:
```

```

EJERCICIO 12

Recreación del Cálculo del Factorial de un Número con Funciones Normales y Funciones Recursivas

TEMAS: Condicional IF / Bucles / funciones normales / funciones recursivas

- Crear una función normal y una función recursiva para calcular el factorial de un número
- Como dato extra, el factorial de un número obedece a la siguiente regla matemática:
  - ***factorial 3 = 3! = 3 x 2 x 1 = 6***
  - ***factorial 4 = 4! = 4 x 3 x 2 x 1 = 4 x factorial 3 = 24***
  - ***factorial 5 = 5! = 5 x 4 x 3 x 2 x 1 = 5 x factorial 4 = 120***
- Recordar:
  - Se puede usar la función math.factorial de la librería “math” como referencia de cálculo

- OBJETIVO: Llegar a formular el cálculo del factorial con función normal y con función recursiva (es decir se deben plantear 2 soluciones para este ejercicio)
- Ejecutar el siguiente test:

```
print( factorial(3) )
print( factorial(4) )
print( factorial(5) )
```

- El resultado en consola debe ser respectivamente:

```
6
24
120
```

- BUENA SUERTE !!!

[ ]:

EJERCICIO 13

Función en Python para el cálculo de la velocidad

TEMAS: try-except / try-except + else / try-except + finally / None / Condicional IF / Funciones Normales / Listas / str.format()

- La Velocidad obedece a la siguiente fórmula física:

0.1.2 `VELOCIDAD = DISTANCIA / TIEMPO`

- Crear una función en Python que se llame `calcular_velocidad(distancia, tiempo)` que cumpla lo siguiente:
  - Tiene que aceptar dos parámetros: `distancia` y `tiempo`
  - El tiempo no puede ser CERO, caso contrario el error de división para cero por default de Python debe mostrarse, PERO, el programa no se debe de cortar.
  - La distancia y el tiempo deben ser siempre valores numéricos, caso contrario un error de Tipo por default establecido en Python debe ejecutarse. IGUALMENTE, esto no debe cortar la ejecución del programa.
  - De manera adicional el usuario debe definir 2 tipos personalizados de error al momento de ejecutar esta función:
    * 1) El tiempo no debe ser menor a 1
    * 2) La distancia no puede tener un valor negativo
    * IGUALMENTE, la ejecución del programa no se debe cortar incluso por estos errores personalizados definidos por el usuario.
  - En caso de que la función se ejecute correctamente, el resultado debe mostrarse de la siguiente manera:

```
Distancia = 25.15
Tiempo    = 17.86
```

```
--------------------
Velocidad = 1.41
```

- FINALMENTE, así exista errores (definidos o personalizados), o en caso que la función se ejecute correctamente sin ningún error, la función debe RETORNAR los valores de distancia, tiempo y velocidad (en este orden) en forma de LISTA.
- En caso de que exista un error, la velocidad deberá mostrarse en esta lista como un valor `None`

0.1.3 CÓDIGO TEST

- Para simplificar el entendimiento del estudiante, se ha preparado un código test / de prueba para la función `calcular_velocidad()`
- El siguiente código:

```
# =============
# CÓDIGO TEST
# =============

a = calcular_velocidad(5, 2)
b = calcular_velocidad(25.14896, 17.8561)
c = calcular_velocidad('a', 20)
d = calcular_velocidad('Z', 'Y')
e = calcular_velocidad(20, 0.5)
f = calcular_velocidad(-80, 10)
g = calcular_velocidad(50, 0)

print('**************************\n\n')

print('a =' , a)
print('b =' , b)
print('c =' , c)
print('d =' , d)
print('e =' , e)
print('f =' , f)
print('g =' , g)
```

- Debe dar la siguiente solución en la terminal:

```
Distancia = 5.00
Tiempo    = 2.00
--------------------
Velocidad = 2.50

Distancia = 25.15
Tiempo    = 17.86
--------------------
Velocidad = 1.41

TypeError | unsupported operand type(s) for /: 'str' and 'int'
```

```
ERROR - Por favor solo se aceptan números

TypeError | unsupported operand type(s) for /: 'str' and 'str'
ERROR - Por favor solo se aceptan números

Exception | ERROR - Tiempo no puede ser menor a 1
ERROR - El cálculo de la velocidad no es posible...

Exception | ERROR - Distancia no puede ser menor a 0
ERROR - El cálculo de la velocidad no es posible...

ZeroDivisionError | division by zero
ERROR - Tiempo no puede ser CERO

**************************


a = [5, 2, 2.5]
b = [25.14896, 17.8561, 1.4084240119622984]
c = ['a', 20, None]
d = ['Z', 'Y', None]
e = [20, 0.5, None]
f = [-80, 10, None]
g = [50, 0, None]
```

- BUENA SUERTE !!!

[ ]:

EJERCICIO 14

Juego de Ruleta Player VS. Computadora en Python

TEMAS: Librería Random / Condicional IF / Condicional ANIDADO / Bucle while True / input()

- Generar un juego con interacción del usuario en la terminal.
- El usuario debe ser preguntado un número del 1 al 10.
- La computadora genera de manera aleatoria un número del 1 al 10.
- Quien saque mayor valor gana.
- NOTA IMPORTANTE: hacerlo de manera básica, no vamos a controlar el error en caso de que el usuario no ponga un número sino por ejemplo una letra o cualquier otra cosa que no sea un número.
- Lo que si vamos a verificar es que el número dado por el usuario esté entre 1 y 10, caso contrario vamos a presentar un error y vamos a repetir la iteración.
- A continuación se muestra un flujo de ejemplo de cómo debería funcionar el programa:

```
--------------------------------
¿Quién saca el número más alto?
1) Jugar
```

```
2) Salir del Juego (s/q)

Elija su opción : 1

Escriba un número del 1 al 10 : 80
ERROR - El número debe ser entre 1 y 10


--------------------------------
¿Quién saca el número más alto?
1) Jugar
2) Salir del Juego (s/q)

Elija su opción : 1

Escriba un número del 1 al 10 : 10
Opción Player = 10
Opción CPU = 2
PLAYER GANA !!


--------------------------------
¿Quién saca el número más alto?
1) Jugar
2) Salir del Juego (s/q)

Elija su opción : 1

Escriba un número del 1 al 10 : -8
ERROR - El número debe ser entre 1 y 10


--------------------------------
¿Quién saca el número más alto?
1) Jugar
2) Salir del Juego (s/q)

Elija su opción : 8
ERROR - Opción Equivocada !!!


--------------------------------
¿Quién saca el número más alto?
1) Jugar
2) Salir del Juego (s/q)

Elija su opción :    q
Gracias por JUGAR !!
```

- BUENA SUERTE !!!

[ ]:

EJERCICIO 15

Generación de todas las Combinaciones posibles en una lista de elementos

TEMAS: Funciones Normales / Librería random / Librería math / Bucles / Condicionales / Listas y Métodos / Mutabilidad / Inmutabilidad / Dirección en la Memoria de las Variables / hex(id())

- Supongamos tenemos una lista de 3 Elementos: #### ***[x, y, z]***
- ¿Cuántas combinaciones posibles puedo generar a partir de esta lista?
- Para una lista de 3 elementos, si trato de resolver esto a mano, pues tendré 6 posibles combinaciones, y estas serían: #### ***Combinación 1 : [x, y, z]*** #### ***Combinación 2 : [x, z, y]*** #### ***Combinación 3 : [y, x, z]*** #### ***Combinación 4 : [y, z, x]*** #### ***Combinación 5 : [z, x, y]*** #### ***Combinación 6 : [z, y, x]***
- ¿El 6 es un número arbitrario?
- **NO** => El número total de combinaciones, está ligado al número total de elementos de una lista y obedece a una fórmula sencilla:

  Núm de Combinaciones = Factorial(Núm de Elementos)

0.1.4 *OBJETIVO:*

- Realizar un programa en Python que a partir de 2 listas dadas
- Me genere como resultado una lista de listas, donde cada lista sea una posible combinación
- Tomando en cuenta el ejemplo anterior:

```
lista_ejemplo = ['x', 'y', 'z']

funcion_a_realizar(lista_ejemplo)
```

- Esta función a realizar puede ser una o varias, eso queda a libre selección del estudiante, pero al final debemos ver en la consola algo como esto:

```
Combinación #  1 : ['y', 'x', 'z']
Combinación #  2 : ['y', 'z', 'x']
Combinación #  3 : ['x', 'y', 'z']
Combinación #  4 : ['x', 'z', 'y']
Combinación #  5 : ['z', 'y', 'x']
```

```
Combinación #  6 : ['z', 'x', 'y']
```

0.1.5 *PLANTEAMIENTO DE DATOS:*

- Tomar en cuenta las siguientes listas para este ejercicio:

```
lista_3_elementos = ['A', 'B', 'C']
lista_4_elementos = ['A', 'B', 'C', 'D']
```

0.1.6 *NOTA PRINCIPAL / SUGERENCIA:*

- Se necesita importar algunas librerías, al menos 1 es imprescindible para la realización de este ejercicio.
- La otra es opcional
- Utilizar funciones
- Si una función resulta en varias líneas de código, dividir la funcionalidad en más funciones.

0.1.7 *REPASO JUSTAMENTE NECESARIO*

- Para este ejercicio, el tutor ha preparado un repaso NECESARIO sobre 3 temas que generan mucha confusión no solo en Python sino en la programación en sí.
- Me refiero a los temas: Inmutabilidad, Mutabilidad y Dirección de las Variables en la Memoria
- Al momento de mostrar el video con la solución, el tutor desarrollará este repaso para el estudiante.

- BUENA SUERTE !!!

[ ]:
```
# ========================================================
# REPASO JUSTAMENTE NECESARIO
# Mutabilidad / Inmutabilidad / Dirección en la Memoria
# ========================================================


# ------------------------------------------------------------------------------
# 1) Todas las variables creadas en Python tienen una dirección en la memoria
# ------------------------------------------------------------------------------
#    ==> hex(id())
print('\n-----1-----')
```

```
# ---------------------------------------------------------
# 2) Las variables básicas son INMUTABLES (str, int, float)
# ---------------------------------------------------------
#    ==> INMUTABLE: no pueden cambiar
print('\n-----2-----')



# -----------------------------------------
# 3) Las colecciones, ej: Listas son MUTABLES
# -----------------------------------------
#    ==> MUTABLE: puede cambiar
print('\n-----3-----')



#␣
 ↪-------------------------------------------------------------------------------------------
# 4) Reasignación en tipos básicos no necesariamente cambia la dirección en␣
 ↪memoria
#␣
 ↪-------------------------------------------------------------------------------------------
print('\n-----4-----')



# -----------------------------------------
# 5) Asignar un tipo básico en función a otro
# -----------------------------------------
print('\n-----5-----')



# ---------------------------------------------------------
# 6) CUIDADO !! Cuando creamos una lista a partir de otra
# ---------------------------------------------------------
print('\n-----6-----')



# -------------------------------------
# 7) SOLUCIÓN: método .copy() de listas
# -------------------------------------
print('\n-----7-----')
```

[ ]:

EJERCICIO 16

Juego de Adivinazas en Python - El Ahorcado

TEMAS: Funciones / Bucle FOR / Bucle WHILE / Listas / Método .append de LISTAS / Condicional IF / str.format() / Controladores de Flujo: break, continue / Librería random / set / input()

- De manera inicial se proporciona el siguiente código:

```
# ===============
# Código Inicial
# ===============

import random

palabras = ['guitarra', 'gato', 'sol', 'luna', 'profe']
```

- Generar un código en Python con una función o serie de funciones donde al final tengamos solo la siguiente línea y la siguiente función final `jugar_ahorcado()`:

```
# ================
# Ejecutar Juego
# ================

jugar_ahorcado()
```

- El objetivo de este ejercicio vendría a ser el siguiente:
  - De la lista de `palabras`, una de ellas se selecciona de manera aleatoria.
  - Supongamos que la palabra seleccionada de manera aleatoria es "gato"
  - En la terminal se ejecuta un tablero / presentación de la palabra secreta por medio de guiones bajos y espacios. Por ejemplo para el caso de la palabra "gato", se vendría a presentar algo así: "_ _ _ _"
  - Luego de esto, el programa pedirá al usuario una letra (por medio de un input)
  - Si la letra proporcionada por el usuario existe dentro de la palabra secreta, este tablero / presentación se rellena y el programa vuelve a pedir al usuario otra letra. En nuestro ejemplo, suponiendo que se proporciona la letra "o", la consola de Python muestra lo siguiente:

```
Adivine una letra : o
Bravo! Letra Adivinada!
_ _ _ o
```

- Si por el contrario, el usuario proporciona una letra incorrecta, se muestra un mensaje indicando al usuario que ha perdido un turno. OJO: El usuario empieza con 5 turnos, y el juego termina cuando el usuario se queda sin turnos:

```
*******************************
BIENVENIDO AL JUEGO DEL AHORCADO
*******************************
```

```
Su palabra a adivinar tiene : 4 letras
- - - -

Que empiece el JUEGO!!
Adivine una letra : l

Intento Fallido! Usted Pierde un turno!
Le quedan : 4 turnos
- - - -

Adivine una letra : s

Intento Fallido! Usted Pierde un turno!
Le quedan : 3 turnos
- - - -

Adivine una letra : o
Bravo! Letra Adivinada!
_ _ _ o
```

0.1.8 *EJEMPLOS DE EJECUCIÓN*

- A continuación se muestra una ejecución del juego donde el usuario gana y adivina la palabra "gato":

```
# ================
# Ejecutar Juego
# TEST 1
# ================

jugar_ahorcado()

*******************************
BIENVENIDO AL JUEGO DEL AHORCADO
*******************************
Su palabra a adivinar tiene : 4 letras
- - - -

Que empiece el JUEGO!!
Adivine una letra : l

Intento Fallido! Usted Pierde un turno!
Le quedan : 4 turnos
- - - -

Adivine una letra : s

Intento Fallido! Usted Pierde un turno!
Le quedan : 3 turnos
```

```
_ _ _ _

Adivine una letra : o
Bravo! Letra Adivinada!
_ _ _ o

Adivine una letra : g
Bravo! Letra Adivinada!
g _ _ o

Adivine una letra : a
Bravo! Letra Adivinada!
g a _ o

Adivine una letra : t
Bravo! Letra Adivinada!
g a t o

Usted ha ganado el juego!!
```

- A continuación se muestra una ejecución del juego donde el usuario PIERDE por quedarse sin turnos con la palabra "sol"

```
# ================
# Ejecutar Juego
# TEST 2
# ================

jugar_ahorcado()

*******************************
BIENVENIDO AL JUEGO DEL AHORCADO
*******************************
Su palabra a adivinar tiene : 3 letras
_ _ _

Que empiece el JUEGO!!
Adivine una letra : l
Bravo! Letra Adivinada!
_ _ l

Adivine una letra : t

Intento Fallido! Usted Pierde un turno!
Le quedan : 4 turnos
_ _ l

Adivine una letra : h
```

```
Intento Fallido! Usted Pierde un turno!
Le quedan : 3 turnos
_ _ l

Adivine una letra : j

Intento Fallido! Usted Pierde un turno!
Le quedan : 2 turnos
_ _ l

Adivine una letra : k

Intento Fallido! Usted Pierde un turno!
Le quedan : 1 turnos
_ _ l

Adivine una letra : s
Bravo! Letra Adivinada!
s _ l

Adivine una letra : u

Intento Fallido! Usted Pierde un turno!
Le quedan : 0 turnos
s _ l

GAME OVER
```

- BUENA SUERTE !!!

