**ÍNDICE**

***Temas***  ***nro pág***

**Información Básica** **2**

**Simple Data Types (Tipos de Datos Simples)** **2**

**Variables** **3**

Concatenación **3**

Convertir cualquier tipo de dato a texto **3**

**Compound Data Types (Datos Compuestos) 4**

Listas **4**

Tuplas **4**

Conjuntos **4**

Diccionario (Dict) **5**

**Arithmetic Operators (Operadores Aritméticos)** **6**

Función para saber de qué tipo es un dato **6**

**Comparison Operators (Operadores de Comparación) 7**

Ej de validar contraseña de una cuenta **7**

**Conditionals (Condicionales) 8**

If-Else **8**

Elif **8**

**Logical Operators (Operadores Lógicos) 8**

and **8**

or **8**

not **9**

**String Methods (Métodos de Cadenas) 9**

Observar lo que podemos hacer con un dato **9**

Pasar un string a Mayúsculas **.upper()**

\*Python es **key sensitive** (Sensible a mayúsculas y minúsculas)

\*Los archivos en Python deben terminar en ***.py***

\*Para salida por pantalla usamos el **print (**variable**) o print (“** string **”)**

Ej: print (“Hello World”) print (2+2) print (edad)

\*Para limpiar la consola escribimos en ella **clear**

\*Usamos el ***#*** para comentar (hashtag)

\*Para saber el tipo de dato que contiene una variable debemos poner:

**Print (Type(variable))**

\*Printear de forma horizontal en un for : se usa **,end=””** dentro del print

\*Para **deshacer el último cambio** usamos **control + Z**

\* Para **hacer un salto de línea** usamos **\n**

***Simple Data Type (Datos Simples)***

Existen 4 tipos de Datos Simples:

Strings (cadenas de caracteres)

Para una sola línea de código usamos las comillas simples **“ Hello World ”**

Para más de una línea usamos comillas triples **“”” Hello World… ”””**

Int (enteros)

Ej: 4, 10, 90, 150

Float (nros con coma)

Ej: 2.5, 13.4, 23.7

Bool (True or False)

Solo puede tomar una de esas dos posibilidades

En Python debemos escribir la primer letra de valores booleanos en Mayúscula, ya sea True o sea False

*Variables*

Son espacios de memoria en los que almacenamos datos

Las variables se declaran y se definen

Para almacenar datos en la variable se usa esta forma: variable = dato

Ej: nombre = “Emiliano Schonhals”

Edad = 19

Dni = 44702037

**Concatenación**

Concatenamos cadenas a través del = y el +

Ej: Bienvenida = “Hola ” + nombre + “ como estas?”

**CONVERTIR CUALQUIER TIPO DE DATO A TEXTO**

Usamos los **Fstring**

Ej: nombre = “Emi”

bienvenida = ***f*** “Hola {nombre} como estas?”

print (bienvenida)

**Operadores de pertenencia**

Nos permiten saber si un string está dentro de otro

Son dos: ***in*** y ***not in***

Ej: nombre = “Emiliano”

print (“mili” ***in*** nombre) #nos dará True ya que mili esta dentro de nombre

Ej 2: dni = 44702037

print (“100” ***in*** dni) #nos dará False ya que 100 no está dentro de dni

***Compound Data Types (Datos Compuestos)***

**Listas**

Nos sirven para agrupar datos

Ej: **lista = [“Emi Schonhals”, “Hilario de la Quintana 1297”, 19, 1.87]**

**Para que nos devuelva la lista** debemos poner **print (lista)**

Ahora bien, **si queremos que nos devuelva un valor en especial** debemos poner:

**Print (lista[1])** #en este caso nos devuelve “Hilario de la Quintana 1297”, ya que los índices arrancan en 0

La lista del ejemplo va del índice 0 al 3

**Para modificar un dato en específico** debemos poner:

**Lista[0] = “Axel Nahuel”** #en este caso modificara “Emi Schonhals” y lo sustituirá por “Axel Nahuel”, ya que modificamos el índice 0

**Tupla**

Funciona de la misma forma que las Listas, la diferencia es que **sus datos no se pueden modificar**

**Las tuplas debemos crearlas cuando son datos de SOLO LECTURA, ya que la tupla es inmutable y no puede modificarse**

En vez de corchetes usa paréntesis al definir sus elementos

Ej: **tupla = (“Emi Schonhals”, “Hilario de la Quintana 1297”, 19, 1.87)**

**Conjunto**

Funciona de la misma forma que las dos anteriores pero **sus datos no tienen un orden fijo, y no podemos modificarlos**. Además **no podemos acceder a sus elementos a través de sus índices**. Además de eso en los conjuntos **no puede haber datos duplicados**

En vez de usar corchetes o paréntesis usa llaves para definir sus elementos

Ej: **conjunto = {“Maca Medrano”, 19, 1.65, 22222222}**

**Diccionario (Dict)**

La diferencia con los demás es que en vez de tener un índice por defecto, **el índice será la Key que pongamos nosotros**

De esa forma para **llamar a un elemento debemos poner print (diccionario[´KEY´])**

El elemento que está entre **´ ´** se llama **Key (Clave)**, luego van los dos puntos (:) y por ultimo entre **“ ”** el **Value (Valor)**

Ej: **Diccionario = {**

**´nombre´ : “Emi Schonhals”,**

**´está\_estudiando´ : True**

**´altura´ : 1.87**

**´equipo´ : “Boca Juniors”**

**}**

**Print (diccionario[´equipo´])**

#en este caso nos dará el valor “Boca Juniors”

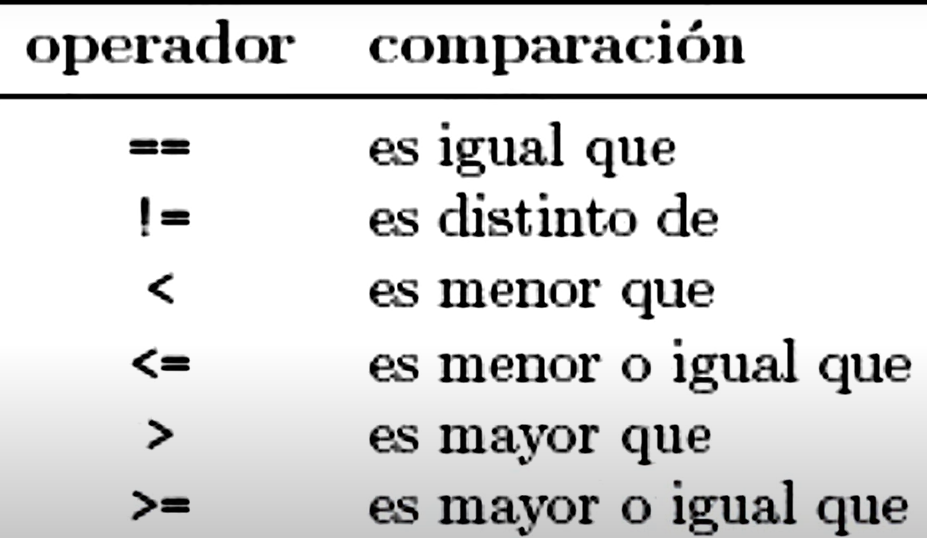
***Arithmetic Operators (Operadores Aritméticos)***

***Función para saber de que tipo es un dato***

**Saber\_tipo = type(*dato que deseamos conocer su tipo*)**

**Print (saber\_tipo)**

***Comparison Operators (Operadores de Comparación)***



**Ej de Validar contraseña de una cuenta:**

password\_guardado = "mitomito10"

password\_ingresado = "emiliano"

print ("La contraseña ingresada es igual a la contraseña de tu cuenta?")

print (password\_guardado == password\_ingresado)

#Si la contraseña coincide nos dará TRUE

#Si la contraseña no coincide nos dará FALSE

#en este caso nos dará False ya que no coinciden las contraseñas

***Conditionals (Condicionales)***

Son códigos a ejecutar en el caso de que la condición se cumpla

edad = 19

if edad >= 18 :

    print ("Sos mayor de edad")

else : print ("Sos menor de edad")

#usamos los dos puntos : en los condicionales para indicar acciones

**En caso de tener que usar varios condicionales anidados usamos ELIF (unión de else if)**

ingreso\_mensual = 10000

if ingreso\_mensual >= 100000 :

    print ("Sos millonario")

elif ingreso\_mensual >= 50000 :  #USAMOS ELIF (unión de else if) PARA CONDICIONALES ANIDADOS

    print ("Tenes buena money")

elif ingreso\_mensual >= 9000 : #LOS ELIF JAMÁS PUEDAN IR DSP DE UN ELSE, SOLO DSP DE UN IF

    print ("Sos clase media")

else: print ("Sos pobre")        #EN ESTE CASO USAMOS SOLO ELSE YA QUE ACCIONA SI NO SE CUMPLE NINGUNA OTRA CONDICIÓN

**Logical Operators (Operadores Lógicos)**

Tenemos 3 operadores lógicos: AND, OR y NOT

**and (&)**

**Ambas condiciones deben ser TRUE**

#AND (&)

resultado = True & True     #Nos devolverá TRUE

resultado2 = True & False   #Nos devolverá FALSE

resultado3 = False & True   #Nos devolverá FALSE

resultado4 = False & False  #Nos devolverá FALSE

**or (|)**

**Basta con que una o ambas condiciones sean verdaderas para ser TRUE**

#OR (|)

resultado5 =  True | True    #Nos devolverá TRUE

resultado6 =  True | False   #Nos devolverá TRUE

resultado7 =  False | True   #Nos devolverá TRUE

resultado8 =  False | False  #Nos devolverá FALSE

**not (not True o not False)**

Es un operador de negación, **invertirá la condición que le pongamos**

#NOT (not True o not False)

resultado9 = not True    #Nos devuelve False

resultado10 = not False  #Nos devuelve True

**String Methods (Métodos de Cadenas)**

La función DIR nos permite observar por consola las cosas que podemos hacer con un dato determinado

En este caso usaremos strings

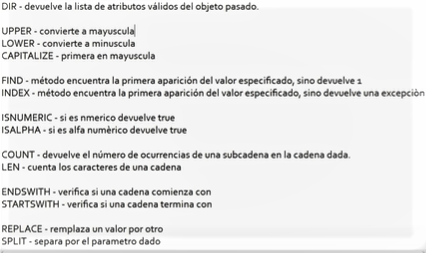
La forma de verlo es:

cadena = "Emiliano"

print (dir(cadena))

**Los métodos se escriben de esta forma:** **dato.método()**

**Los métodos son funciones específicas de objetos**

Podemos usar diversos métodos para strings:

Pasar un string a Mayúsculas **.upper()**

mayus = cadena.upper()

print (mayus)

o también

mayus2 = "hola que tal".upper()

print (mayus2)

Pasar un string a minúsculas **.lower()**

minus = cadena.lower()

print (minus)

Primera letra en mayúscula **.capitalize()**

primer\_letra\_mayus = cadena.capitalize()

print (primer\_letra\_mayus)

# El funcionamiento del método .capitalize() es que 1ro convierte todo el string a minúscula y luego pone en mayúscula solo la primer letra del string

Buscar una cadena dentro de otra cadena **.find(“ ”)**

busqueda\_find = cadena.find("ilia")

print (busqueda\_find)

#SI NOS RETORNA -1 SIGNIFICA QUE NO SE ENCUENTRA LA CADENA, CASO CONTRARIO NOS RETORNARÁ LA POSICION DONDE SE ENCUENTRA

Python es **key sensitive** (sensible a mayúsculas y minúsculas), eso hay q tenerlo en cuenta a la hora de buscar

Otra forma de buscar una cadena dentro de otra cadena **.index(“ ”)**

Funciona igual que el método .find(“ ”), pero **si no encuentra resultados en vez de devolvernos -1 nos devolverá una excepción en forma de error**

busqueda\_index = cadena.index("jose")

print (busqueda\_index)

Saber si el string es numérico **.isnumeric()**

**Nos ayuda a saber si el string contiene sólo números**

Es\_numerico = cadena.isnumeric()

print (es\_numerico)  #Si es numérico nos devuelve TRUE, caso contrario nos devuelve FALSE

Saber si el string es alfanumérico **.isalpha()**

es\_alfanumerico = cadena.isalpha()

print (es\_alfanumerico)  #Si es alfanumérico devuelve TRUE, caso contrario devuelve FALSE

**IMPORTANTE: LOS ESPACIOS NO SON ALFANUMÉRICOS SINO QUE SON CARACTERES ESPECIALES, POR ENDE SI EL STRING CONTIENE ESPACIOS NOS DEVOLVERÁ FALSE**

Saber cuántas veces se encontró una coincidencia dentro de un string **.count(“ ”)**

contar\_coincidencias = cadena.count("i")

print (contar\_coincidencias)  #devuelve la cantidad de coincidencias

Saber cuántos caracteres tiene una cadena **len(variable)**

Len() es una función, no un método

Lo usamos de la forma:

#Contamos cuantos caracteres tiene una cadena

contar\_caracteres = len(cadena)

print (contar\_caracteres)

Saber si una cadena empieza con… **.startswith(“ ”)**

Nos devolverá True o False, ya que verifica si lo que ingresamos dentro del método se corresponde con la cadena

empieza\_con = cadena.startswith("E")

print (empieza\_con)

Saber si una cadena termina con… **.endswith(“ ”)**

También nos devolverá True o False

termina\_con = cadena.endswith("no")

print (termina\_con)

Reemplazar una parte de una cadena por otra **.replace(“ ”, “ ”)**

Se ingresan dos parámetros, el primero es la parte de la cadena que deseamos reemplazar y el segundo parámetro es lo que queremos ingresar a la cadena

cadena\_nueva = cadena.replace("liano","to")

print (cadena\_nueva)

En caso de no encontrar coincidencias, nos devolverá la cadena anterior

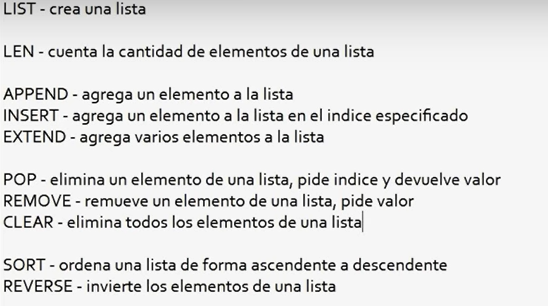
Separar cadenas mediante el string o delimitador que queramos **.split(“ ”)**

**Nos devuelve una lista con la cadena separada mediante el delimitador usado**, en caso de que la cadena no posea el delimitador o string ingresado nos devolverá la cadena original dentro de la lista

cadena\_separada = cadena.split(",")

print (cadena\_separada)

***List Methods (Métodos de Listas)***



Crear una lista con función **List()**

No es lo más común al momento de hacer listas pero se puede utilizar

lista = list(["Emiliano","Schonhals",19])

print (lista)

Saber la cantidad de elementos que posee una lista **Len()**

En cadenas, len() se usa para saber la cantidad de caracteres que tiene la misma, **pero en listas se usa para saber la cantidad de ELEMENTOS que posee la lista**

cantidad\_elementos = len(lista)

print (cantidad\_elementos)

Agregar elementos a la lista

Tenemos 3 formas de hacerlo:

**.append(“ ”)** #nos agrega el elemento en la última posición

lista.append("Soy de boca")

print (lista)

**.insert(índice , “ ”)** #nos agrega el elemento en la posición elegida

lista.insert(1, "Axel")

print (lista)

**.extend([“ ”])** #agrega todos los elementos que queramos a la lista

lista.extend(["Boca Juniors", 44702037])

print (lista)

Eliminar elementos de la lista

Tenemos 3 formas de hacerlo:

**.pop(índice)** #elimina el elemento que esté en ese índice

lista.pop(0)

print (lista)

Si en el índice ponemos -1 eliminará el ÚLTIMO elemento de la lista, si en el índice ponemos -2 nos eliminará el ANTEÚLTIMO elemento de la lista, y así sucesivamente

**.remove(“ ”)** #Elimina un elemento de la lista por su valor, si lo encuentra lo elimina

lista.remove("Boca Juniors")

print (lista)

**.clear()** #Elimina TODOS los elementos de la lista

lista.clear()

print (lista)

Ordenar elementos de la lista de forma ascendente (**NO FUNCIONA CON STRINGS**)

#de forma descendente:

lista.sort()

print (lista)

#de forma ascendente:

lista.sort(reverse=True)

print (lista)

Invertir los elementos de una lista

**.reverse()**

lista.reverse()

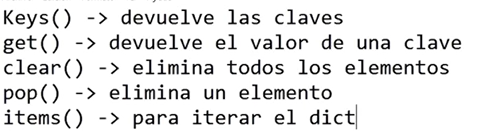
print (lista)