[ ]:

EJERCICIO 17

Función en Python para crear un tablero de juego

TEMAS: Listas Multidimensionales / Funciones Normales / Bucle FOR / Bucles Anidados / Función Interna range

- En este ejercicio se pide crear dos funciones:
  - `crear_tablero_3_x_3()`
  - `crear_tablero_N_x_N(n)`

0.1.9 *A) `crear_tablero_3_x_3()`*

- Esta función debe ser capaz de hacer lo siguiente:

```
# ===============
# CÓDIGO TEST 1
# ===============
```

```
crear_tablero_3_x_3()
```

- Lo cual debe producir el siguiente resultado en nuestra terminal:

```
--------------------
|  O  |  O  |  O  |
--------------------
|  O  |  O  |  O  |
--------------------
|  O  |  O  |  O  |
--------------------
```

- Como se puede observar no acepta ningún parámetro
- Por defecto y como su nombre lo indica, crea un tablero de 3 x 3 que se llena con la letra "O" en mayúscula.

0.1.10 *B) `crear_tablero_N_x_N(n)`*

- A diferencia de la anterior función, esta recibe un parámetro n que representa el número de filas y columnas que va a tener nuestro tablero.
- La mejor manera de entender cómo opera esta función de Python es mediante la ejecución del siguiente código test:

```
# ===============
# CÓDIGO TEST 2
# ===============

crear_tablero_N_x_N(5)
print('\n\n')
crear_tablero_N_x_N(4)
print('\n\n')
crear_tablero_N_x_N(3)
print('\n\n')
crear_tablero_N_x_N(8)
```

- Lo cual genera el siguiente resultado en nuestra terminal:

```
-----------------------------
|  O  |  O  |  O  |  O  |  O  |
-----------------------------
|  O  |  O  |  O  |  O  |  O  |
-----------------------------
|  O  |  O  |  O  |  O  |  O  |
-----------------------------
|  O  |  O  |  O  |  O  |  O  |
-----------------------------
|  O  |  O  |  O  |  O  |  O  |
-----------------------------
```

```
------------------------
|  0  |  0  |  0  |  0  |
------------------------
|  0  |  0  |  0  |  0  |
------------------------
|  0  |  0  |  0  |  0  |
------------------------
|  0  |  0  |  0  |  0  |
------------------------
```

```
------------------
|  0  |  0  |  0  |
------------------
|  0  |  0  |  0  |
------------------
|  0  |  0  |  0  |
------------------
```

```
------------------------------------------------
|  0  |  0  |  0  |  0  |  0  |  0  |  0  |  0  |
------------------------------------------------
|  0  |  0  |  0  |  0  |  0  |  0  |  0  |  0  |
------------------------------------------------
|  0  |  0  |  0  |  0  |  0  |  0  |  0  |  0  |
------------------------------------------------
|  0  |  0  |  0  |  0  |  0  |  0  |  0  |  0  |
------------------------------------------------
|  0  |  0  |  0  |  0  |  0  |  0  |  0  |  0  |
------------------------------------------------
|  0  |  0  |  0  |  0  |  0  |  0  |  0  |  0  |
------------------------------------------------
|  0  |  0  |  0  |  0  |  0  |  0  |  0  |  0  |
------------------------------------------------
|  0  |  0  |  0  |  0  |  0  |  0  |  0  |  0  |
------------------------------------------------
```

- Como dato adicional, por favor notar que los marcos horizontales del tablero que se dibujan con guiones '—' también se generan de manera automática dependiendo del tamaño "n" del tablero. Una pequeña pista para lograr esto es primero realizar una prueba + error mediante la repetición de string multiplicado por un número entero.
- BUENA SUERTE !!!

[ ]:

EJERCICIO 18

Programa en Python para manipular un tablero de 3 x 3

TEMAS: Función Normal / str.format() / bucle WHILE / Scope Global / Scope Local / Keyword global / Listas Multidimensionales / Condicional IF / Indexing en Listas

- Para este ejercicio se pide por favor tomar en cuenta el ejercicio anterior que consistía en la programación de funciones en Python que nos permitan esquematizar un tablero de 3 x 3 y de n x n lleno de círculos (letra "O" mayúscula)
- Para este ejercicio se da como dato inicial una matriz o lista de listas que simula un tablero de 3 x 3.

```
# ====================================
# DATO INICIAL: Matriz tablero inicial
# ====================================

tablero = [
  ['O', 'O', 'O'],
  ['O', 'O', 'O'],
  ['O', 'O', 'O'],
]
```

- Adicionalmente, un punto crucial que debemos recordar / repasar para la correcta realización de este ejercicio es el tema de las variables globales y locales en Python.
- Para este repaso se ha desarrollado la siguiente celda script donde se retoman los fundamentos básicos de que implicaba el concepto de variables globales y locales en Python:

```
# ========================================
# REPASO SOBRE VARIABLES GLOBALES Y LOCALES
# ========================================

variable_1 = 10 # variable global
variable_2 = 50 # variable global

print('VARIABLES INICIALES:')
print( 'variable_1 =', variable_1 )
print( 'variable_2 =', variable_2 )

def funcion_cambiar_1():
  variable_1 = 100 # variable local diferente a variable_1 definida afuera
# end def

def funcion_cambiar_2():
  global variable_2
  variable_2 = 500 # variable global "variable_2" a la cual podemos acceder gracias a keyword
# end def
```

```
print('\nDESPUÉS DE DEFINIR FUNCIONES SIN EJECUTAR:')
print( 'variable_1 =', variable_1 )
print( 'variable_2 =', variable_2 )

funcion_cambiar_1()
funcion_cambiar_2()

print('\nDESPUÉS DE DEFINIR EJECUTAR FUNCIONES:')
print( 'variable_1 =', variable_1 )
print( 'variable_2 =', variable_2 )
```

- El resultado de ejecutar este script de repaso es el siguiente:

```
VARIABLES INICIALES:
variable_1 = 10
variable_2 = 50

DESPUÉS DE DEFINIR FUNCIONES SIN EJECUTAR:
variable_1 = 10
variable_2 = 50

DESPUÉS DE DEFINIR EJECUTAR FUNCIONES:
variable_1 = 10
variable_2 = 500
```

- Básicamente lo que tenemos que recordar de este repaso son los siguientes conceptos importantes:
  - `VARIABLE GLOBAL` : Aquella que se declara en el espectro global de nuestro código con indentación 0 fuera de un bloque de función.
  - `VARIABLE LOCAL` : Aquella que se declara dentro del bloque de la función y existe única y exclusivamente dentro de la función. Puede tener el mismo nombre que una variable ahí fuera, pero no tiene nada que ver con esta.
  - `KEYWORD "global"` : Permite el acceso a una variable global ahí fuera dentro del bloque de una función, e incluso podemos modificar esta variable dentro de la función.

0.1.11 *OBJETIVO PRINCIPAL DE ESTE EJERCICIO*

- Desarrollar el código de las siguientes 3 funciones en Python: ### - **`dibujar_tablero()`** ### - **`marcar_posicion(X, Y)`** ### - **`resetear_tablero()`**
- De tal manera que podamos ejecutar el siguiente código test:

```
# ====================
# CÓDIGO TEST FINAL
# ====================

tablero = [
  ['O', 'O', 'O'],
  ['O', 'O', 'O'],
  ['O', 'O', 'O'],
```

```
]

dibujar_tablero()
print('\n\n')
marcar_posicion(2,3)
print('\n\n')
marcar_posicion(3,1)
print('\n\n')
marcar_posicion(5,1)
print('\n\n')
marcar_posicion(1,5)
print('\n\n')
marcar_posicion(5,5)
print('\n\n')
marcar_posicion(1,1)
print('\n\n')
resetear_tablero()
print('\n\n')
marcar_posicion(1,1)
print('\n\n')
marcar_posicion(-1,1)
print('\n\n')
marcar_posicion(-1,-1)
print('\n\n')
marcar_posicion(2,2)
```

- Y podamos obtener el siguiente resultado en consola:

```
-------------------
|  0  |  0  |  0  |
-------------------
|  0  |  0  |  0  |
-------------------
|  0  |  0  |  0  |
-------------------




-------------------
|  0  |  0  |  0  |
-------------------
|  0  |  0  |  X  |
-------------------
|  0  |  0  |  0  |
-------------------



-------------------
```

```
|  O  |  O  |  O  |
-------------------
|  O  |  O  |  X  |
-------------------
|  X  |  O  |  O  |
-------------------
```

```
ERROR - Coordenadas (X,Y) tienen que estar entre 1 y 3
```

```
ERROR - Coordenadas (X,Y) tienen que estar entre 1 y 3
```

```
ERROR - Coordenadas (X,Y) tienen que estar entre 1 y 3
```

```
-------------------
|  X  |  O  |  O  |
-------------------
|  O  |  O  |  X  |
-------------------
|  X  |  O  |  O  |
-------------------
```

```
-------------------
|  O  |  O  |  O  |
-------------------
|  O  |  O  |  O  |
-------------------
|  O  |  O  |  O  |
-------------------
```

```
-------------------
|  X  |  O  |  O  |
-------------------
|  O  |  O  |  O  |
-------------------
|  O  |  O  |  O  |
-------------------
```

```
ERROR - Coordenadas (X,Y) tienen que estar entre 1 y 3


ERROR - Coordenadas (X,Y) tienen que estar entre 1 y 3


------------------
|  X  |  O  |  O  |
------------------
|  O  |  X  |  O  |
------------------
|  O  |  O  |  O  |
------------------
```

0.1.12 *NOTAS IMPORTANTES*

- El tablero a manejar va a ser de 3 x 3
- La función `marcar_posición(X,Y)` recibe las coordenadas del tablero donde `X` es la fila / `Y` es la columna
- POR FAVOR !! Tomar en cuenta que Python y las computadoras analizan los índices desde `(0,0)`, por lo tanto esto debe ser considerado al momento de escribir el código para la función `marcar_posición(X,Y)`
- Cada vez que se marca una posición la "O" cambia por "X"
- Si se proporciona una posición menor a 1 o mayor a 3 se debe mostrar el mensaje de error tal y como está arriba, ya sea que se intente esto en el parámetro X, Y, o en los 2.
- La función `resetear_tablero()` vuelve a dejar el tablero marcado en todas sus posiciones con la "O"
- Básicamente, estas 3 funciones manipulan la variable `tablero` que se da como dato inicial.
- Por tanto, aquí entra a teoría y recordatorio que se dio sobre variables globales y locales.
- BUENA SUERTE !!!

[ ]:
```python
# ====================================
# DATO INICIAL: Matriz tablero inicial
# ====================================

tablero = [
  ['O', 'O', 'O'],
  ['O', 'O', 'O'],
  ['O', 'O', 'O'],
]
```

[ ]:
```
# ========================================
# REPASO SOBRE VARIABLES GLOBALES Y LOCALES
# ========================================
```

[ ]:
```

```

EJERCICIO 19

Practicando las funciones con decoradores en Python

TEMAS: Funciones / Decoradores / Listas / Librería random

- Se da como datos iniciales del ejercicio el siguiente código de Python:

```
# ======================
# Datos e Importaciones
# ======================

import random

numeros = [10, 80, 50]
colores = ['amarillo', 'azul', 'rojo']


# ====================
# Funciones Iniciales
# ====================
def numero_al_azar():
  print( random.choice(numeros) )
# end def

def color_al_azar():
  print( random.choice(colores) )
# end def


# =======
# TEST
# =======
numero_al_azar()
print()
color_al_azar()
```

- Al ejecutar el programa, dentro de tantas soluciones posibles, una de ellas puede ser por ejemplo:

```
80

rojo
```

- El ejercicio pide lo siguiente:

  - SIN MODIFICAR las funciones originales (es decir, sin cambiar el código dentro de las funciones)
  - Generar un código en Python tal que, al momento de volver a ejecutar este test planteado, podamos tener algo así:

```
# ======
# TEST
# ======

numero_al_azar()
print()
color_al_azar()
```

```
Un número al azar es:
50

Un color al azar es:
amarillo
```

- Hay un concepto que nos permite hacer esto sin necesidad de cambiar la estructura de la función
- `TIP: decoradores`
- BUENA SUERTE !!!

[ ]:
```python
# ======================
# Datos e Importaciones
# ======================

import random

numeros = [10, 80, 50]
colores = ['amarillo', 'azul', 'rojo']


# ====================
# Funciones Iniciales
# ====================
def numero_al_azar():
  print( random.choice(numeros) )
# end def

def color_al_azar():
  print( random.choice(colores) )
# end def


# =======
# TEST
```

```
# =======
numero_al_azar()
print()
color_al_azar()
```

```
50

azul
```

[ ]:

EJERCICIO 20

Listado alfabético de estudiantes numeración en formato de diccionario

TEMAS: listas / diccionarios / función zip / Bucle FOR / enumerate

- Se da la siguiente lista de estudiantes:

```
lista_estudiantes = ['goku', 'naruto', 'akira', 'krilin', 'piccolo', 'seiya', 'zelda']
```

- El objetivo es presentar una nueva variable llamada `dict_estudiantes`, tal que:

```
{1: 'akira', 2: 'goku', 3: 'krilin', 4: 'naruto', 5: 'piccolo', 6: 'seiya', 7: 'zelda'}
```

- Esta variable `dict_estudiantes` es un diccionario, que presenta los estudiantes ordenados de manera alfabética, y donde las claves son números enteros que inician en 1.
- BUENA SUERTE !!!