Verificar si un elemento se encuentra en la lista

**.index(“ ”)** #Nos devolverá el índice donde se encuentra el elemento, caso contrario nos dirá que el elemento no pertenece a la lista

elemento\_encontrado =lista.index("Emiliano")

print (elemento\_encontrado)

***Dict Methods (Métodos de Diccionario)***

diccionario = {

    "nombre" : 'Emiliano',

    "apellido" : 'Schonhals',

    "edad" : 19

}

**.keys()** #Nos devuelve las claves en forma de TUPLAS, tmb sirve para iterar

claves = diccionario.keys()

print (claves)

**.get(“ ”)** #Nos devuelve el valor que le solicitemos

devuelve\_valor = diccionario.get("nombre")

print (devuelve\_valor)

**.clear()** #Elimina todos los elementos del diccionario

diccionario.clear()

print (diccionario)

**.pop(“ ”)** #Elimina el valor que deseemos

diccionario.pop("nombre")

print (diccionario)

**.items()** #Obtenemos un elemento dict\_items iterable

diccionario\_iterable = diccionario.items()  #devuelve dict\_items iterables

print (diccionario\_iterable)

***Data Inputs (Entrada de Datos)***

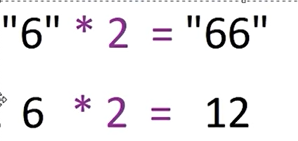
Se usa el **variable = input(“ ”)**

nombre = input("Ingrese su nombre: ")

apellido = input("Ingrese su apellido: ")

print (f"Su nombre completo es {nombre} {apellido}")

#es vital poner el Fstring para poder asi mostrar las variables

Qué pasa cuando ingresamos datos

**Aclaración:** cuando multiplicamos un string sencillamente se multiplican sus caracteres, para poder operar bien debemos pasarlos a int o float

**TODOS los datos ingresados son considerados strings**, por ende para tratarlos como números int o float (por ejemplo al hacer operaciones o demás cosas así) necesitamos pasarlos de string a int o float, por ende lo que debemos hacer es:

#Si queremos operar con int

resultado = int(variable) \* 2

print(f"El resultado es: {resultado}")

#Si queremos operar con float

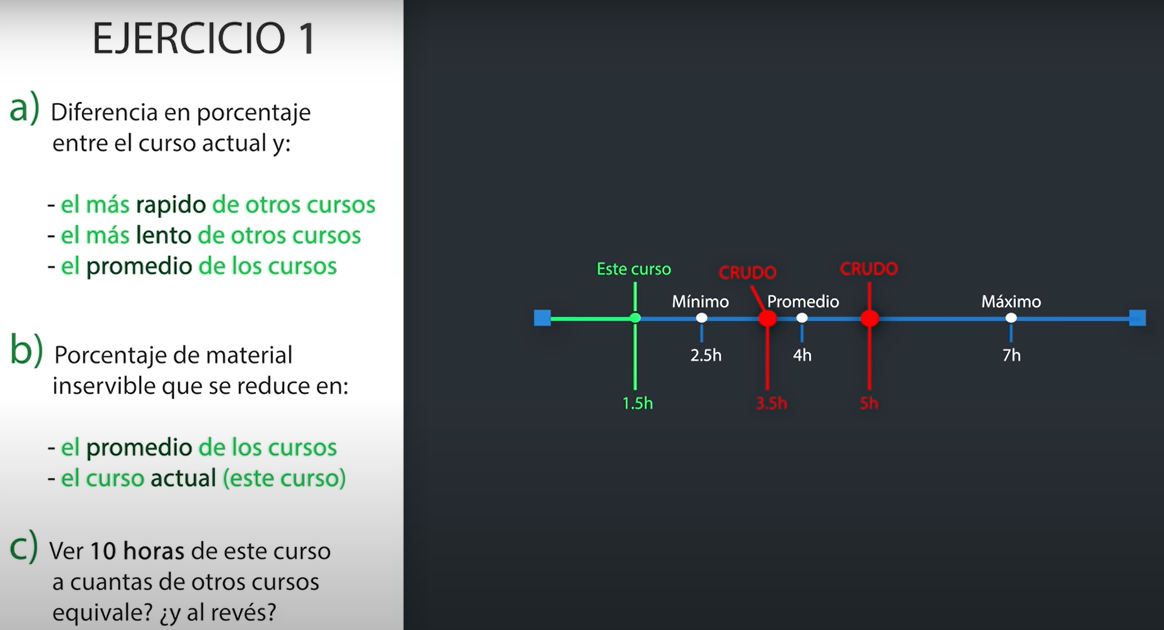
resultado = float(variable) \* 2

print(f"El resultado es: {resultado}")

#Si queremos operar con strings

resultado = variable \* 2

print(f"El resultado es: {resultado}")

***Ejercicio 1***

#PARTE A

#Promedios de duración

otros\_cursos\_min = 2.5

otros\_cursos\_max = 7

otros\_cursos\_promedio = 4

dalto\_curso = 1.5

#Diferencias de duración

#restamos 100 a los cálculos del porcentaje para saber la DIFERENCIA

diferencia\_con\_min = 100 - (dalto\_curso / otros\_cursos\_min \* 100)

diferencia\_con\_max = 100 - (dalto\_curso \* 1000 // otros\_cursos\_max / 10) #------> en este caso usamos esta fórmula para que el resultado tenga solo un decimal

diferencia\_con\_promedio = 100 - (dalto\_curso / otros\_cursos\_promedio \* 100)

print (f"El curso de Dalto dura un {diferencia\_con\_min}% menos que el mas rápido")

print (f"El curso de Dalto dura un {diferencia\_con\_max}% menos que el mas lento")

print (f"El curso de Dalto dura un {diferencia\_con\_promedio}% menos que el promedio")

*Observaciones:*

**Para saber la diferencia el cálculo es: 100 – (cálculo de porcentaje)**

**Para que el resultado no tenga muchos decimales:**

(EN ESTE CASO)

diferencia\_con\_max = 100 - (dalto\_curso \* 1000 // otros\_cursos\_max / 10)

#PARTE B

#Calculando el porcentaje de tiempo vacío removido

crudo\_promedio = 5

crudo\_dalto= 3.5

tiempo\_vacio\_promedio = 100 - (otros\_cursos\_promedio \* 1000 // crudo\_promedio / 10)

tiempo\_vacio\_dalto = 100 - (dalto\_curso \* 1000 // crudo\_dalto / 10)

print(f"Un curso promedio elimina un {tiempo\_vacio\_promedio}% de tiempo vacío")

print(f"El curso de Dalto elimina un {tiempo\_vacio\_dalto}% de tiempo vacío")

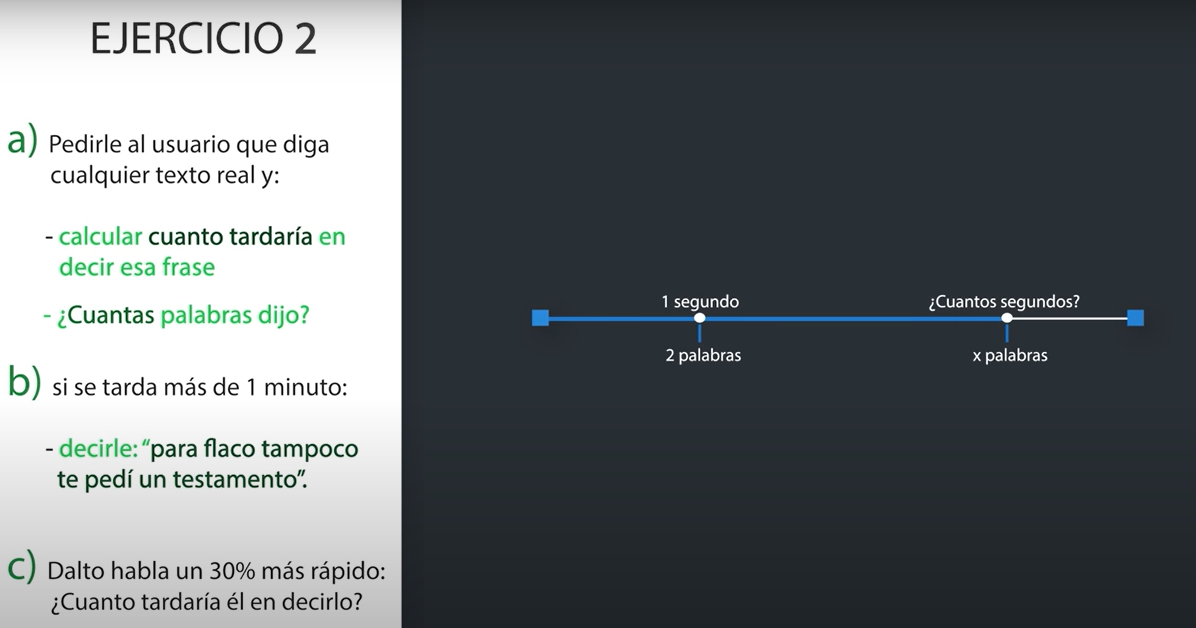
#PARTE C

#Mostrando diferencias si los cursos duraran 10 horas

print (f"Ver 10 horas de este curso equivale a ver {otros\_cursos\_promedio \* 100 // dalto\_curso / 10} horas de otros cursos")

print (f"Ver 10 horas de otros cursos equivale a ver {dalto\_curso \* 100 // otros\_cursos\_promedio / 10} horas del curso de Dalto")

***Ejercicio 2***



frase = input("Ingrese la frase y calcularemos cuánto tardarías en decirla: ")

palabras\_separadas = frase.split(" ") #separamos las palabras mediante el delimitador (en este caso el espacio en blanco)

cantidad\_de\_palabras = len(palabras\_separadas) #con la función len() conocemos la cantidad de palabras del string

segundos\_normal = cantidad\_de\_palabras / 2

#suponiendo q una persona normal diga dos palabras por segundo

print (f"Dijiste {cantidad\_de\_palabras} palabras, y tardarías {segundos\_normal} segundos en decirlo")

#Si demora mas de 1 minuto con su frase

if (cantidad\_de\_palabras > 120) :

print ("Para flaco tampoco te pedí un testamento")

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

segundos\_dalto = segundos\_normal – ((cantidad\_de\_palabras / 2) \* 0.3)

print (f"Dalto lo diría en {segundos\_dalto} segundos")

***Variables 2.0***

**Desempaquetado de variables**

Cada variable va a tomar su elemento correspondiente del array, por eso **siempre la cantidad de variables debe ser igual a la cantidad de elementos del array**

#en este caso usamos una lista, tmb podemos usar tuplas o conjuntos

datos = ["Emiliano", "Schonhals", 19]

#desempaquetado (SIEMPRE DEBE HABER LA MISMA CANTIDAD DE VARIABLES QUE DE ELEMENTOS EN EL ARRAY)

nombre,apellido,edad = datos

#cada variable ya tomo el valor correspondiente, por ende ya podemos mostrar la que queramos

print (f"El nombre completo es {nombre} {apellido} y tiene {edad} años")

**Formas de crear Tuplas:**

**Las tuplas debemos crearlas cuando son datos de SOLO LECTURA, ya que la tupla es inmutable y no puede modificarse**

Forma normal

tupla = ("Dato 1", "Dato 2")

Función Tuple() (Convierte una lista en una tupla)

#creando tuplas con tuple() (SOLO RECIBE COMO PARÁMETRO UNA LISTA)

tupla\_funcion = tuple(["Dato 1", "Dato 2"]) #le pasamos una lista y la transforma en tupla

Sin paréntesis de múltiples datos

#creando una tupla sin paréntesis de múltiples datos

tupla\_multiple = "Dato 1","Dato2"

Sin paréntesis de un solo dato

#creando una tupla sin paréntesis de un solo dato

tupla\_sola = "Dato 1",  #si es un solo dato debemos poner la coma (,) al final ya que sino lo tomara como un string común y corriente

**Formas de crear conjuntos:**

Forma normal

conjunto = {"Dato 1", "Dato 2"}

Función Set()

#Función Set()

conjunto\_set = set("Dato 1", "Dato 2")

Metiendo un conjunto dentro de otro conjunto

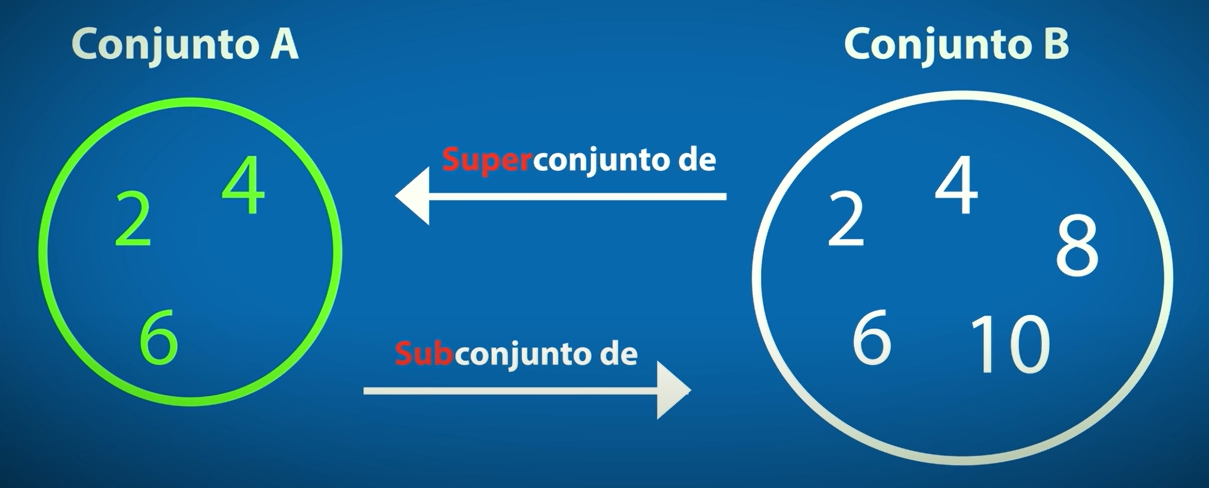
#Metiendo un conjunto dentro de otro conjunto

conjunto1 = frozenset(["Dato 1", "Dato 2"])  #crea un conjunto que puede ser congelado para que de esa forma sea hasheable (puede ser modificado)

conjunto2 = {conjunto1, "Dato 3"}

print (conjunto2)

Teoría de Conjuntos



Saber si un conjunto es subconjunto de otro **.issubset()**

#Saber si un conjunto es subconjunto de otro

conjunto1 = {1,2,3,4,5}

conjunto2 = {2,4}

resultado = conjunto2.issubset(conjunto1) #El método .issubset() nos permite saber si un conjunto es subconjunto de otro

#Nos devuelve True o False dependiendo si es subconjunto o no

print(resultado)

Otra forma de saber si es un subconjunto

resultado2 = conjunto2 <= conjunto1

print (resultado2)

Saber si un conjunto es superconjunto de otro **.issuperset()**

#Saber si un conjunto es superconjunto de otro

resultado3 = conjunto2.issuperset(conjunto1) #El método .issuperset() nos permite saber si un conjunto es superconjunto de otro

#Nos devuelve True o False dependiendo si es superconjunto o no

print (resultado3)

Otra forma de saber si es un superconjunto

resultado4 = conjunto2 > conjunto1

print (resultado4)

Saber si hay algún elemento en común **.isdisjoint()**

resultado5 = conjunto2.isdisjoint(conjunto1)

#El método .isdisjoint() sólo nos devolverá True si no hay ningún elemento que sea igual en ambos conjuntos, con que solo haya uno igual nos devolverá False

print (resultado5)

**Formas de crear Diccionarios:**

Forma normal

diccionario = {

    'nombre'  "Emiliano",

    'Apellido' : "Schonhals" ,

    'edad' : 19

}

Función Dict()

diccionario2 = dict(nombre="Emiliano", apellido="Schonhals", edad=19)

#Nos permite crear diccionarios vacíos

**#La única forma de crear diccionarios vacíos es con la función dict, lo mismo pasa con las listas y tuplas, sólo con sus funciones respectivas podemos crear conjuntos, listas o tuplas vacías**

Crear diccionarios con todos valores indefinidos

diccionario3 = dict.fromkeys(["nombre", "apellido", "edad"])

Crear diccionarios en los que cada elemento tenga el valor que deseemos

diccionario4 = dict.fromkeys(["nombre", "apellido", "edad"], "Bokita papá")

Crear diccionarios en los que cada caracter tenga valores indefinidos

diccionario4 = dict.fromkeys("ABCDE")