[ ]:

[ ]:

EJERCICIO 21

Generación de ID (identificador único) de estudiantes y presentación en diccionario

TEMAS: listas / diccionarios / función zip / Bucle FOR / enumerate / indexing / slicing / métodos de str / map / list comprehension / función normal / función lambda

- Se da la siguiente lista de estudiantes:

```
lista_estudiantes = [
    'Carlos Andrade',
    'Sebastián Silva',
    'Ana García',
    'Pedro Samaniego',
    'Lucía Hernández',
    'Renato Santamaría',
    'Juan Izurieta'
]
```

- Con esta lista como dato se pide construir el siguiente diccionario, incorporando como clave un id de cada estudiante

```
{'CAR001AND': 'Carlos Andrade', 'SEB002SIL': 'Sebastián Silva', 'ANA003GAR': 'Ana García', 'PE
```

- Como se puede observar, cada clave se construye de la siguiente manera:
  - Nombre reducido a las 3 primeras letras y mayúsculas
  - Numeración con 2 ceros a la izquiera
  - Apellido reducido a las 3 primeras letras y mayúsculas

0.1.13 *NOTA IMPORTANTE:*

- Existen diferentes caminos y soluciones para este ejercicio, puede construirse esta nueva colección por ejemplo:
  - Utilizando recorrido for y creación de colecciones temporales
  - Utilizando la función map para crear una nueva lista a partir de la dada
  - Esta opción de función map puede utilizar función normal o lambda
  - Se puede hacer todo de manera más corta con list comprehension
  - ETC…
- El objetivo de este ejercicio es demostrar que existen no solo una, sino varias maneras de llegar a la solución de un problema de programación.
- Esta es una buena oportunidad para poner a prueba el conocimiento teórico del estudiante y poder ponerlo en práctica con ejemplos sencillos.
- BUENA SUERTE !!!

[ ]:

[ ]:

[ ]:

[ ]:

EJERCICIO 22

Cálculo de los 3 valores estadísticos más básicos e importantes: Media, Mediana y Moda

0.1.14 *(NOTA: Este ejercicio se compone de 3 partes)*

TEMAS: Funciones Normales / Casting / str.format() / Listas y Métodos / Función Map / Condicional IF / Librería math / Diccionarios / Bucle FOR / Indexing en Listas / Métodos Internos para Trabajar Números / Función LAMBDA

- Referencia Bibliográfica:
  - https://edu.gcfglobal.org/es/estadistica-basica/media-mediana-y-moda/1/
- Realizar un programa en Python para calcular la media, la mediana y la moda
- Cada programa constituye un subejercicio de este ejercicio 22 y se lo puede encontrar en la parte de abajo clasificado como:
  - 22.1) Función en Python para el Cálculo de la MEDIA Aritmética / Promedio
  - 22.2) Función en Python para el Cálculo de la MEDIANA Estadística

  - 22.3) Función en Python para el Cálculo de la MODA Estadística
- A continuación se explica de una manera muy básica y sencilla de entender, como se calculan estos 3 elementos estadísticos.

0.1.15 *Cálculo de la Media Aritmética / Promedio*

- Es el simple promedio aritmético
- Por ejemplo, si se tiene una lista:

```
lista = [1, 5, 3, 2]
```

- el promedio es igual a:

```
promedio = (1 + 5 + 3 + 2) / 4
promedio = 2.75
```

0.1.16 *Cálculo de la Mediana Estadística*

- La mediana no es más que el dato que se encuentra en la posición central de una lista de datos numéricos ordenados de menor a mayor.
- Esto va a ser distinto, si tenemos un número PAR de elementos y un número IMPAR.
- Ejemplo:

```
conjunto_datos_1 = [1, 5, 3, 2]
conjunto_datos_2 = [1, 5, 3, 2, 8]
```

- Ordenando los datos de menor a mayor:

```
datos_ordenados_1 = [1, 2, 3, 5]    # número PAR de datos
datos_ordenados_2 = [1, 2, 3, 5, 8] # número IMPAR de datos
```

- Cómo se puede ver para un número IMPAR de datos, se tiene una posición central, en este caso en el index = 2, por tanto:

```
datos_ordenados_2 = [1, 2, 3, 5, 8] # número IMPAR de datos
#                    0  1  2  3  4

posicion_central = 2
mediana = 3
```

- Para un número de datos IMPAR, se tienen 2 posiciones centrales y 2 valores de la mediana, y la mediana final sería el promedio de ambas

```
datos_ordenados_1 = [1, 2, 3, 5]    # número PAR de datos
#                    0  1  2  3

posicion_central_1 = 1
posicion_central_2 = 2

mediana_1 = 2
mediana_2 = 3

mediana = (2+3)/2 = 2.5
```

0.1.17 *Cálculo de la Moda Estadística*

- En palabras sencillas, la moda estadística es el dato que más se repite en un conjunto de datos.
- Por ejemplo, para el siguiente conjunto de datos:

```
conjunto_datos = [1, 5, 2, 9, 2, 5, 7, 8, 6, 5, 3]
#                    x           x           x

moda = 5
```

- El número 5 se repite 3 veces y el número 2 se repite 2 veces.
- El número 5 es el que más se repite, y este sería el valor de la moda.
- También puede darse, que tenemos un 2 o más datos que se repiten igual número de veces, por ejemplo:

```
conjunto_datos = [1, 5, 2, 9, 2, 5, 7, 8, 6, 5, 3, 2]
#                    x           x           x
#                       o     o                   o
```

- El 5 y el 2 se repiten 3 veces
- La moda vendría a ser el promedio de ambos

```
moda = (5 + 2)/2
moda = 3.5
```

0.1.18 *Planteamiento del Ejercicio:*

- En la parte 22.1 realizar una función en Python para calcular el promedio o media aritmética.
- En la parte 22.2 una función para la mediana.
- En la parte 22.3 una función para calcular la moda.
- Las 3 funciones reciben como argumento un conjunto de datos
- Por ejemplo:

A) Para el caso de la media

- Debemos crear la función `calcular_media(datos)`
- Que recibe como parámetro un conjunto de datos.

```
conjunto_datos_1 = [8, 14, 11, 12, 14, 11, 15, 11, 12, 8, 11, 14, 12, 9, 12]
conjunto_datos_2 = [8, 14, 11, 12, 14, 11, 15, 11, 12, 8, 11, 14, 12, 9, 12, 11]


# ========================
# TEST # 1
# Función: calcular_media
# ========================

calcular_media(conjunto_datos_1)
print('\n\n')
calcular_media(conjunto_datos_2)
```

- El resultado en consola sería (por ejemplo):

```
La media / promedio del siguiente conjunto de datos:
[8, 14, 11, 12, 14, 11, 15, 11, 12, 8, 11, 14, 12, 9, 12]
Es igual a 11.6



La media / promedio del siguiente conjunto de datos:
[8, 14, 11, 12, 14, 11, 15, 11, 12, 8, 11, 14, 12, 9, 12, 11]
Es igual a 11.56
```

B) Para el caso de la mediana

- Debemos crear la función `calcular_mediana(datos)`
- Igualmente recibe como parámetro un conjunto de datos.

```
conjunto_datos_1 = [8, 14, 11, 12, 14, 11, 15, 11, 12, 8, 11, 14, 12, 9, 12]
conjunto_datos_2 = [8, 14, 11, 12, 14, 11, 15, 11, 12, 8, 11, 14, 12, 9, 12, 11]



# ==========================
# TEST # 2
# Función: calcular_mediana
# ==========================

calcular_mediana(conjunto_datos_1)
print('\n\n')
calcular_mediana(conjunto_datos_2)
```

- El resultado en consola sería (por ejemplo):

```
Calculo / Análisis de la mediana estadística:
La lista tiene un número IMPAR de datos
La lista ordenada de MENOR a MAYOR valor es la siguiente:
[8, 8, 9, 11, 11, 11, 11, 12, 12, 12, 12, 14, 14, 14, 15]
pos_central_x = 7, Valor = 12
El valor de la mediana estadística es = 12.0



Calculo / Análisis de la mediana estadística:
La lista tiene un número PAR de datos
La lista ordenada de MENOR a MAYOR valor es la siguiente:
[8, 8, 9, 11, 11, 11, 11, 11, 12, 12, 12, 12, 14, 14, 14, 15]
pos_central_x = 7, Valor = 11 | pos_central_y = 8, Valor = 12
El valor de la mediana estadística es = 11.5
```

C) Para el caso de la moda

- De igual manera se debe crear una función `calcular_moda(datos)`
- Y de igual manera recibe como parámetro un conjunto de datos.

```
conjunto_datos_1 = [8, 14, 11, 12, 14, 11, 15, 11, 12, 8, 11, 14, 12, 9, 12]
conjunto_datos_2 = [8, 14, 11, 12, 14, 11, 15, 11, 12, 8, 11, 14, 12, 9, 12, 11]


# =========================
# TEST # 3
# Función: calcular_moda
# =========================

calcular_moda(conjunto_datos_1)
print('\n\n')
calcular_moda(conjunto_datos_2)
```

- El resultado en consola sería (por ejemplo):

```
Calculo de la moda estadística para el siguiente conjunto de datos:
[8, 14, 11, 12, 14, 11, 15, 11, 12, 8, 11, 14, 12, 9, 12]

(1) Datos y Número de Repeticiones
DATO ÚNICO: 8 | REPETICIONES: 2
DATO ÚNICO: 9 | REPETICIONES: 1
DATO ÚNICO: 11 | REPETICIONES: 4
DATO ÚNICO: 12 | REPETICIONES: 4
DATO ÚNICO: 14 | REPETICIONES: 3
DATO ÚNICO: 15 | REPETICIONES: 1

(2) Lista de Modas / en caso que haya más de una
[11, 12]

(3) Valor final de Moda Estadística:
11.5



Calculo de la moda estadística para el siguiente conjunto de datos:
[8, 14, 11, 12, 14, 11, 15, 11, 12, 8, 11, 14, 12, 9, 12, 11]

(1) Datos y Número de Repeticiones
DATO ÚNICO: 8 | REPETICIONES: 2
DATO ÚNICO: 9 | REPETICIONES: 1
DATO ÚNICO: 11 | REPETICIONES: 5
DATO ÚNICO: 12 | REPETICIONES: 4
DATO ÚNICO: 14 | REPETICIONES: 3
DATO ÚNICO: 15 | REPETICIONES: 1

(2) Lista de Modas / en caso que haya más de una
```

```
[11]

(3) Valor final de Moda Estadística:
11
```

0.1.19 *Notas Importantes / Recomendaciones*

- Este ejercicio es LARGO en comparación a los demás.
- Iniciar por lo más fácil: la media o promedio.
- Antes de definir la función se podría escribir el código aparte, comprobar su funcionamiento y luego plantear la función.
- Se aconseja tomar en cuenta los conjuntos de datos planteados en el ejemplo de arriba:

```
conjunto_datos_1 = [8, 14, 11, 12, 14, 11, 15, 11, 12, 8, 11, 14, 12, 9, 12]
conjunto_datos_2 = [8, 14, 11, 12, 14, 11, 15, 11, 12, 8, 11, 14, 12, 9, 12, 11]
```

- BUENA SUERTE !!!

[ ]:
```
# ============================
# Pleanteamiento del Ejemplo
# ============================

#                                              X                                  ␣
 ↪ # posición centro para número impar de datos
conjunto_datos_1 = [8, 14, 11, 12, 14, 11, 15, 11, 12, 8, 11, 14, 12, 9, 12]   ␣
 ↪ # número impar de datos
conjunto_datos_2 = [8, 14, 11, 12, 14, 11, 15, 11, 12, 8, 11, 14, 12, 9, 12,␣
 ↪11] # número par de datos
#                                              X   Y                              ␣
 ↪ # posiciones centrales para número par de datos

# INDEX             0  1   2   3   4   5   6   7   8   9  10  11  12  13 14  15␣
 ↪ (COMPUTADORAS)

# COUNTER           1  2   3   4   5   6   7   8   9   10 11  12  13  14 15  16␣
 ↪ (SERES HUMANOS)



# ---- Análisis para la MEDIANA ----
# Para número IMPAR de datos => EJ: 15 / posición central: index 7 / dato # 8
# Para número PAR de datos => EJ: 16 / posiciones centrales: index 7 y 8 /␣
 ↪datos # 8 y 9
```

0.2 *(22.1) Función en Python para el Cálculo de la MEDIA Aritmética / Promedio*

```
[ ]: # ================================
     # Función para calcular la MEDIA
     # ================================
```

0.3 *(22.2) Función en Python para el Cálculo de la MEDIANA Estadística*

```
[ ]: # ================================
     # Función para calcular la MEDIANA
     # ================================
```

0.4 *(22.3) Función en Python para el Cálculo de la MODA Estadística*

```
[ ]: # ================================
     # Función para calcular la MODA
     # ================================
```

EJERCICIO 23

Programa para obtener el nombre de una variable en Python

TEMAS: Funciones Normales / Librería Externa "varname" / Instalación pip dentro de Google Colab / nameof() & argname() de "varname" / Métodos de string / str.split() & str.join() / Keywords: from & import / str.format()

- El objetivo de este ejercicio es contestar una pregunta muy repetida en el ámbito de la programación: ¿Cómo puedo obtener el nombre de una variable declarada?
- Aunque parece algo sencillo, bueno esta pregunta tiene muchas maneras de ser contestada.
- Y la respuesta depende también del lenguaje de programación en sí.
- En el caso de Python una solución muy factible es por medio de una librería externa llamada `varname`
- Por tanto este ejercicio tiene más de una finalidad (que al inicio sonaba muy sencillo: "obtener el nombre de una variable"):
  - Vamos a aprender a INSTALAR una librería externa dentro de nuestro espacio de Google Colab.
  - Vamos a repasar un poco sobre importación de módulos y elementos de módulo (keywords: from & import)
  - Vamos a jugar un poco con este módulo y generar 2 funciones en python:
    *

    *

0.4.1 *Instalación de un módulo / librería externa en Google Colab*

- Para las personas que conocen un poco más del tema de Python, deben saber que la instalación de paquetes y módulos externos se hace generalmente en nuestra terminal (Command Window CMD o Power Shell de Windows) mediante el comando **`pip`**
- `pip` => es el gestor de paquetes y módulos de python
- Como bien se indica, pues esto funciona en la terminal, pero ¿cómo lo hacemos en Colab?
- De una manera muy sencilla en Colab, si vamos a usar un módulo / librería que Colab no lo tiene tenemos que ejecutar el comando pip de instalación pero ANTECEDIDO por un signo de exclamación.
- Para nuestro caso vamos a usar el módulo o librería externa `varname`
- https://pypi.org/project/varname/

- Si leemos la documentación, pues como nos indica ahí tenemos que ejecutar el siguiente comando pip para instalarlo.
- Y tomando en cuenta lo que se dijo anteriormente, debemos en una celda de código de Colab anteceder el símbolo `!`.
- Entonces la primera celda a ejecutar sería la siguiente:

```
# ============================================================
# LÍNEA DE CÓDIGO PARA INSTALAR EL PAQUETE EXTERNO "varname"
# ============================================================
# https://pypi.org/project/varname/

!pip install -U varname
```

- Lo cual al ejecutarla `(SHIFT + ENTER)`, nos va a presentar el siguiente resultado:

```
Collecting varname
  Downloading varname-0.12.2-py3-none-any.whl (24 kB)
Collecting executing<3.0,>=2.0 (from varname)
  Downloading executing-2.0.1-py2.py3-none-any.whl (24 kB)
Installing collected packages: executing, varname
Successfully installed executing-2.0.1 varname-0.12.2
```

- Una vez hecho esto, podemos acceder a los métodos y elementos que tengamos dentro de esta librería (tal y como lo hacemos con math por ejemplo)

- En el siguiente código se muestra el uso de 2 funciones de esta librería que vamos a usar para nuestro ejercicio:
  - ***nameof()***
  - ***argname()***

```
# ====================================
# EJEMPLO DE USO DE PAQUETE "varname"
# ====================================

# importación de varname y sus funciones relevantes
from varname import nameof, argname

mascota_favorita = 'gatito siamés'

# ----------------------------------------------------------
# Obtener el nombre de una variable declarada en el global
# ----------------------------------------------------------

print( nameof(mascota_favorita) )


# ------------------------------------------------
# Obtener el nombre del argumento en una función
# ------------------------------------------------

def obtener_nombre_variable( variable ):
  print( argname( 'variable' ) )
# end def

obtener_nombre_variable( mascota_favorita )
```

- El resultado de ejecutar esta celda de código va a ser el siguiente:

```
mascota_favorita
mascota_favorita
```

- Espero de corazón que esté clara la diferencia de ambas funciones con solo ver el código, pero bueno, en caso que no la explicación es muy sencilla:
  - nameof() : me devuelve el nombre de una variable que declaro en el global
  - argname() : me devuelve el nombre de un argumento que pasamos a una función que declaramos.