Crear diccionarios en los que a cada caracter le demos un valor elegido

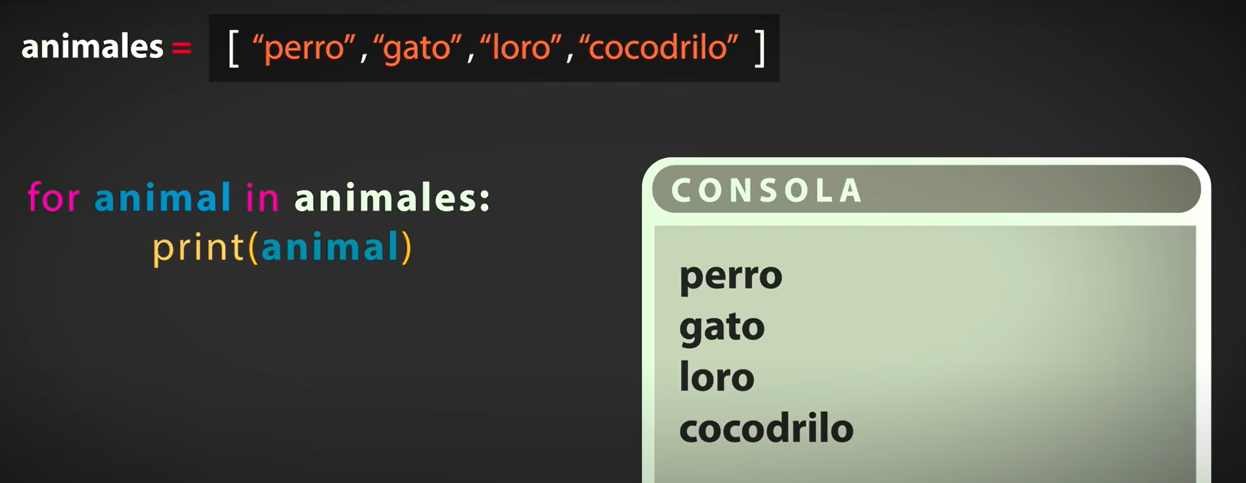
diccionario5 = dict.fromkeys("ABCDE", 5)

**#El primer parámetro es el iterable y el segundo es el valor que deseamos que tomen todos los caracteres**

***BUCLES***

**Sirven para repetir de forma controlada la ejecución de un código**

Nos permiten iterar elementos como listas, tuplas, conjuntos, diccionarios, cadenas de texto

**For Loop (Bucle For)**

**Se crea la lista, luego se ejecuta el bucle tantas veces como elementos tenga y cuándo no tenga más elementos para iterar el programa continuará con su flujo natural**

animales = ["perro", "gato", "loro", "cocodrilo"]

for animal in animales :         #FOR se ejecuta tantas veces como elementos tenga

    print (animal)

Ej 2:

#Recorriendo la lista números y multiplicando sus elementos por 10

numeros = [2, 4, 6, 8, 10]

for num in numeros :

    resultado = num \* 10

    print (f"{num} \* 10 es {resultado}")

Para iterar dos o más listas juntas **zip(**lista1 **,** lista2**)**

**TODAS las listas deben tener la misma cantidad de elementos**

for animal,num in zip(animales,numeros) :

    print (f"Recorriendo lista 1: {animal}")

    print (f"Recorriendo lista 2: {num}")

Recorrer una lista por su índice **enumerate ()**

**La función enumerate() genera una tupla de dos elementos, en el primero guarda los índices y en el otro guarda los valores**

for nro in enumerate(numeros) :

#La función ENUMERATE() nos genera una tupla de dos elementos

    indice = nro[0]    #en el parámetro 0 se guardan los índices

    valor = nro[1]     #en el parámetro 1 se guradan los valores

    print (f"El índice {indice} contiene el valor {valor}")

Uso del ELSE

**Los else se ejecutarán siempre al final del bucle, a menos que usemos un break**

for nro in numeros :

    print (f"los números son: {nro}")

else: print ("El bucle se terminó")

Iterar *Diccionarios* (obteniendo índice y valor en una tupla) **.items()**

diccionario = {

    'nombre' : "Emiliano",

    'apellido' : "Schonhals",

    'altura' : 1.87,

    'peso' : 87

}

for datos in diccionario.items() :

    key = datos[0]

    value = datos[1]

    print (f"El índice es {key} y el dato es {value}")

**Iterar salteando lo que no queremos**

frutas = ["banana", "manzana", "naranja", "sandía"]

for fruta in frutas :

    #supongamos que no me gusta la manzana y deseo que no se muestre en pantalla al #iterarse, deseo saltearla

    if (fruta == "manzana") :

        continue  #hace que salteemos el elemento, continuando así el programa

    print (f"Me gusta la {fruta}")

Iterar una cadena

#al iterar una cadena, nos devolverá caracter por caracter

cadena = "Hola que tal?"

for caracter in cadena :

    print (caracter)

**FOR en una sola línea de código**

lista = [2, 4, 6, 8, 10]

#multiplicamos por 2 sus elementos y los mostramos en UNA SOLA LÍNEA DE CÓDIGO

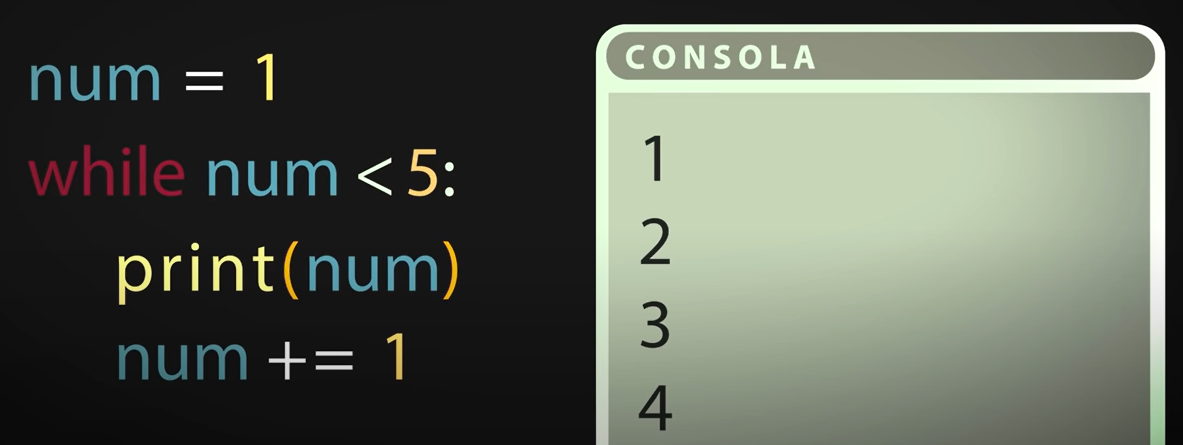
resultado = [numero\*2 for numero in lista]

print (resultado)

#le restamos 2 a cada elemento y los mostramos en UNA SOLA LÍNEA DE CÓDIGO

resultado\_resta = [numero-2 for numero in lista]

print (resultado\_resta)

**While Loop (Bucle While)**

contador = 0

while (contador < 5) : #mientras que la condición se cumpla, el bucle while se ejecutará

    contador += 1  #le vamos sumando 1 al contador

    print (contador)

print ("El while llegó a su fin")

**Built-In Functions (Funciones Integradas)**

Definición de **función**

Una función es un **fragmento de código que podemos reutilizar** en cualquier otro momento sin necesidad de volver a escribir todo el código

Nos permite separar el programa en partes más pequeñas y manejables, permitiéndonos probar la parte del código que deseemos y modificarlo sin necesidad de correr todo el programa. También es más legible

Encontrar el número **max()** y **min()**

numeros = [2, 29, 5, 6, 11, 30, 4]

#Función para encontrar el número mas grande

numero\_mayor = max(numeros)

print (f"El número mas alto es el {numero\_mayor}")

#Función para encontrar el número mas chico

numero\_menor = min(numeros)

print (f"El número mas bajo es el {numero\_menor}")

Redondear a la cantidad de decimales que deseemos **round(lista , cant de decimales)**

#Función para redondear a la cantidad de decimales que deseemos

redondear = round(3.14120394 , 2)  #en el 1er parámetro ponemos el número y en el 2do la cantidad de decimales que queremos que tenga

print (f"El número redondeado es {redondear}")

Función Bool

#Función Bool   nos devuelve False ---> 0, vacío, False, none

#               nos devuelve True ----> distinto a 0, datos no vacíos, True, cadenas

resultado\_bool\_1 = bool(["Jorge", 67])

print (resultado\_bool\_1)

resultado\_bool\_2 = bool(0)

print (resultado\_bool\_2)

El primer resultado devolverá True y el segundo False en este ejemplo

Saber si todos los valores son verdaderos **all([iterable])**

#Función All    nos devuelve True si todos los valores son verdaderos

#comprueba todos los valores que están dentro de un iterable (lista, tupla, conjunto, etc)

resultado\_all = all(["Aaron", 19, 1.87])

print (resultado\_all)  #nos devuelve True ya que no hay un elemento que se falso

resultado\_all\_2 = all([0, None])

print (resultado\_all\_2) #nos devuelve False ya que existen elementos falsos

Sumar todos los valores de un iterable **sum(iterable)**

numero = [2, 4, 10]

resultado\_sum = sum(numero)

print (resultado\_sum)

**Writing Functions (Creando funciones)**

Se usan **parámetros**, los cuales son variables que se crean para ser usadas dentro de la función, afuera de ella no existen

***Conflictos de nombres de variables (USO DE VARIABLES EN FUNCIONES)***

Una de las principales ventajas de las subrutinas (FUNCIONES) es que permiten reutilizar el código. Pero copiar y pegar subrutinas de un programa a otro puede producir lo que se llama un **conflicto de nombres de variables**. En efecto, si la subrutina que pegamos en un programa utiliza alguna variable auxiliar para algún cálculo intermedio y resulta que el programa ya utilizaba una variable con el mismo nombre que esa variable auxiliar, los cambios en la variable que se hagan en la subrutina podrían afectar al resto del programa de forma imprevista.

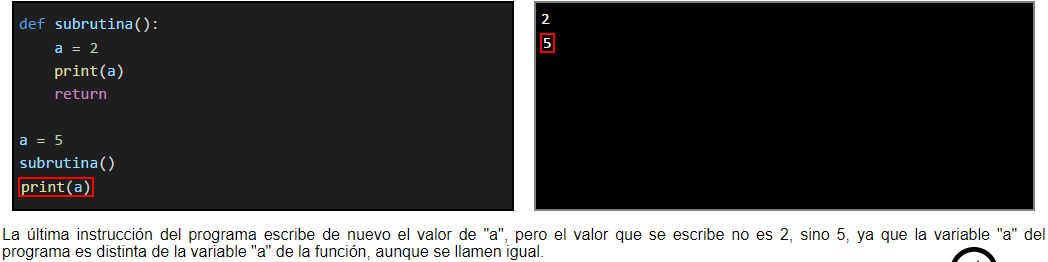
**Para resolver el problema de los conflictos de nombres, los lenguajes de programación limitan lo que se llama el alcance o el ámbito de las variables**. Es decir, que los lenguajes de programación permiten que una variable exista únicamente en el interior de una subrutina y no afecte a otras variables de mismo nombre situadas fuera de esa subrutina. Como las subrutinas pueden contener a su vez otras subrutinas, se suele hablar de niveles: **el nivel más alto sería el programa principal, el siguiente nivel serían las subrutinas incluidas en el programa principal y cada vez que hay una subrutina incluida dentro de otra estaríamos bajando un nivel**.

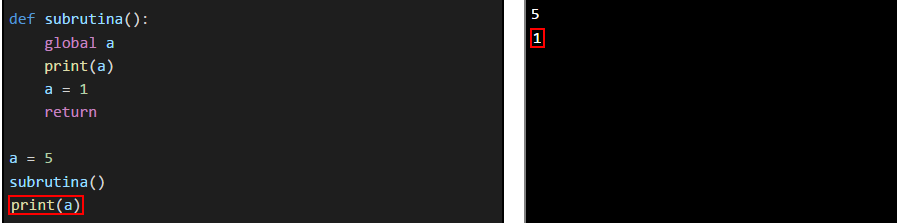
El problema es más complicado de lo que parece a primera vista, porque a menudo también nos interesará que una subrutina pueda modificar variables que estén definidas en otros puntos del programa. Así que los lenguajes de programación tienen que establecer mecanismos para aislar las variables y evitar los conflictos de nombres, pero al mismo tiempo deben permitir el acceso a las variables en los casos que así lo quiera el programador.

Aunque cada lenguaje tiene sus particularidades, el mecanismo más habitual se basa en los siguientes principios:

* cada variable pertenece a un ámbito determinado: al programa principal o a una subrutina.
* las variables son completamente inaccesibles en los ámbitos superiores al ámbito al que pertenecen
* las variables pueden ser accesibles o no en ámbitos inferiores al ámbito al que pertenecen (el lenguaje puede permitir al programador elegir o no esa accesibilidad)
* en cada subrutina, las variables que se utilizan pueden ser entonces:
  + variables locales: las que pertenecen al ámbito de la función y no pueden ser usadas en el programa principal a menos que las declaremos como global (y que pueden ser accesibles a niveles inferiores)
  + variables libres: las que pertenecen a ámbitos superiores pero son accesibles en la función

Si no se han declarado como globales o no locales, las variables **a las que se asigna valor** en una función se consideran variables **locales**, es decir, sólo existen en la propia función



Si queremos asignar valor a una variable en una subrutina, pero no queremos que Python la considere local, debemos declararla en la función como global o nonlocal

Forma Simple

def saludar() :

    print ("Hola máquina, como andas?")

#llamando a la función creada

saludar()

Creando una función con parámetros

def saludo(nombre,sexo) :

    sexo = sexo.lower() #pasamos sexo a minúsculas para su uso

    if (sexo == "mujer") :

        adjetivo = "reina"

        print (f"Hola {nombre}, mi {adjetivo}, como andas?")

    elif (sexo == "hombre") :

        adjetivo = "campeón"

        print (f"Hola {nombre}, mi {adjetivo}, como andas?")

#llamando a la función dependiendo lo que le pidamos

saludo("Emi","HOMBRE") #no importa como escribimos el sexo por que lo transformamos en minúsculas para su lectura

saludo("Brisa","MUjer")

Creando una función que nos devuelva valores **RETURN**

def sumando(num1,num2) :

    suma = num1 + num2

    return suma

#es necesario que una variable tome el valor retornado

resultado = sumando(4,10)

print (f"El resultado de la suma es: {resultado}")

Creando una función que retorne varios valores

def sumando(num1,num2) :

    suma = num1 + num2

    return (suma,num1,num2)

#en este caso nos devuelve una tupla que podemos desempaquetar

resultado,nro1,nro2 = sumando(3,7)

print (f"Sumando los números {nro1} y {nro2} llegamos al {resultado}")

Parámetro \*Args

def sumas (\*numeros) : #con el asterisco definimos el parámetro como una lista, por lo tanto convierte todos los parámetros que le pasemos a un sólo parámetro

    numeros\_sumados = 0

    resultado = sum(numeros) #función que suma todos los valores

    return (resultado)

calculo = sumas(2,4,6,8)

print (calculo)

En este caso nos devolverá 20, que es la suma de todos los números ingresados

#si usamos el parámetro args junto con otros parámetros que no lo sean, siempre debemos ponerlo al final ya que sino no podremos agregar otro parámetro

def sumas (nombre,\*numeros) : #con el asterisco definimos el parámetro como una lista, por lo tanto convierte todos los parámetros que le pasemos a un sólo parámetro

    numeros\_sumados = 0

    resultado = print (f"{nombre} la suma de tus números es igual a {sum(numeros)}")

    return (resultado)

calculo = sumas("Emiliano",2,4,6,8)

print (calculo)

En este caso nos devolverá “Emiliano la suma de tus números es igual a 20”

Permite pasar un número variable de argumentos a una función, por lo que si queremos definir una función cuyo número de parámetros de entrada puede ser variable, se usa \*args

También podemos usar \*\*kwargs (keyword arguments)

\*\*kwargs permite pasar argumentos de longitud variable asociados con un nombre o **key** a una función. Debemos usar \*\*kwargs si queremos manejar argumentos con nombre como entrada a una función

Es decir, dentro de la función no solo tenemos acceso a la variable como con \*args, sino que también tenemos acceso a un nombre o key asociado

**Funciones Lambda**

A diferencia de las funciones convencionales, Lambda crea funciones anónimas que luego podemos almacenarlas en variables