0.4.2 *Objetivo del Ejercicio*

- Esperando que estén SÚPER CLARAS las siguientes explicaciones:
  - Como instalar un módulo / paquete externo en Colab.
  - Como usar las funciones `nameof()` & `argname()`
- Nuestro objetivo es el siguiente:
  - Dadas las siguientes variables como dato inicial:

```
# ================
# DATOS INICIALES
# ================

color_favorito = 'Azul'
edad = 25
carrera_de_estudios = 'Ingeniería Mecánica'
platillo_favorito = 'Bolón de Verde con Chicharrón'
```

- Generar el código de 2 funciones en Python:
  - ***get_nombre_variable(variable)***
  - ***generar_frase(variable)***
- De tal manera que podamos satisfacer el siguiente CÓDIGO TEST:

```
# ================
# CÓDIGO DE TEST
# ================

# Test de la función:  get_nombre_variable
print( get_nombre_variable(color_favorito) )
print( get_nombre_variable(edad) )
print( get_nombre_variable(carrera_de_estudios) )
print( get_nombre_variable(platillo_favorito) )

# Test de la función: generar_frase
print('\n\n')

generar_frase( color_favorito )
generar_frase( edad )
generar_frase( carrera_de_estudios )
generar_frase( platillo_favorito )
```

- El cual nos debe dar el siguiente resultado (tomando en cuenta las variables dadas como dato inicial):

```
Color Favorito
Edad
Carrera De Estudios
Platillo Favorito
```

```
Su Color Favorito es AZUL
Su Edad es 25
Su Carrera De Estudios es INGENIERÍA MECÁNICA
Su Platillo Favorito es BOLÓN DE VERDE CON CHICHARRÓN
```

- BUENA SUERTE !!!

[ ]:
```
# ============================================================
# LÍNEA DE CÓDIGO PARA INSTALAR EL PAQUETE EXTERNO "varname"
# ============================================================
# https://pypi.org/project/varname/

# !pip install -U varname
```

[ ]:
```
# ====================================
# EJEMPLO DE USO DE PAQUETE "varname"
# ====================================

# importación de varname y sus funciones relevantes

# ----------------------------------------------------------
# Obtener el nombre de una variable declarada en el global
# ----------------------------------------------------------


# ------------------------------------------------
# Obtener el nombre del argumento en una función
# ------------------------------------------------
```

[ ]:
```
# ================
# DATOS INICIALES
# ================

color_favorito = 'Azul'
```

```
edad = 25
carrera_de_estudios = 'Ingeniería Mecánica'
platillo_favorito = 'Bolón de Verde con Chicharrón'
```

[ ]:

EJERCICIO 24

varname con funciones *args | Presentando una Lista de Productos

TEMAS: Funciones Normales / Funciones args / Librería Externa "varname" / Instalación pip dentro de Google Colab / nameof() & argname() de "varname" / Métodos de string / str.split() & str.join() / Keywords: from & import / str.format() / Listas / Listas de Listas / Métodos de Listas / .append() / Bucle FOR / FOR Anidado / enumerate / str.format() para formato de texto

- Este ejercicio es un complemento del anterior donde aprendimos a capturar y retornar el nombre de una variable en Python.
- Básicamente, vamos a utilizar esa misma teoría más todo lo que ya conocemos sobre listas, bucles y bucles anidados con el fin de presentar una lista completa de productos en una tienda.

0.4.3 *TEORÍA DE REPASO: Funciones* args*

- Recordar que una función `*args` es aquella que:
  - Recibe un número indefinido de parámetros.
  - Y los devuelve como tupla
  - El nombre "args" no es importante, pero si lo es el `*`
- Un pequeño resumen de este funcionamiento lo podemos ver en el siguiente código:

```
# ================================================
# UN POCO DE TEORÍA - PARTE 1
# Recordando la función con *args
# ================================================

# ==> un ejemplo básico de función *args
def sumar(*args):
  resultado = 0

  for numero in args:
    resultado += numero
  # end for

  return resultado
# end def

# TEST
print( sumar(1,2,3,4,5) ) # 15
print( sumar(10,20) ) # 30
```

```
# ==> RECORDAR: ¿qué devuelve *args?
# (el nombre args no es importante)

def que_devuelve( *argumentos ):
  print( argumentos , type(argumentos) )
# end def

# TEST
que_devuelve('hola', 10, True) # ('hola', 10, True) <class 'tuple'>
```

- En el anterior código ya se pone en forma de comentario el resultado de cada test, sin embargo, una vez más se presenta aquí cuál sería el resultado al ejecutar el anterior código:

```
15
30
('hola', 10, True) <class 'tuple'>
```

- De esta manera, espero de corazón haya quedado clara la funcionalidad de las funciones args:
  - Reciben un número indeterminado de parámetros, puede ser 1 puede ser N.
  - El nombre *args no es relevante pero si el uso del asterísco
  - Básicamente todos estos argumentos se almacenan a manera de tupla y podemos recorrerlos

0.4.4 *TEORÍA DE REPASO: Funciones* args + varname, argname*

- Habiendo recordado como operan las funciones `*args`.
- Estamos en la capacidad de unir este concepto con lo revisado en el anterior ejercicio con la librería externa `varname`
- Básicamente lo que deseamos es unir estos dos conceptos para obtener el nombre de las variables que pasen como argumentos en una función `*args`
- Para eso se ha preparado el siguiente código explicativo:

```
# ================================================
# UN POCO DE TEORÍA - PARTE 2
# Utilizando el módulo varname con función *args
# ================================================

# importación de varname y sus funciones relevantes
from varname import nameof, argname


# función args + varname / argname
"""
def get_nombres_variables( *args ):
  tupla_nombres_variables = argname('args')
  return tupla_nombres_variables
# end def
"""
```

```
def get_nombres_variables( *args ):
  tupla_nombres_variables = argname( nameof(args) )
  return tupla_nombres_variables
# end def


# TEST: variables para prueba
saludo = 'hola'
despedida = 'chao'
nombre = 'sebas'


# TEST: devolviendo los nombres de las variables en función *args
print( get_nombres_variables( saludo , despedida ) )
print( get_nombres_variables( nombre , saludo, despedida ) )
```

- Ejecutando este código tendríamos el siguiente resultado:

```
('saludo', 'despedida')
('nombre', 'saludo', 'despedida')
```

- Básicamente como podemos observar, en esta función `get_nombres_variables()` le estamos pasando una serie de variables que hemos creado en el espectro global: `saludo`, `despedida` y `nombre`.
- Lo que deseamos es justamente esos nombres de las variables y lo hacemos de una manera sencilla por medio de una función `*args` que devuelve estos nombres a manera de tupla.

0.4.5 *Objetivo del Ejercicio*

- En primer lugar se da un bloque de código en Python con los siguientes datos iniciales para nuestro ejercicio:

```
# ======================================
# DATOS INICIALES DEL PROBLEMA
# lista de tiendas => productos & precio
# ======================================

tienda_de_ropa = [
    ['Camisa' , 15.55],
    ['Pantalón' , 24.99],
    ['Corbata' , 12.45],
    ['Zapatos de Terno' , 65.45],
]


tienda_de_alimentos = [
    ['Cola', 0.75],
    ['Papas Fritas', 1.25],
    ['Empanada de Queso', 3.75],
```

```
    ['Capuccino', 2.25],
]


tienda_de_musica = [
    ['Piano', 158.99],
    ['Guitarra', 78.12],
    ['Flauta', 11.99],
    ['Harmónica', 19.99],
    ['Batería', 250.78],
]
```

- Teniendo en cuenta la teoría de las funciones `args*` y la teoría de como combinar estas funciones con varname y argname se pide generar el código python de las siguientes 2 funciones:
  –

  –

- De tal manera que se pueda satisfacer el siguiente código test, teniendo en consideración las variables presentadas como datos iniciales:

```
# ====================
# CÓDIGO TEST FINAL
# ====================

# Generar 2 listas finales distintas
lista_final_1 = generar_lista_total_de_productos( tienda_de_ropa , tienda_de_alimentos , tienda
lista_final_2 = generar_lista_total_de_productos( tienda_de_alimentos , tienda_de_musica , tien

# TEST: Resultado de lista_final_1
print(lista_final_1)
print(lista_final_2)

# Presentar lista 1
print('\n\nLISTA FINAL 1\n')
presentar_lista_productos( lista_final_1 )

# Presentar lista 2
print('\n\nLISTA FINAL 2\n')
presentar_lista_productos( lista_final_2 )
```

- Este código deberá presentar el siguiente resultado en la consola:

```
[['TDR-1', 'Tienda De Ropa', 'Camisa', 15.55], ['TDR-2', 'Tienda De Ropa', 'Pantalón', 24.99],
[['TDA-1', 'Tienda De Alimentos', 'Cola', 0.75], ['TDA-2', 'Tienda De Alimentos', 'Papas Frita
```

```
LISTA FINAL 1

N  |  ID   |   NOMBRE DE TIENDA   |      PRODUCTO      |   PRECIO
1  | TDR-1 | Tienda De Ropa       | Camisa             | 15.55  USD
2  | TDR-2 | Tienda De Ropa       | Pantalón           | 24.99  USD
3  | TDR-3 | Tienda De Ropa       | Corbata            | 12.45  USD
4  | TDR-4 | Tienda De Ropa       | Zapatos de Terno   | 65.45  USD
5  | TDA-1 | Tienda De Alimentos  | Cola               | 0.75   USD
6  | TDA-2 | Tienda De Alimentos  | Papas Fritas       | 1.25   USD
7  | TDA-3 | Tienda De Alimentos  | Empanada de Queso  | 3.75   USD
8  | TDA-4 | Tienda De Alimentos  | Capuccino          | 2.25   USD
9  | TDM-1 | Tienda De Musica     | Piano              | 158.99 USD
10 | TDM-2 | Tienda De Musica     | Guitarra           | 78.12  USD
11 | TDM-3 | Tienda De Musica     | Flauta             | 11.99  USD
12 | TDM-4 | Tienda De Musica     | Harmónica          | 19.99  USD
13 | TDM-5 | Tienda De Musica     | Batería            | 250.78 USD


LISTA FINAL 2

N  |  ID   |   NOMBRE DE TIENDA   |      PRODUCTO      |   PRECIO
1  | TDA-1 | Tienda De Alimentos  | Cola               | 0.75   USD
2  | TDA-2 | Tienda De Alimentos  | Papas Fritas       | 1.25   USD
3  | TDA-3 | Tienda De Alimentos  | Empanada de Queso  | 3.75   USD
4  | TDA-4 | Tienda De Alimentos  | Capuccino          | 2.25   USD
5  | TDM-1 | Tienda De Musica     | Piano              | 158.99 USD
6  | TDM-2 | Tienda De Musica     | Guitarra           | 78.12  USD
7  | TDM-3 | Tienda De Musica     | Flauta             | 11.99  USD
8  | TDM-4 | Tienda De Musica     | Harmónica          | 19.99  USD
9  | TDM-5 | Tienda De Musica     | Batería            | 250.78 USD
10 | TDR-1 | Tienda De Ropa       | Camisa             | 15.55  USD
11 | TDR-2 | Tienda De Ropa       | Pantalón           | 24.99  USD
12 | TDR-3 | Tienda De Ropa       | Corbata            | 12.45  USD
13 | TDR-4 | Tienda De Ropa       | Zapatos de Terno   | 65.45  USD
```

0.4.6 *Función: generar_lista_total_de_productos*

- Esta función recibe como argumentos las variables dadas como dato, en cualquier orden.
- Como se puede observar de la ejecución del código test, nos retorna una lista de listas, donde cada lista representa un producto de la tienda.
- Por favor observar los siguientes detalles:

  - Cada "sublista" contiene 4 elementos.
  - El primer subelemento es un ID del producto que se forma de las primeras 3 letras del

nombre de la tienda. Por ejemplo: Tienda De Ropa = TDR. Y adicionalmente tiene un guión y un número que representa el número del producto dentro de esa tienda.
  - El segundo subelemento es el nombre de la tienda que proviene del nombre de la variable dato. Por ejemplo: tienda_de_ropa = Tienda De Ropa. Esto es algo que como sabemos podemos hacer con el módulo externo "varname".
  - El tercer y cuarto subelemento es la información que tenemos de cada producto en las variables datos: el producto y su precio.
- Básicamente, esta función "generar_lista_total_de_productos" combina las variables representativas de las tiendas y genera una lista de todos los productos de las tiendas, eso es lo que hace.