Beneficios

Podemos usarlas cuando queremos crear algo sencillo y rápido

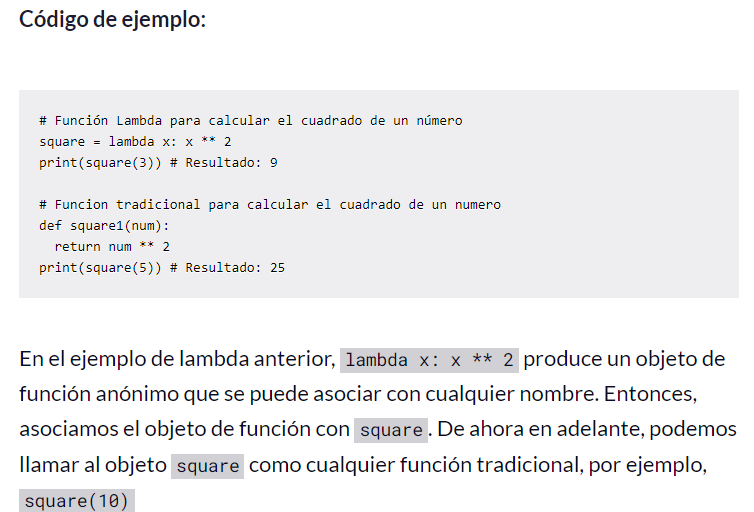
Retornan automáticamente, sin necesidad de que escribamos un Return

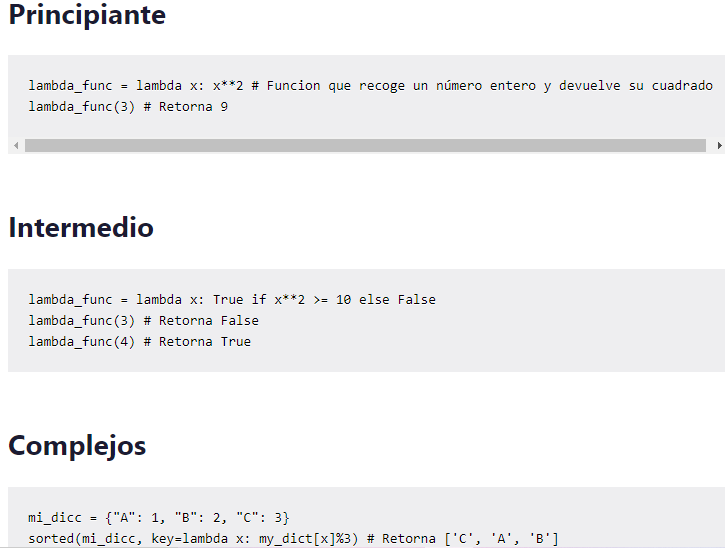
Contras

No son aptas cuando necesitamos ejecutar más de una expresión

**Sintaxis de una función Lambda**

**lambda argumentos : expresión**

Las funciones Lambda pueden tener cualquier número de argumentos, pero solo una expresión



***EJERCICIO 2.1***

1. Pedir nombre y edad de los compañeros que fueron a clase y ordenar los datos de menor a mayor
2. El mayor es el profesor y el menor es el asistente, ¿quién es quién?

def obtener\_compañeros (cant\_compañeros) :

    compañeros = []

    for i in range(cant\_compañeros) :

        nombre = input("Ingrese el nombre: ")

        edad = int(input("Ingrese la edad: "))

        #hacemos que persona sea una tupla y luego la agregamos dentro de la lista principal con .append para dsp ordenarla por edad

        persona = (nombre, edad)

        compañeros.append(persona) #.append agrega elementos a la lista, uno atrás de otro

    #ordenando por edad

    compañeros.sort(key = lambda x: x[1]) #.sort crea un iterable que ordena la key que le pongamos, en este caso el índice 1 de compañeros, osea por la edad

#como tiene tuplas dentro una lista, el primer índice es la tupla y el segundo es el elemento que queremos de esa tupla

    asistente = compañeros[0][0] #primer tupla, primer elemento de ella (ya que la ordenamos de menor a mayor)

    profesor = compañeros[-1][0] #última tupla, primer elemento de ella

    #retornamos los valores que necesitamos, en este caso una tupla

    return (asistente, profesor)

#INICIO DEL PROGRAMA

numero = int(input("Ingrese la cantidad de compañeros que fueron a la clase: "))

#llamamos a la función y desempaquetamos los valores de la tupla

asistente,profesor = obtener\_compañeros(numero)

#printeamos los resultados

print(f"El profesor es {profesor} y su asistente es {asistente}")

***EJERCICIO 2.2***

Crear una función que nos devuelva los números primos entre 0 y el valor que ingresemos

#creamos la función que verifica qué números son primos

def es\_primo (num) :

    #verificamos que el número anterior no pueda dividirse por ningún número entre el 2 y el mismo número

    for i in range(2 , num-1) :

        #si es divisible por alguno retornamos False

        if (num%i==0) :

            return False

    #si no fue divisible retornamos True

    return True

#creamos otra función para que cree una lista con todos los números primos

def primos\_hasta(num) :

    #creamos la lista

    primos = []

    for i in range (2, num+1) :

        #verificamos si el valor es primo

        resultado = es\_primo(i)

        #en caso de que sea primo lo agregamos a la lista

        if (resultado == True) :

            primos.append(i)

    #devolvemos la lista

    return primos

#INICIO DEL PROGRAMA

print("Se mostrarán todos los números primos entre el 2 y el número que ingrese")

numero = int(input("Ingrese el número: "))

#llamada a la función

resultado = primos\_hasta(numero)

#printeo final

print (resultado)

***Modules (Módulos)***

**Un módulo es cualquier archivo con la extensión .py**

**Un módulo es una porción de un**[**programa**](https://es.wikipedia.org/wiki/Programa_(computaci%C3%B3n))**. De las varias tareas que debe realizar un programa para cumplir con su función, un módulo debe realizar una tarea puntual, o varias en algunos casos**

Desde un módulo podemos llamar o crear otros módulos, usando lo que hay en ellos para el programa que estemos haciendo

Existen **3 tipos de Módulos**

***Python modules:*** Son los módulos que ya vienen incorporados en Python

***Módulos de Terceros:*** Módulos de otras personas que podemos usar

***Módulos Propios:*** Módulos creados por nosotros

**Acceder a un módulo desde otro módulo**

Debemos importarlo

Para eso usamos el **import** **módulo**

#Creamos una función que luego llamaremos en el siguiente módulo

def saludar (name) :

    return f"Hola {name}, como estas?"

#Usaremos la función que se encuentra en el módulo saludar

import modulo\_saludar #importamos el módulo

saludo = modulo\_saludar.saludar("Emi")

print (saludo)

**Al llamar una función de otro módulo debemos darle a una variable el valor del :**

Nombre\_del\_módulo**.**nombre\_de\_la\_función(valor\_que\_ingresemos)

La forma en la que nos referimos a todas las cosas que están dentro del módulo importado es **NameSpace**

Cada vez que lo ejecutamos nos crea una carpeta llamada pycaché que básicamente sirve para que el proceso de ejecución sea más rápido, no es algo de qué preocuparse

***Operador AS***

Nos permite, en el caso que lo deseemos, **darle otro nombre más simple al módulo** al que queremos acceder para poder usar ese nuevo nombre cuando nos refiramos a él

Nos ayuda sobre todo con módulos cuyo nombre es muy tedioso y largo

import modulo\_saludar as modulo

nombre = input("Ingrese su nombre: ")

#Al llamar una función de otro módulo debemos darle a una variable el valor del :

# Nombre\_del\_módulo.nombre\_de\_la\_función(valor\_que\_ingresemos)

saludo = modulo.saludar(nombre)

print(saludo)

**También podemos renombrar funciones**

from modulo\_saludar import saludar as hola,bardear as manco

***Cómo importar solo las funciones que necesitemos y no el módulo completo (ASI DEBEMOS HACERLO SIEMPRE)***

Usamos el **FROM**

Nos ayuda para **no traer al programa todo el módulo de forma innecesaria** si sólo necesitáramos algunas funciones u objetos que estén definidos dentro del módulo

#en este caso llamaremos a dos funciones que están dentro de esta carpeta

#Desde el modulo\_saludar importamos las funciones saludar y bardear

from modulo\_saludar import saludar,bardear

saludo = saludar("Emi")

bardo = bardear("Marcos")

print(saludo)

print(bardo)

Con este método, **al momento de llamar las funciones simplemente hacemos que una variable tome el valor de la función y entre paréntesis el parámetro elegido**

**Enrutamiento de Módulos**

Básicamente nos ayuda a movernos entre carpetas, a saber cómo llamar desde un archivo a otro, ver donde están todos los módulos que ya existen de Python, etc.