0.4.7 *Función: `presentar_lista_productos()`*

- Ahora esta función `presentar_lista_productos()` lo que hace no es más que recibir como argumento la lista generada por la función `generar_lista_total_de_productos()` y presentar los datos en pantalla tal y como se observa.
- Adicionalmente a los elementos que se tiene en la lista resultante, esta función genera una numeración para todos los productos que se tiene empezando con el 1.
- También genera como primera fila un encabezado:
  - `N | ID | NOMBRE DE TIENDA | PRODUCTO | PRECIO`
- Y después de este encabezado se presentan los productos.
- Básicamente esta función RECORRE la lista resultante e IMPRIME los datos tal y como se muestra.
- Lo importante aquí, y nótese por favor es la manera como se presentan los datos, por ejemplo:
  - Los elementos del encabezado están centrados
  - El espaciado parece er perfecto
- Si bien es cierto hay varias maneras como se puede hacer esto con prueba y error, nosotros habíamos aprendido también que con `string.format()` podemos brindar un formato especial a lo que es texto (por favor RECORDAR esto)

- BUENA SUERTE !!!

[ ]:
```
# ============================================================
# LÍNEA DE CÓDIGO PARA INSTALAR EL PAQUETE EXTERNO "varname"

# NOTA:
# - A veces al cerrar Colab
# - Tenemos que volver a instalar el paquete externo
# ============================================================
# https://pypi.org/project/varname/

# !pip install -U varname
```

[ ]:
```
# ================================================
# UN POCO DE TEORÍA - PARTE 1
# Recordando la función con *args
# ================================================

# ==> un ejemplo básico de función *args



# ==> RECORDAR: ¿qué devuelve *args?
# (el nombre args no es importante)
```

[ ]:
```
# ================================================
# UN POCO DE TEORÍA - PARTE 2
# Utilizando el módulo varname con función *args
# ================================================

# importación de varname y sus funciones relevantes



# función args + varname / argname



# TEST: variables para prueba



# TEST: devolviendo los nombres de las variables en función *args
```

[ ]:
```
# ======================================
# DATOS INICIALES DEL PROBLEMA
# lista de tiendas => productos & precio
# ======================================

tienda_de_ropa = [
    ['Camisa' , 15.55],
    ['Pantalón' , 24.99],
    ['Corbata' , 12.45],
    ['Zapatos de Terno' , 65.45],
]


tienda_de_alimentos = [
    ['Cola', 0.75],
    ['Papas Fritas', 1.25],
```

```
    ['Empanada de Queso', 3.75],
    ['Capuccino', 2.25],
]


tienda_de_musica = [
    ['Piano', 158.99],
    ['Guitarra', 78.12],
    ['Flauta', 11.99],
    ['Harmónica', 19.99],
    ['Batería', 250.78],
]
```

[ ]:

EJERCICIO 25

Programa de Facturación de una Tienda de Productos Online

TEMAS: Funciones Normales / Casting entre Colecciones / str.format() / Librería Externa varname / Diccionarios / Listas / str.format() - formato de números / str.format() - formato de texto / Condicional IF / IF Anidado / Bucle FOR / FOR Anidado / map / reduce / Indexing de Listas / Métodos de Listas / Recorrido de Colecciones con Bucle / Bucle WHILE True / Sentencias de Control / break

- ATENCIÓN: Para este ejercicio vamos a tomar en cuenta la funcionalidad de la librería `varname` descrita en los ejercicios revisados anteriormente
- El objetivo de este ejercicio es crear un programa en Python que simule una tienda online de productos.
- Por favor tomar en cuenta si bien este ejercicio NO ES COMPLEJO, pero pone en práctica muchos conceptos revisados tanto en el Crash Course como a lo largo de esta ronda de ejercicios.
- Por lo tanto, la complejidad de este ejercicio radica más en lo largo del código que se debe escribir para poder realizarlo.
- De manera auxiliar por parte del instructor, los siguientes bloques / celdas de código son proporcionadas al estudiante, de manera que se pueda usar esto de manera inicial:

0.4.8 *Código 1 proporcionado*

- El primer bloque corresponde a la línea que se debe ejecutar para instalar la librería externa `varname`

```
# *** CÓDIGO # 1 PROPORCIONADO PARA EL EJERCICIO ***

# ========================================================
# LÍNEA DE CÓDIGO PARA INSTALAR EL PAQUETE EXTERNO "varname"
# NOTA:
# A veces al cerrar Colab
```

```
# Tenemos que volver a instalar el paquete externo
# ============================================================
# https://pypi.org/project/varname/

!pip install -U varname
```

0.4.9 *Código 2 proporcionado*

- Se presentan los datos del ejercicio que básicamente son 3 diccionarios correspondientes a una categoría de productos y los productos de la tienda online. La clave de cada diccionario es el código único del producto y el valor de la clave es una tupla que contiene el nombre del producto y el precio.

```
# *** CÓDIGO # 2 PROPORCIONADO PARA EL EJERCICIO ***

# ================================
# DATOS INICIALES DEL EJERCICIO
# ================================

IVA = 0.15

articulos_de_computacion = {
    '00A-1' : ('Laptop I7' , 1250.99),
    '00A-2' : ('Teclado Gamer' , 55.65),
    '00A-3' : ('Mouse Gamer' , 24.99),
    '00A-4' : ('Web Cam 4K' , 88.38),
    '00A-5' : ('Monitor 32 pulgadas' , 228.75),
}

articulos_gaming = {
    '00B-1' : ('Play Station 5' , 550.59),
    '00B-2' : ('Nintendo Switch' , 335.78),
    '00B-3' : ('Steam Deck' , 845.99),
    '00B-4' : ('Control XBox' , 56.88),
    '00B-5' : ('RetroPie 400GB' , 158.99),
}

articulos_telefonos_tablets = {
    '00C-1' : ('Samsung Galaxy S23' , 1200.99),
    '00C-2' : ('Iphone 15' , 1325.88),
    '00C-3' : ('Microsoft Surface 500GB' , 775.66),
    '00C-4' : ('IPad Air 500GB' , 980.45),
    '00C-5' : ('Huawei XPad 128GB' , 335.28),
}
```

0.4.10 *Código 3 proporcionado*

- Se proporcionan 2 funciones auxiliares para recorrer e imprimir los elementos tanto de una lista como de un diccionario en la consola o terminal de Python

```
  # *** CÓDIGO # 3 PROPORCIONADO PARA EL EJERCICIO ***

# ================================
# FUNCIONES AUXILIARES PARA TEST
# ================================


def presentar_lista(lista):
  for index, elemento in enumerate(lista):
    print( '{} | {}'.format( index+1 , elemento ) )
  # end for
# end def



def presentar_diccionario(diccionario):
  for clave, valor in diccionario.items():
    print( '{} | {}'.format( clave , valor ) )
  # end for
# end def
```

0.4.11 *Objetivo del Ejercicio*

- Con estos elementos dados, se pide generar un programa al estilo de "while True"
- Para simplificar las cosas, se presenta una ejecución de ejemplo en la terminal de Python de cómo debería de funcionar este programa:

```
-----------------------------------------------
Bienvenidos a la Tienda Online de @sebassilvap
La tienda más increíble del mundo entero :D
-----------------------------------------------
(1) Articulos De Computacion
CÓDIGO: 00A-1 |        Laptop I7        | PRECIO: 1250.99 USD
CÓDIGO: 00A-2 |      Teclado Gamer      | PRECIO: 55.65   USD
CÓDIGO: 00A-3 |       Mouse Gamer       | PRECIO: 24.99   USD
CÓDIGO: 00A-4 |       Web Cam 4K        | PRECIO: 88.38   USD
CÓDIGO: 00A-5 |   Monitor 32 pulgadas   | PRECIO: 228.75  USD
(2) Articulos Gaming
CÓDIGO: 00B-1 |     Play Station 5      | PRECIO: 550.59  USD
CÓDIGO: 00B-2 |     Nintendo Switch     | PRECIO: 335.78  USD
CÓDIGO: 00B-3 |       Steam Deck        | PRECIO: 845.99  USD
CÓDIGO: 00B-4 |      Control XBox       | PRECIO: 56.88   USD
CÓDIGO: 00B-5 |     RetroPie 400GB      | PRECIO: 158.99  USD
(3) Articulos Telefonos Tablets
CÓDIGO: 00C-1 |   Samsung Galaxy S23    | PRECIO: 1200.99 USD
CÓDIGO: 00C-2 |        Iphone 15        | PRECIO: 1325.88 USD
CÓDIGO: 00C-3 | Microsoft Surface 500GB | PRECIO: 775.66  USD
CÓDIGO: 00C-4 |     IPad Air 500GB      | PRECIO: 980.45  USD
```

```
CÓDIGO: 00C-5 |    Huawei XPad 128GB    | PRECIO: 335.28  USD
-------------------------------------------------------------

OPCIONES DEL USUARIO:
[1] - Agregar Producto a Carrito de Compras
[2] - Ver Carrito de Compras
[3] - Generar Factura
[4] - Salir del Programa (s/q)

ELIJA SU OPCIÓN : 2

SU CARRITO DE COMPRAS:
EL CARRITO DE COMPRAS ESTÁ VACÍO !!!
OPCIONES DEL USUARIO:
[1] - Agregar Producto a Carrito de Compras
[2] - Ver Carrito de Compras
[3] - Generar Factura
[4] - Salir del Programa (s/q)

ELIJA SU OPCIÓN : 3

SU FACTURA HASTA EL MOMENTO:
EL CARRITO DE COMPRAS ESTÁ VACÍO !!!
OPCIONES DEL USUARIO:
[1] - Agregar Producto a Carrito de Compras
[2] - Ver Carrito de Compras
[3] - Generar Factura
[4] - Salir del Programa (s/q)

ELIJA SU OPCIÓN : 1
INGRESE EL CÓDIGO DEL PRODUCTO A AGREGAR : 00c-3
PRODUCTO AGREGADO AL CARRITO !!!
OPCIONES DEL USUARIO:
[1] - Agregar Producto a Carrito de Compras
[2] - Ver Carrito de Compras
[3] - Generar Factura
[4] - Salir del Programa (s/q)

ELIJA SU OPCIÓN : 1
INGRESE EL CÓDIGO DEL PRODUCTO A AGREGAR :      00c-3
PRODUCTO AGREGADO AL CARRITO !!!
OPCIONES DEL USUARIO:
[1] - Agregar Producto a Carrito de Compras
[2] - Ver Carrito de Compras
[3] - Generar Factura
[4] - Salir del Programa (s/q)

ELIJA SU OPCIÓN : 00b-1
```

```
ERROR - OPCIÓN DE MENÚ INCORRECTA...
OPCIONES DEL USUARIO:
[1] - Agregar Producto a Carrito de Compras
[2] - Ver Carrito de Compras
[3] - Generar Factura
[4] - Salir del Programa (s/q)

ELIJA SU OPCIÓN : 1
INGRESE EL CÓDIGO DEL PRODUCTO A AGREGAR : 00b-1
PRODUCTO AGREGADO AL CARRITO !!!
OPCIONES DEL USUARIO:
[1] - Agregar Producto a Carrito de Compras
[2] - Ver Carrito de Compras
[3] - Generar Factura
[4] - Salir del Programa (s/q)

ELIJA SU OPCIÓN : 1
INGRESE EL CÓDIGO DEL PRODUCTO A AGREGAR : 00a-4
PRODUCTO AGREGADO AL CARRITO !!!
OPCIONES DEL USUARIO:
[1] - Agregar Producto a Carrito de Compras
[2] - Ver Carrito de Compras
[3] - Generar Factura
[4] - Salir del Programa (s/q)

ELIJA SU OPCIÓN : 5555
ERROR - OPCIÓN DE MENÚ INCORRECTA...
OPCIONES DEL USUARIO:
[1] - Agregar Producto a Carrito de Compras
[2] - Ver Carrito de Compras
[3] - Generar Factura
[4] - Salir del Programa (s/q)

ELIJA SU OPCIÓN : 2

SU CARRITO DE COMPRAS:
N  | Nombre de Producto      | Cantidad | Precio Unitario
1  | Microsoft Surface 500GB | 2        | 775.66
2  | Play Station 5          | 1        | 550.59
3  | Web Cam 4K              | 1        | 88.38
OPCIONES DEL USUARIO:
[1] - Agregar Producto a Carrito de Compras
[2] - Ver Carrito de Compras
[3] - Generar Factura
[4] - Salir del Programa (s/q)

ELIJA SU OPCIÓN : 3
```

```
SU FACTURA HASTA EL MOMENTO:
N  | Nombre de Producto      | Cantidad | Precio Unitario | Subtotal Unitario
1  | Microsoft Surface 500GB | 2        | 775.66          | 1551.32
2  | Play Station 5          | 1        | 550.59          | 550.59
3  | Web Cam 4K              | 1        | 88.38           | 88.38
                                                SUBTOTAL | 2190.29 USD
                                                     IVA | 15.0 %
                                                   TOTAL | 2518.83 USD
OPCIONES DEL USUARIO:
[1] - Agregar Producto a Carrito de Compras
[2] - Ver Carrito de Compras
[3] - Generar Factura
[4] - Salir del Programa (s/q)

ELIJA SU OPCIÓN : 4

GRACIAS POR VISITAR NUESTRA TIENDA! :D
```

0.4.12 *Observaciones Importantes*

- En este punto sería más que suficiente el presentar este ejemplo de ejecución del programa, sin embargo vamos a aclarar algunos puntos.
- Como se puede observar al ejecutar el programa se presentan todos los productos que se plantean como dato.
  - Pero esta presentación no es así de simple.
  - Observar que se presenta el nombre de la categoría de los productos (igual a los nombres de las variables de los diccionarios de datos)
  - Se numeran estas categorías
  - Y se presentan los productos
  - El centrado y el espaciado de como se presentan los datos NO ES COINCIDENCIA (Recordar lo que hacíamos con `str.format()`)
- Tomar en cuenta que esta presentación de los productos ocurre 1 vez.
- Luego de esto se muestran las opciones del programa:
  - [1] - Agregar Producto a Carrito de Compras
  - [2] - Ver Carrito de Compras
  - [3] - Generar Factura
  - [4] - Salir del Programa (s/q)
- Al inicio, cuando no tenemos nada en el carrito, como podemos observar al utilizar tanto la opción 2 como la 3, nos va a salir un mensaje indicando que el carro esta vacío.
- La opción 1 nos permite agregar elementos a nuestro carro de compras (una variable que se crea al momento que empezamos a agregar algo)
- Y pues la lógica es de esta manera:
  - Si el producto no existe, se agrega al carrito

  - Si el producto ya está en el carrito, la cantidad sube 1

- Esta variable del carrito de compras se mantiene (PERSISTE) mientras el bucle while True del programa esté vigente.
- Cuando tenemos algo en el carrito de compras podemos usar:
  - Opción 2: Para ver el carrito de compras (tal y como se indica en la ejecución de prueba arriba)
  - Opción 3: Generamos la factura, subtotales y total final considerando el IVA (que se da como dato)
- No olvidar una vez más: LOS CENTRADOS Y ESPACIADOS de como se presenta el texto en la consola, no es coincidencia (**Recordar `str.format()`**)
- Podemos salir del programa con el `4, s, q, S, Q`

0.4.13 *Recomendaciones*

- Tratar de usar varias celdas de código.
- Tratar de generar varias funciones auxiliares que se puedan usar luego dentro del **`while True:`**
- De esta manera el código final para la ejecución del programa no va a ser muy extenso.
- Es decir tratar de dividir el código en varias partes.
- No hacerlo complicado, no es necesario, recordar lo que se dijo en un principio: EL EJERCICIO NO ES COMPLICADO, PERO ES LARGO.

0.4.14 *Valor del Ejercicio*

- Uno de los puntos fuertes de aprendizaje de este ejercicio es la manipulación de colecciones en Python a cualquier otra colección que nosotros queramos.
- Básicamente los diccionarios iniciales vamos a presentarlos y transformarlos de varias maneras.
- En la vida real de la programación en Python, las colecciones de datos tienen que ser manipuladas de manera que nosotros podamos usarlas de varias maneras y en varias aplicaciones.
- Recordar que para este fin tenemos 2 maneras esenciales:
  - (1) Con funciones `sort / map / filter / reduce`
  - (2) Con una variable de colección vacía y recorrido (bucle) a la colección DATO. De esta manera vamos poblando la colección vacía nueva de la manera que queramos.
- BUENA SUERTE !!!

[ ]:
```
# *** CÓDIGO # 1 PROPORCIONADO PARA EL EJERCICIO ***

# ============================================================
# LÍNEA DE CÓDIGO PARA INSTALAR EL PAQUETE EXTERNO "varname"
# NOTA:
# A veces al cerrar Colab
# Tenemos que volver a instalar el paquete externo
# ============================================================

# https://pypi.org/project/varname/
```

```
# !pip install -U varname
```

[ ]:
```
# *** CÓDIGO # 2 PROPORCIONADO PARA EL EJERCICIO ***

# ================================
# DATOS INICIALES DEL EJERCICIO
# ================================

IVA = 0.15

articulos_de_computacion = {
    '00A-1' : ('Laptop I7' , 1250.99),
    '00A-2' : ('Teclado Gamer' , 55.65),
    '00A-3' : ('Mouse Gamer' , 24.99),
    '00A-4' : ('Web Cam 4K' , 88.38),
    '00A-5' : ('Monitor 32 pulgadas' , 228.75),
}

articulos_gaming = {
    '00B-1' : ('Play Station 5' , 550.59),
    '00B-2' : ('Nintendo Switch' , 335.78),
    '00B-3' : ('Steam Deck' , 845.99),
    '00B-4' : ('Control XBox' , 56.88),
    '00B-5' : ('RetroPie 400GB' , 158.99),
}

articulos_telefonos_tablets = {
    '00C-1' : ('Samsung Galaxy S23' , 1200.99),
    '00C-2' : ('Iphone 15' , 1325.88),
    '00C-3' : ('Microsoft Surface 500GB' , 775.66),
    '00C-4' : ('IPad Air 500GB' , 980.45),
    '00C-5' : ('Huawei XPad 128GB' , 335.28),
}
```

[ ]:
```
# *** CÓDIGO # 3 PROPORCIONADO PARA EL EJERCICIO ***

# ================================
# FUNCIONES AUXILIARES PARA TEST
# ================================


def presentar_lista(lista):
  for index, elemento in enumerate(lista):
    print( '{} | {}'.format( index+1 , elemento ) )
  # end for
# end def
```

```
def presentar_diccionario(diccionario):
  for clave, valor in diccionario.items():
    print( '{} | {}'.format( clave , valor ) )
  # end for
# end def
```

0.4.15 El ejercicio empieza desde AQUÍ:

[ ]:

EJERCICIO 26

Conociendo todos los días de nuestros cumpleaños hasta la fecha de hoy

TEMAS: Librería datetime / str.format() / Funciones Normales / Bucle WHILE / Diccionarios / Operaciones entre Fechas

- El siguiente ejercicio consiste en llegar a la construcción de la función **generar_fechas_nacimiento()**
- El ejemplo de ejecución de esta función se muestra en el siguiente código de muestra:

```
# =====
# TEST
# =====

generar_fechas_nacimiento()
```

- Tomando en cuenta mi fecha de cumpleaños: 28 de Julio de 1987, esto es lo que pide el programa por medio de inputs:

```
==========================================
CONOCE TODOS LOS DÍAS EN QUE NACISTE !! :)
==========================================

Por favor ingrese su fecha de nacimiento (AÑO / MES / DÍA)
Ingrese el año : 1987
Ingrese el mes : 7
Ingrese el día : 28

Sus cumpleaños hasta la fecha han sido:
 0) MARTES 28 de Julio de 1987
 1) JUEVES 28 de Julio de 1988
 2) VIERNES 28 de Julio de 1989
 3) SÁBADO 28 de Julio de 1990
 4) DOMINGO 28 de Julio de 1991
 5) MARTES 28 de Julio de 1992
 6) MIÉRCOLES 28 de Julio de 1993
 7) JUEVES 28 de Julio de 1994
```

```
 8) VIERNES 28 de Julio de 1995
 9) DOMINGO 28 de Julio de 1996
10) LUNES 28 de Julio de 1997
11) MARTES 28 de Julio de 1998
12) MIÉRCOLES 28 de Julio de 1999
13) VIERNES 28 de Julio de 2000
14) SÁBADO 28 de Julio de 2001
15) DOMINGO 28 de Julio de 2002
16) LUNES 28 de Julio de 2003
17) MIÉRCOLES 28 de Julio de 2004
18) JUEVES 28 de Julio de 2005
19) VIERNES 28 de Julio de 2006
20) SÁBADO 28 de Julio de 2007
21) LUNES 28 de Julio de 2008
22) MARTES 28 de Julio de 2009
23) MIÉRCOLES 28 de Julio de 2010
24) JUEVES 28 de Julio de 2011
25) SÁBADO 28 de Julio de 2012
26) DOMINGO 28 de Julio de 2013
27) LUNES 28 de Julio de 2014
28) MARTES 28 de Julio de 2015
29) JUEVES 28 de Julio de 2016
30) VIERNES 28 de Julio de 2017
31) SÁBADO 28 de Julio de 2018
32) DOMINGO 28 de Julio de 2019
33) MARTES 28 de Julio de 2020
34) MIÉRCOLES 28 de Julio de 2021
35) JUEVES 28 de Julio de 2022
36) VIERNES 28 de Julio de 2023
```

- Esta primera parte donde se pide ingresar año, mes y día, se debe ingresar con números enteros nuestra fecha de cumpleaños (por medio de 3 inputs)
- Hecho esto se genera automáticamente una lista numerada desde el 0 (el día en que nacimos) donde se nos muestra la fecha en español con cada uno de nuestros cumpleaños hasta la fecha actual.
- Al momento que se realiza este ejercicio es en Febrero de 2024, por lo que mi cumpleaños todavía no ha sucedido entonces se muestra hasta el año 2023
- La numeración entonces muestra también mi edad actual (la última numeración)

0.4.16 TIPS

- Recordar datetime y sus métodos
- datetime para generar fecha actual y una fecha cualquiera
- Recordar que los atributos y métodos de un objeto de tipo datetime
- Recordar que podemos personalizar la fecha creando un artilugio para poder pasar los valores a palabras.

- BUENA SUERTE !!!

[ ]:

EJERCICIO 27

Reloj mundial para 5 lugares del mundo cada 5 segundos

TEMAS: Librería pytz / Librería datetime / Librería time / Método .sleep() / Listas / Tuplas / Función Interna zip / Bucle FOR / Bucle WHILE / while True / str.format() / Funciones Normales

- El siguiente ejercicio tiene por finalidad generar un reloj mundial que muestre la hora de 5 lugares distintos del mundo:
  - Hamburgo / Alemania
  - Quito / Ecuador
  - Tokio / Japón
  - Sydney / Australia
  - Moscú / Rusia
- El objetivo del ejercicio es por medio de un bucle **`while True`** generar el siguiente resultado en la terminal de Python:

```
---------------
REPETICIÓN # 1
---------------
1) Hamburgo: 25/01/2024 - 23h:13m:04s PM
2) Quito: 25/01/2024 - 17h:13m:05s PM
3) Tokio: 26/01/2024 - 07h:13m:06s AM
4) Sydney: 26/01/2024 - 09h:13m:07s AM
5) Moscú: 26/01/2024 - 01h:13m:08s AM


---------------
REPETICIÓN # 2
---------------
1) Hamburgo: 25/01/2024 - 23h:13m:09s PM
2) Quito: 25/01/2024 - 17h:13m:10s PM
3) Tokio: 26/01/2024 - 07h:13m:11s AM
4) Sydney: 26/01/2024 - 09h:13m:12s AM
5) Moscú: 26/01/2024 - 01h:13m:13s AM


---------------
REPETICIÓN # 3
---------------
1) Hamburgo: 25/01/2024 - 23h:13m:14s PM
2) Quito: 25/01/2024 - 17h:13m:15s PM
3) Tokio: 26/01/2024 - 07h:13m:16s AM
4) Sydney: 26/01/2024 - 09h:13m:17s AM
5) Moscú: 26/01/2024 - 01h:13m:18s AM
```

```
--------------
REPETICIÓN # 4
--------------
1) Hamburgo: 25/01/2024 - 23h:13m:19s PM
2) Quito: 25/01/2024 - 17h:13m:20s PM
3) Tokio: 26/01/2024 - 07h:13m:21s AM
4) Sydney: 26/01/2024 - 09h:13m:22s AM
5) Moscú: 26/01/2024 - 01h:13m:23s AM


--------------
REPETICIÓN # 5
--------------
1) Hamburgo: 25/01/2024 - 23h:13m:24s PM
2) Quito: 25/01/2024 - 17h:13m:25s PM
3) Tokio: 26/01/2024 - 07h:13m:26s AM
4) Sydney: 26/01/2024 - 09h:13m:27s AM
5) Moscú: 26/01/2024 - 01h:13m:28s AM



FIN DE NUESTRAS ITERACIONES - BYE !
```

0.4.17 *Consideraciones:*

- Son 5 repeticiones
- Es decir en algún momento de este while True debe de existir un **`break`**
- En cada iteración / repetición se muestra la fecha y la hora en ese **`formato personalizado`** que se indica.
- Nosotros ya sabemos que podemos generar la hora y fecha actual con datetime de esta manera:
  - `datetime.datetime.now()`
- Sin embargo, como observamos nosotros generamos la fecha actual en diferentes lugares, y esto es posible tomando en cuenta las zonas horarias.
- Este tema de las zonas horarias es nuevo, y para esto se necesita la librería **`pytz`**
- El uso de la librería pytz se muestra de manera directa en el siguiente código elaborado que se brinda como recurso al estudiante:

```
# ------------------------
# *** Código de DATO 1 ***
# ------------------------

# ==================================
# Importación y uso de librería pytz
# ==================================
import pytz
```

```
zones = pytz.all_timezones

print(zones) # me imprime todas las zonas horarias


# ====================
# EJ de zonas horarias
# ====================
# Hamburgo / Alemania: 'Europe/Amsterdam'
# Quito / Ecuador:     'America/Lima'
# Tokio / Japón:       'Asia/Tokyo'
# Sydney / Australia:  'Australia/Sydney'
# Moscú / Rusia:       'Europe/Moscow'


# =========================
# Uso de pytz con datetime
# =========================
import datetime as dt

# creando la variable de zona_horaria con "pytz.timezone"
zona_america_lima = pytz.timezone('America/Lima')

# usando la zona horaria con datetime.now
# EJ: hora actual en Quito, Ecuador
print( dt.datetime.now(zona_america_lima) ) # EJ: 2024-01-24 11:43:49.458354-05:00
```

- Que nos vendría a dar el siguiente resultado en la terminal:

```
['Africa/Abidjan', 'Africa/Accra', 'Africa/Addis_Ababa', 'Africa/Algiers', 'Africa/Asmara', 'A
2024-01-24 11:43:49.458354-05:00
```

0.4.18 IMPORTANTE:

- Observar que cada vez que se imprime la hora fecha, esto se hace cada segundo, es decir, no es inmediato (recordar `time.sleep()`)
- Obsrvar que los dígitos de día, mes, hora, minuto y segundo se muestran siempre de a 2 dígitos.
- Es decir, si tenemos una fecha tal como:
  - `1/1/2010 - 1:1:2 AM`
  - Se nos debe mostrar como:
  - `01/01/2010 - 01:01:02 AM`
- Esta corrección de los dígitos se hace con un par de funciones
- Igualmente el tener AM y PM
- Esto es parte de la tarea del estudiante llegar a presentar la fecha:
  - Desde un `datetime.datetime.now()`
  - Al formato presentado en la pantalla
- Revisar en el código de arriba como se utiliza `pytz` para generar la hora/fecha en distintas zonas horarias.

- Revisar el resultado final para saber que se debe hacer para llegar a ese objetivo.
- BUENA SUERTE !!!

[ ]:
```
# ------------------------
# *** Código de DATO 1 ***
# ------------------------

# ==================================
# Importación y uso de librería pytz
# ==================================


# =====================
# EJ de zonas horarias
# =====================
# Hamburgo / Alemania: 'Europe/Amsterdam'
# Quito / Ecuador:     'America/Lima'
# Tokio / Japón:       'Asia/Tokyo'
# Sydney / Australia:  'Australia/Sydney'
# Moscú / Rusia:       'Europe/Moscow'


# ==========================
# Uso de pytz con datetime
# ==========================

# creando la variable de zona_horaria con "pytz.timezone"


# usando la zona horaria con datetime.now
# EJ: hora actual en Quito, Ecuador
```

[ ]:
```

```

EJERCICIO 28

Programa Conversor de Unidades de Longitud / Distancia

TEMAS: Funciones Normales / Diccionarios / Dictionary Comprehension / str.format() / str.format() para formato de números / str.format() para formato de texto / Listas / Casting de Variables Básicas / input()

- Este ejercicio pretende crear un programa en Python que nos sirva para almacenar unidades de distancia ingresadas por el usuario en **METROS**.
- Y adicionalmente convertir estos valores de distancia y presentarlos en las siguientes unidades:
  - METROS [M] (ingresadas por el usuario, por defecto)
  - MILÍMETROS [MM]
  - CENTÍMETROS [CM]
  - PIES [FT]

  - PULGADAS [IN]
- Básicamente se pide construir las siguientes 2 funciones en Python:
  - **1) `presentar_distancias_y_conversiones()`**
  - **2) `ingresar_y_convertir_distancias()`**

0.4.19 *Código de Dato y Explicación sobre la Conversión de Unidades de Longitud*

- A manera de dato se muestra un código preparado para el estudiante.
- En este código se muestran 2 puntos importantes:
  - 1) Las respectivas conversiones entre: metro, milímetro, centímetro, pie & pulgada
  - 2) Una función auxiliar que nos ayuda a recorrer los valores de clave y valor de un diccionario
- Este código se muestra a continuación:

```
# ===============
# CÓDIGO DE DATO
# ===============

# --------------------------
# (A) Conversión de Unidades
# --------------------------
# 1 metro (m) = 1000 milímetros (mm)
# 1 metro (m) = 100 centímetros (cm)
# 1 pie (ft) = 0.3048 metros (m)      => 1 metro (m) = (1/0.3048) pies (ft)
# 1 pulgada (in) = 0.0254 metros (m)  => 1 metro (m) = (1/0.0254) pulgadas (in)


# ----------------------------------------
# (B) Función para presentar un diccionario
# ----------------------------------------
def presentar_diccionario( diccionario ):
  for clave, valor in diccionario.items():
    print( '{} | {}'.format(clave, valor) )
  # end for
# end def
```

0.4.20 *Funcionamiento y Código Test*

- Estas 2 funciones deben ser creadas de tal manera que el siguiente código test debe ser ejecutado:

```
# ===========
# TEST FINAL
# ===========

presentar_distancias_y_conversiones( ingresar_y_convertir_distancias() )
```

- Como se observa la función **`ingresar_y_convertir_distancias()`** es el argumento de la función **`presentar_distancias_y_conversiones()`**
- Y el resultado en consola debe ser el siguiente:

```
Ingresar valores de distancias en METROS (m):
Ingrese distancia d1 [m] : 5
Ingrese distancia d2 [m] : 2.55
Ingrese distancia d3 [m] : 0.97
Ingrese distancia d4 [m] : 1.2
Ingrese distancia d5 [m] : 3

RESULTADOS:

DISTANCIA |     [M]     |     [MM]     |     [CM]     |     [FT]     |     [IN]
       d1 |   5.000     |   5000.000   |   500.000    |    16.404    |   196.850
       d2 |   2.550     |   2550.000   |   255.000    |    8.366     |   100.394
       d3 |   0.970     |   970.000    |    97.000    |    3.182     |    38.189
       d4 |   1.200     |   1200.000   |   120.000    |    3.937     |    47.244
       d5 |   3.000     |   3000.000   |   300.000    |    9.843     |   118.110
```

- En un inicio como se puede observar, por medio de **input** el usuario debe ingresar 5 valores de distancia en metros.
- Una vez que los valores son ingresados (pueden ser números enteros o de punto flotante), se presentan los resultados tal y como se muestra:
  - El espaciado no es coincidencia (recordar `str.format()`)
  - La alineación del texto y los números en la tabla es tal y como se muestra
  - Las distancias resultantes en cada una de sus unidades se muestran con 3 puntos decimales SIEMPRE
- Recordar que existen varias maneras de poder generar este resultado, NO existe una ÚNICA SOLUCIÓN
- Pero de ser posible, no olvidar el uso de **"Dictionary Comprehension"**
- BUENA SUERTE !!!

[ ]:
```python
# ===============
# CÓDIGO DE DATO
# ===============

# --------------------------
# (A) Conversión de Unidades
# --------------------------
# 1 metro (m) = 1000 milímetros (mm)
# 1 metro (m) = 100 centímetros (cm)
# 1 pie (ft) = 0.3048 metros (m)      => 1 metro (m) = (1/0.3048) pies (ft)
# 1 pulgada (in) = 0.0254 metros (m)  => 1 metro (m) = (1/0.0254) pulgadas (in)


# ----------------------------------------
# (B) Función para presentar un diccionario
# ----------------------------------------
def presentar_diccionario( diccionario ):
  for clave, valor in diccionario.items():
```

```
    print( '{} | {}'.format(clave, valor) )
  # end for
# end def
```

[ ]:

EJERCICIO 29

Función Graficadora de Funciones BÁSICA en Python

TEMAS: Librería numpy / Librería matplotlib / Keyword import / Keyword as / Funciones Normales VS. Funciones Lambda / Funciones como Variables / Indexing de Colecciones / Tuplas / Listas / str.format()

- **NOTA IMPORTANTE:** Este ejercicio toma en cuenta el repaso y conocimiento que se dio al último del Crash Course con los temas: numpy y matplotlib.
- Teniendo en cuenta lo anterior, se pide al alumno desarrollar la función:
  –

0.4.21 `graficar_X_vs_Y`

0.4.22 *Función: `graficar_X_vs_Y( funcion_y, min, max, color )`*

- Esta función tiene 4 parámetros:
  - **`funcion_y`** : Representa la función a gráficar definida en Python y que pasa como argumento para la función principal **graficar_X_vs_Y**
  - **`min`** : El valor MÍNIMO de los valores de X (el que más a la IZQUIERDA se encuentra del eje de las X)
  - **`max`** : El valor MÁXIMO de los valores de X (el que más a la DERECHA se encuentra del eje de las X)
  - **`color`** : Es un argumento de tipo STRING que representa el valor del color con el que se muestra la gráfica
- Prácticamente, lo que tenemos que hacer es llevar esa teoría que aprendimos de `matplotlib` y `numpy` a la práctica y combinándola con lo que hemos aprendido sobre funciones en Python.

0.4.23 *Funciones a Graficar*

- Las funciones que tomaremos en cuenta para nuestras gráficas son las siguientes:

```
# funcion_y1 : y1 = f1(x) = x + 10
# funcion_y2 : y2 = f2(x) = 1 / x
# funcion_y3 : y3 = f3(x) = x^2 -5*x + 8
# funcion_y4 : y4 = f4(x) = x^3
```

- Para un mejor entendimiento, veamos el código test a ejecutar.
- Básicamente cada una de estas funciones debe ser ejecutada en un plot con matplotlib y por medio de la función:
  - ***`graficar_X_vs_Y( funcion_y, min, max, color )`***

0.4.24 *Consideración IMPORTANTE*

- En COLAB no necesitamos realizar una instalación !pip de las librerías matplotlib y numpy.
- Por tanto, podemos tranquilamente ejecutar una celda de código que represente las improtaciones de ambas librerías.
- Esta celda entonces debe ejecutarse al inicio, antes que nada, y no va a darnos ningún problema:

```
# ========================================
# IMPORTACIONES NECESARIAS PARA EL EJERCICIO
# ========================================
# => NOTA: No es necesario instalar estas librerías en COLAB

import numpy as np
import matplotlib.pyplot as plt
```

0.4.25 *Código TEST a ejecutar*

- Una vez que hayamos desarrollado entonces la función:
  - ***graficar_X_vs_Y(funcion_y, min, max, color)***
- Vamos a ponerla en práctica por medio del siguiente código TEST:

```
# ============================
# TEST RESULTADOS DE GRÁFICAS
# ============================

print('\n\nGRÁFICA 1\n')
graficar_X_vs_Y( funcion_y1 , -50, 50, 'y' )

print('\n\nGRÁFICA 2\n')
graficar_X_vs_Y( funcion_y2 , -50, 50, 'r' )

print('\n\nGRÁFICA 3\n')
graficar_X_vs_Y( funcion_y3 , -10, 10, 'g' )

print('\n\nGRÁFICA 4\n')
graficar_X_vs_Y( funcion_y4 , -5, 5, 'b' )
```

- Cuando ejecutemos esto, y si todo ha salido bien por nuestro lado, en la consola de Python vamos a ver el siguiente resultado:

- **Consejo Importante:** El ejercicio puede causar un poco de confusión, pero sólamente recordar como se hizo esto en el Crash Course, tal y como se dijo arriba, la única diferencia es que estamos utilizando funciones.
- BUENA SUERTE !!!

[ ]:
```
# ========================================
# IMPORTACIONES NECESARIAS PARA EL EJERCICIO
# ========================================
# => NOTA: No es necesario instalar estas librerías en COLAB

import numpy as np
import matplotlib.pyplot as plt
```

[ ]:
```

```

EJERCICIO 30

0.5 ÚLTIMO EJERCICIO : Función "generar_subplots()" para gráficos dinámicos e inteligentes

TEMAS: Librería numpy / Librería matplotlib / Plots y Subplots (TEMA NUEVO) / Keyword import / Keyword as / Funciones Normales / Funciones Lambda / Funciones* args / Funciones como Variables / Indexing de Colecciones / Tuplas / Listas / Diccionarios / str.format() / Librería random / Librería math / Bucle FOR / Bucles Anidados / Condicional IF / Listas y Funciones / enumerate()

- Para despedida de este curso se ha formulado un súper último ejercicio.
- Este ejercicio tiene el objetivo principal de invitar al estudiante a usar la mayor cantidad de herramientas aprendidas no solo en el Crash Course, sino también durante la resolución de estos ejercicios propuestos.
- Herramientas y Técnicas de programación.
- El ejercicio puede parecer muy sencillo, pero no lo es.
- Este ejercicio final se compone de 2 partes:
  - ***PRIMERA PARTE*** (Introducción a un tema nuevo - Subplots): La primera parte de este ejercicio es una presentación de teoría que se desarrolla junto con el instructor. El objetivo es topar un tema importante que no se cubrió en el Crash Course, me refiero a los subplots. Para este fin, un script de varias celdas y su explicación han sido elaborados.
  - ***SEGUNDA PARTE*** (Resolución del ejercicio como tal): Una vez explicada esta parte teórica mencionada, el estudiante deberá aplicar este nuevo conocimiento, junto con todo lo aprendido hasta el momento, con el fin de desarrollar la función `generar_subplots()` que se va a explicar a continuación.

0.5.1 *Función:* `generar_subplots( f(x) )`

- Esta es la función que se debe desarrollar para este ejercicio.
- Una sola función que representa un gran trabajo de por medio.

- Tal y como se indica se construye con UN SOLO PARÁMETRO, una variable que representa una función (Recordar que las funciones pueden ser también variables, como todo en Python)
- Para este ejercicio, se ha planteado 7 funciones `f(x)` que servirán para ejemplificar el uso de la función `generar_subplots()`
- Estas 7 funciones son las siguientes:

0.5.2 *Códigos TEST para comprobar resultados*

- La mejor manera de explicar el funcionamiento de la función `generar_subplots()` es por medio de 7 códigos test que se presentan a continuación.
- Primero vamos a presentar al estudiante cada uno de estos códigos test y el resultado en pantalla esperado.
- Se pide al estudiante analizar los resultados esperados en pantalla.
- Luego de esto se procede a una pequeña explicación para dejar sentadas las bases del ejercicio y qué es lo que se espera del estudiante al programar esta función.

```
# ===============
# TEST FINAL 1
# => 1 Gráfica
# ===============

generar_subplots( y4 )



# ===============
# TEST FINAL 2
# => 2 Gráficas
# ===============

generar_subplots( y5, y6 )



# ===============
# TEST FINAL 3
# => 3 Gráficas
# ===============

generar_subplots( y5, y6, y7 )



# ===============
# TEST FINAL 4
# => 4 Gráficas
# ===============
```

```
generar_subplots( y1, y2, y3, y4 )



# ===============
# TEST FINAL 5
# => 5 Gráficas
# ===============

generar_subplots( y1, y2, y3, y4, y5 )



# ===============
# TEST FINAL 6
# => 6 Gráficas
# ===============

generar_subplots( y1, y2, y3, y4, y5, y6 )



# ===============
# TEST FINAL 7
# => 7 Gráficas
# ===============

generar_subplots( y1, y2, y3, y4, y5, y6, y7 )
```

0.5.3 *Explicación y Objetivos del Ejercicio*

- **1)** Observar en primer lugar que la función `generar_subplots()` puede recibir un número indeterminado de agumentos (recordar funciones *args)
  - **RECORDAR: Parámetros VS. Argumentos**
    - `PARÁMETROS` : Cuando se construye la función con def por ejemplo. (Una función se construye con PARÁMETROS)
    - `ARGUMENTOS` : Cuando se invoca / ejecuta la función con esos valores en los paréntesis. (Una función recibe ARGUMENTOS)
- **2)** También observar que dependiendo del número de argumentos, que vienen a ser el "número de funciones a graficar" nosotros tenemos en pantalla una disposición diferente de las gráficas:
  - **`1 argumento`** : Presentamos 1 sola gráfica en pantalla
  - **`2 argumentos`** : Se presentan 2 gráficas en el orden que se puso dentro de los argumentos y organizadas en filas (o en disposición vertical)

  - `3 argumentos` : Se presentan 3 gráficas en el orden que se puso dentro de los argumentos y organizadas en columnas (o en disposición horizontal)
  - `4 argumentos` : Se presentan 4 gráficas en el orden de los argumentos y organizadas en una cuadrícula 2 x 2.
  - `5 argumentos` : 5 gráficas en sentido vertical o en filas.
  - `6 argumentos` : Se presentan las gráficas en una cuadrícula de 2 filas y 3 columnas.
  - `7 o más argumentos` : A partir de 7 argumentos o funciones que se reciben, la disposición de las gráficas es SIEMPRE vertical o en filas.

- **3)** IMPORTANTE: Cada vez que se ejecuta la función `generar_subplots()` los colores de líneas de cada gráfica y el marcador de los puntos de la gráfica se genera al AZAR (recordar librería random)

  - `Color al Azar` : En la teoría que se va a explicar como parte inicial del ejercicio se va a explicar un poco sobre teoría de colores y su valor HEXADECIMAL. Explicado esto, el estudiante está en estos momentos en toda la capacidad de generar una función que retorne un string con un código de color hexadecimal al azar.

  - `Puntero / Marcador de gráfica al Azar` : Esto viene de la teoría de la librería Matplotlib. En el siguiente link de la librería oficial se explica todas las posibilidades de punteros o marcadores que podemos usar en nuestras gráficas:

https://matplotlib.org/stable/api/markers_api.html

- **4)** Los argumentos que recibe la función `generar_subplots()` contienen la información de las ecuaciones mostradas anteriormente. No solamente las funciones, estos parámetros tienen que ser al final del día una variable compuesta que tenga la información de estas ecuaciones
  - **NOTA:** Ya vimos algo de esto en el ejercicio anterior (EJ 29), básicamente se necesita una descripción de la ecuación con un string y la función en sí

0.5.4 *IMPORTANTE !!!*

- Recordar como se indicó arriba, que para realizar este ejercicio es necesaria una introducción teórica a cargo del profesor que será realizada a continuación de la explicación de este enunciado.
- Este reforzamiento teórico pretende introducir y explicar al estudiante fundamentos avanzados de la librería matplotlib y el uso de subplots
- BUENA SUERTE CON ESTE ÚLTIMO EJERCICIO !!!
- ÁNIMO !!! Y recuerda que con dedicación, pasión y compromiso, TODO es posible en esta vida !!!

1 Repaso y Explicación de Conceptos Nuevos

- Esta sección del ejercicio esta destinada a un repaso de los conceptos vistos en matplotlib y numpy
- Adicionalmente, tiene la intención de adentrar al estudiante a un nuevo concepto que son los **"subplots"**
- Este nuevo conocimiento es necesario para la resolución de este ejercicio.
- El tema de los subplots no fue tratado en el `Crash Course`, pero lo haremos aquí.
- Inicialmente verán estos bloques de código sólo con comentarios y vacíos.

- Esto es porque a intención es que puedan desarrollar esta teoría de la mano del profesor y entender el concepto.
- Explicada esta parte teórica, inicia el ejercicio

1.1 *Importaciones y Código Inicial*

- Vamos a iniciar con las importaciones necesarias no solo para esta parte teórica sino también para la ejecución del ejercicio.
- Por lo que una vez ejecutado este bloque de las importaciones ya no es necesario hacerlo luego para el desarrollo del ejercicio (pero lo pueden hacer si gustan, con el fin que tengan el bloque de importaciones también en la parte de la solución del ejercicio)
- Adicionalmente, se propone un segundo bloque de código con 4 funciones `f(x)` que nos van a ayudar a ejemplicar esta parte teórica:

```
y1 = lambda x : x + 10
y2 = lambda x : 1 / x
y3 = lambda x : x**2 - 5*x + 8
y4 = lambda x : x**3
```

- Estas funciones se las representa con funciones `lambda`
- Se generan los valores de x con numpy
- Y luego de esto se generan los valores de y
- Con valores en `x`, y en `y` en forma de array, estamos listos para representar y explicar las gráficas.

[ ]:
```
# ========================================
# IMPORTACIONES NECESARIAS PARA EL EJERCICIO
# ========================================
# => NOTA: No es necesario instalar estas librerías en COLAB
```

[ ]:
```
# ============================================================
# DATOS NECESARIOS PARA LA REALIZACIÓN DE NUESTRAS GRÁFICAS
# ============================================================


# -------------------
# Funciones f(x) / y
# -------------------


# -------------------------------------
# Valores de X generados con numpy (np)
# -------------------------------------


# ---------------------------------------------
# Valores de Y generados con las funciones y numpy
# ---------------------------------------------
```

1.2 *plt.plot() y la importancia de plt.show()*

- Básicamente, cada vez que usamos `plt` estamos por así decirlo "cargando" esta variable con los gráficos que vayamos generando.
- Estos gráficos se muestran en el momento que nosotros usamos `plt.show()`
- Por tanto, no caer en el primer ERROR que se comete al trazar gráficas con matplotlib que es ir haciendo `plt.plot()` y olvidarse o no usar correctamente el `plt.show()`
- Para eso por favor observar las siguientes 2 celdas de código que han sido preparadas para esta explicación:

[ ]:
```
# ====================================
# USO INCORRECTO DE plt.show()
# - No usándolo
# - Usándolo en la posición incorrecta
# ====================================
```

[ ]:
```
# ======================================================================
# USO CORRECTO DE plt.plot() & plt.show()
# - Cada vez que usamos plt.show()
# - Los gráficos generados con plt.plot()
# - Se muestran
# - Y es como que la variable plt se queda nuevamente vacía
# - La volvemos a cargar con gráficos y los mostramos con plt.show()
# ======================================================================
```

1.3 *plt.subplots() : Plots dentro de un Plot*

- Lo anterior ya lo sabíamos.
- Aquí viene la parte nueva y es que dentro de un plot podemos tener varios plots.
- De manera correcta se dice: **un plot contiene subplots**
- Esta sección tiene 2 objetivos fundamentales
  - `OBJETIVO # 1` : Vamos a aprender la teoría de subplots, cómo hacerlos, como es la distribución de los mismos. Vamos a entender el código que permite la creación de los mismos.
  - `OBJETIVO # 2` : Dar formato a los subplots y plots. Vamos a modificar colores, texto, tipos de línea. Vamos a aprender a modificar su tamaño dentro del Colab
- La explicación con más detalle se da en cada uno de estos puntos, y de manera crucial dentro del código que se va a generar para esta explicación.

1.3.1 *Objetivo # 1 : Teoría y Configuración de Subplots dentro de un Plot*

- Vamos a aprender 4 disposiciones / configuraciones de los subplot:
  - A) Subplots en construcción vertical (en filas)
  - B) Subplots en Construcción Horizontal (en columnas)
  - C) Subplots en Cuadrícula 2 x 2

- D) Subplot en Cuadrícula N x M y estándar fig & axs

1.3.2 *Objetivo # 2 : Formato y Personalización de Plots y Subplots*

- Dentro del literal A, donde vamos a aprender la construcción de Subplots en vertical, vamos a analizar como podemos darles formato a nuestras gráficas.
- Vamos a aprender como cambiar los colores, fuentes, tamaño de texto de estas gráficas.
- Vamos a aprender a modificar su tamaño dentro de Colab.
- Vamos a aprender una manera más estilizada por medio de los diccionarios `fontdict` a cómo dar este formato de texto de una manera fácil.
- Vamos a presentar un pequeño postulado sobre la teoría de colores y su representación en valores HEXADECIMALES. Dónde los podemos encontrar y como se compone esta configuración.
- De manera adicional, vamos a presentar también una estrategia para el formato y personalización de un plot singular sin subplots

1.3.3 A) Subplots en Construcción Vertical (en filas)

- Aquí empezamos con esta nueva teoría de subplots.
- La primera y más fácil es organizar varias gráficas en filas o por así decirlo en línea seguida.
- Adicionalmente, y tal como lo aclaramos arriba, vamos a aprovechar este punto para explicar como podemos dar formato y personalización a nuestras gráficas.

[ ]:
```
# ========================================
# Manera por default sin ningún formato

# - Entendimiento básico de funcionamiento
# - Sin configuración de fuentes / tamaños
# - En construcción vertical (filas)
# - plt.subplots(X) => X : número de filas
# ========================================
```

[ ]:
```
# ============================================
# Subplots en construcción Vertical + Títulos

# - Agregando títulos a cada subplot
# - Con formato por defecto
# ============================================
```

[ ]:
```
# ================================================================================
# Subplots en construcción Vertical + Títulos + Tamaño de Plot en Colab

# ---------------------------------------
# Ajustando el tamaño del Plot en Colab:
# plt.rcParams['figure.figsize'] = (X, Y)
# ---------------------------------------
# - X => Tamaño horizontal del Plot
# - Y => Tamaño vertical del Plot
```

```
# ------------------------------------------------------
# Reseteando tamaño de siguientes Plot
# matplotlib.rcParams.update(matplotlib.rcParamsDefault)
# ------------------------------------------------------
# - De esta manera volvemos al tamaño por default
# - Con el que Colab muestra los gráficos Plot
# - Siempre usarlo al final luego de haber usado .rcParams


# ----------------------------------------
# Generando espaciado entre subplots
# plt.subplots_adjust(wspace=X , hspace=Y)
# ----------------------------------------
# - X => espaciado horizonal
# - Y => espaciado vertical
# - Ponerlo antes del plt.show()

# ============================================================================
```

[ ]:
```
# ================================================================
# Subplots en Vertical + Títulos + Tamaño + Formato de Texto + grid

# - En esta sección aprenderemos a hacer esto de manera individual
# - Es decir, en cada subplot ingresaremos código de formato
# - Vamos a modificar 4 elementos específicos para dar formato:

#   A) El título principal del plot (subtitle)
#   B) Los títulos de cada subplot (set_title)
#   C) Los labels de los ejes x, y de cada subplot (xaxis, yaxis)
#   D) Vamos a generar un GRID en cada subplot
# ================================================================
```

[ ]:
```
# ================================================================
# Alternativa: fontdict y Colores con HEX

# ---------
# fontdict
# ---------
# - Diccionario de propiedades de texto
# - Se lo puede usar dentro de .suptitle y .set_title
# - Para .suptitle es recomendable solo modificar family y color (dentro del␣
 ↪fontdict)
# - El resto de propiedades se lo pone directamente
# - Pero es una perfecta opción para .set_title
```

```
# ----------------------------
# Color de plot en hexadecimal
# ----------------------------
# - Los colores pueden expresarse en términos hexadecimales
# - Se puede usar la herramienta Google Color Picker Para esto
# - https://g.co/kgs/F4K3j5K
# - Básicamente cada suplot lo podremos definir de esta manera
# - EJ:

#   axs[0].plot( x , y, '#a17d7a', marker='o', linestyle='-', ms=4 )

# - El valor: '#a17d7a' en forma de String es color HEX
# - Básicamente un color hexadecimal se conformar:
#   * con el #
#   * 6 valores que pueden ser = '0123456789ABCDEF'

# - RECORDAR ESTO ÚLTIMO PARA EL EJERCICIO !!!!

# ================================================================
```

[ ]:
```
# ================================================================
# BONUS ADICIONAL:
# Formato en plot singular

# - La teoría vendría a ser la misma
# - Recordemos el código del plot singular que generamos arriba
# ================================================================
```

[ ]:
```
# ================================================================
# => Esta sería la manera de dar formato a este plot singular:
# ================================================================

# -----------------
# Plot sin Formato
# -----------------


# -----------------
# Plot con Formato
# -----------------
```

1.3.4 B) Subplots en Construcción Horizontal (en columnas)

- Vamos a ver como se puede modificar los argumentos del método `.subplots(x,y)` de tal manera que podamos disponer las subgráficas ya no en vertical sino en horizontal.
- Por favor prestar atención a cómo sería el acceso por indexing a cada subplot.

[ ]:
```
# ================================================
# Construcción de subplots en Horizontal

# - Se indica esto en los argumentos del .subplots
# - plt.subplots(1,4)
# - Número de Filas    = 1
# - Número de Columnas = 4
# ================================================
```

1.3.5 C) Subplots en Cuadrícula 2 x 2

- Como aprendimos arriba, a través del método `.subplots(x, y)`, nosotros podemos disponer a manera de matriz 2 x 2 nuestras subgráficas.
- De igual manera esto tiene lógica en el acceso por indexing a cada subplot.
- La relación entre los valroes de .subplots y el indexing se explica en la parte final cuando ejemplicamos un plot con N x M subplots.
- Es decir, dando una configuración random y personalizada.
- Se explica esta configuración 2 x 2 porque es una de las más usadas al momento de mostrar nuestras gráficas con matplotlib.

[ ]:
```
# ================================================================
# Subplot en Cuadrícula de 2 x 2

# - Como vimos en .subplots definimos filas y columnas
# - Esto sugiere que podemos disponer los subplots como queramos
# - Entonces al usar:

#   fig, axs = plt.subplots(2,2)

# - Estamos generando una cuadrícula 2 x 2
# - De 2 filas y 2 columnas
# - Y el acceso a cada una para el .plot
# - Se hace con axs[x,y]
# - Es decir, con indexing
# - Esto se demuestra en el siguiente ejemplo preparado
# ================================================================
```

1.3.6 D) Subplot en Cuadrícula N x M y estándar fig & axs

- De la anterior explicación podemos concluir entonces que podemos distribuir nuestros subplots de la manera que queramos.
- En este ejemplo vamos a realizar una distribución de 2 filas y 3 columnas.
- Lo único que cambiaría sería el acceso a los subíndices para la ejecución de cada plot.
- Adicionalmente, hemos visto que de manera **estándar** tal y como se lo puede leer en la librería oficial de matplotlib, se utiliza:
  - **`fig`** para refererirse a la figura o plot principal que contiene los subplots
  - **`axs`** para referirse a los ejes o subplots que están contenidos dentro del plot.
- Sin embargo, esto es solo un estándar recomendado en la librería y uso de matplotlib.

- Por lo que de manera adicional vamos a ver que podemos cambiar estos términos a palabras en español por ejemplo que tengan más sentido para nosotros

RECORDAR:

- Recordar que el indexing hace referencia a `[fila, columna]`
- Pero su numeración empieza desde el 0
- Mientras que `.subplots(filas, columnas)`
- Hace referencia al número total de filas y columnas

[ ]:
```
# ================================================
# Subplots en configuración N x M

# - Se muestra un ejemplo de 2 filas y 3 columnas
# - Se cambian los nombres de fig y axs
# - Recordar que esto último es sólo un ESTÁNDAR
# ================================================
```

2 El ejercicio comienza desde AQUÍ !!!

[ ]:
```

```

[ ]:
```
#  **** THE END ****
```

2.1 *BONUS FINAL*

2.1.1 Exportar Cuaderno a PDF

- La mejor manera de hacer esto es utilizando Jupyter Notebook en lugar de Colab
- La exportación con Colab produce un PDF incompleto.
- Para hacerlo necesitamos DESCARGAR nuestro cuaderno de Colab como .ipynb
- Luego necesitamos abrir Jupyter Notebook (para esto tanto Python, pip y Jupyter Notebook deben estar instalados en nuestro ordenador)
- Se puede averiguar luego de la instalación si realmente tenemos esto en nuestro computador tecleando lo siguiente en la terminal:
  - python –version
  - pip –version
  - jupyter notebook –version
- Abrimos Jupyter Notebook desde la terminal con:
  - jupyter notebook
- Una vez en Jupyter Notebook abrimos el cuaderno.
- Desde aquí vamos a File + Save and Export Notebook As… + PDF / Webpdf
  - La versión PDF es una versión LATEX con numeración automática y algunos elementos no van a salir igual a como tenemos en el cuaderno.
  - La versión Webpdf es la más confiable

-

2.1.2 *Link para instalar Python:*

  - https://www.python.org/downloads/windows/

-

2.1.3 *Instalación de Jupyter Notebook:*

  - `pip install notebook` : para instalar Jupyter Notebook
  - `jupyter notebook` : en el CMD para correr Jupyter Notebook en nuestro ordenador