**Si el módulo está en otra carpeta pero de la misma ruta:**

import nombre\_carpeta.modulo as modulo

**Si el módulo no está en la misma ruta que el módulo que estamos usando:**

Primero debemos saber **dónde están alojados los módulos**

Para eso usamos el **print(sys.path)**, el cual **nos mostrará una lista con la ruta de los módulos**

La primera ruta que nos devuelve es la ruta del módulo actual, el que estamos usando

Las demás rutas que nos aparecen son todos los módulos instalados

**Si queremos instalar un módulo que no esté en esa lista** para poder usar sus funciones necesitamos agregarlo manualmente usando **sys.path.append (“**ruta del módulo”**)**

Una vez que agreguemos la ruta del módulo que queremos usar **ya podemos importarlo** y seguir con el resto del código

Importante

Con **print (sys.builtin\_module\_names)** sabremos cuáles son los **módulos Built-in de Python** **(módulos integrados)**, esto es importantes ya que **Python le da prioridad a sus módulos integrados sobre los módulos que creemos**, por ende **si creamos un módulo con el mismo nombre de un módulo integrado NO nos llamará nuestro módulo, sino que nos llamará al integrado**

***Packages (Paquetes)***

**Un paquete es una carpeta con muchos archivos .py (módulos)**

Es útil cuando queremos importar muchos módulos a la vez, de esa forma **importamos el paquete con todos los módulos que tiene dentro**, en vez de andar importando cada módulo de forma individual

**Como hacer que Python entienda que la carpeta es un paquete**

**Sí o sí debemos crear dentro del paquete un archivo.py que se llame \_\_init\_\_.py y esté vacío**

Ésta es la forma en la que Python entiende que se trata de un paquete

#accederemos al paquete

import paquete.modulo\_salud as salud

import paquete.modulo\_edad as años

peso = 86

altura = 1.87

edad = int(18)

#accedemos a la función imc del modulo\_salud

resultado = salud.imc(peso,altura)

print(resultado)

#accedemos a la función mayor\_de\_edad del modulo\_edad

resultado2 = años.mayor\_de\_edad(edad)

print(resultado2)

***TXT Files (Archivos TXT)***

**Un archivo es un contenedor de información**

**Los archivos tienen sus propios tipos de datos**.

Por ejemplo, si son **.png** es un formato de imagen, **.wav** es un formato de audio, **.mp3** es un formato de audio e imagen, etc.

Por ende **los archivos se componen de un Nombre y un Formato**

En este caso usaremos archivos txt

**Un archivo .txt es un archivo de texto plano**

**Abrir un archivo .txt**

**Archivo = Open(“nombre de la carpeta \\ nombre del archivo”, encoding = “UTF-8”)**

Si nos aparecen caracteres raros cuando leemos el archivo debemos establecer la **codificación UTF-8**(la cual está agregada en el ejemplo anterior), **la cual es una codificación universal que lee todos los caracteres**

Aclaración: **pulsar la tecla 'Alt' seguida del número 92 escrito con tu teclado numérico para que se haga el slash invertido**

**Leer un archivo .txt**

**Print( archivo.read() )**

**Cuando un archivo se lee ya no se puede volver a leer hasta que lo cerremos**, sino nos mostrará un array vacío por motivos de seguridad del archivo

**Crear un array con cada línea del archivo (Leer línea por línea)**

**Variable = archivo.readlines()**

Nos devolverá una lista con cada línea del archivo

**Aclaración:** luego de cada línea nos mostrará una **\n**, la cual significa que se termina el renglón o línea

**Leer la línea completa**

**Variable = archivo.readline()**

**Si no lee la línea completa (dejándole el paréntesis vacío) nos devolverá la cantidad de caracteres que le pongamos dentro del paréntesis**

**Aclaración:** **sirve para archivos no muy extensos**, ya que en ese caso nos consumiría toda la RAM

**Para leer varias líneas debemos poner el archivo.readline() dentro de un bucle FOR**

**Leer la cantidad de caracteres que ingresemos**

**Variable = archivo.readline(cant\_caracteres)**

**Cerrar el archivo .txt**

**Archivo.close()**

**Por más que hayamos cerrado el archivo podemos usar la información previamente almacenada**

**Es importante cerrar el archivo** para que **no se cierre de forma inesperada y pierda datos**

También es importante cerrarlo manualmente ya que **el Sistema Operativo limita la cantidad de veces que un archivo puede ser abierto**

Además **liberamos los recursos que la PC está utilizando para procesar el archivo**

**Forma ÓPTIMA de trabajar con un archivo .txt**

#FORMA ÓPTIMA DE TRABAJAR CON ARCHIVOS .TXT

with open("Archivos.txt\\formacion\_boca.txt", encoding="UTF-8") as archivo :

    #dentro ejecutamos todo lo que necesitemos

    print(archivo.read())

    #el archivo se cierra de forma automática

De esta forma **se consume menos recursos y nos ahorramos líneas de código**

Además **no necesitamos cerrar manualmente el archivo, ya que lo hace de forma automática**

Si nos aparecen caracteres raros cuando leemos el archivo debemos establecer la **codificación UTF-8**(la cual está agregada en el ejemplo anterior), **la cual es una codificación universal que lee todos los caracteres**

**Crear o Reescribir un archivo.txt (borrando datos anteriores)**

Debemos agregar **“w”** de Write **dentro del With open** **para que se nos permita reescribir un archivo existente o crearlo en el caso de que no exista**

with open("Archivos.txt\\archivo\_reescrito.txt", "w", encoding = "UTF-8") as archivo :

**Si “w” no encuentra el archivo, lo crea**

**Reescribir un archivo**

Esto hace que perdamos todos los datos guardados anteriormente en el archivo

archivo.write("Hola que tal")

**Reescribir un archivo línea por línea (usando un array o iterable)**

Es la operación contraria al readlines(), ya que **nos permite ingresar línea por línea el array que querramos escribir en el archivo.txt**

archivo.writelines(["Hola Máquina, todo bien? \n", "Yo ando bien \n", "hasta ver a Boca"])

    #Ponemos \n para indicar que hay un salto de línea

Importante: **poner el \n para indicar un salto de línea**

**Agregar líneas a un archivo.txt (Sin borrar los datos anteriores)**

En vez de “w”, debemos agregar **“a”** de Append **dentro del With open**

**No borra los datos anteriores del archivo.txt, sino que agrega lo que ingresemos al archivo**

with open("Archivos.txt\\archivo\_agregado.txt", "a", encoding = "UTF-8") as archivo :

    #podemos usar las mismas operaciones que usábamos en el "w", con la diferencia de que # al poner la "a" nos lo agregará sin eliminar todos los datos anteriores del archivo.txt

    #podemos usar un Bucle para agregar la cantidad de líneas que deseemos

archivo.write("\n") #para que nos dé un salto de línea si hay algo antes de lo agregado

   for i in range(5) :

       archivo.write(f"Línea {i+1} agregada \n")

**Podemos usar las mismas operaciones que usamos en el “w”, pero esta vez no nos borrará los datos anteriores**

***CSV Files (Archivos CSV)***

**CSV significa valores separados por comas**

**En un archivo.CSV los elementos se separan por comas y luego por salto de líneas**

Ej: **"nombre","apellido","edad"**

**"Emi","Schonhals",20**

**"José","Martí",37**

**"Elías","Gómez",29**

Programas como Excel pueden leer correctamente estos archivos

Cuando llevamos estos archivos a hojas de cálculo nos interpreta siempre la **primer fila como los nombres de las columnas**

Sin embargo, para trabajar de forma óptima con archivos CSV (más que nada para análisis de datos e IA) **en vez de trabajar con la librería CSV debemos trabajar con pandas**, ya que tiene más funcionalidades y menos limitaciones

**Importamos la librería csv**

#Importamos la librería CSV

import csv

**Abrir un archivo.CSV**

**Lo abrimos de la misma forma que abrimos los archivos.TXT**

#Abrimos los archivos .CSV de la misma forma que los archivos txt

with open("Archivos.csv\\datos.csv", encoding="utf-8") as archivo :

**Leer un archivo.CSV**

**Debemos darle a una variable el valor de lectura**

A diferencia de los TXT debemos poner:

**Variable = csv.reader(archivo)**

leer = csv.reader(archivo)

Nos devuelve un iterable, por ende podemos usar el bucle FOR para mostrarlo por pantalla

#nos devuelve un iterable, por ende lo podemos recorrer

    for filas in leer :

        print (filas)

        #nos está devolviendo listas con las filas del archivo.CSV

**Forma óptima de trabajar con archivos.CSV**

Como vimos anteriormente, la forma óptima de trabajar con archivos.CSV no es mediante la librería csv, sino mediante la **librería pandas**, ya que tiene más funcionalidades y menos limitaciones

Además es la librería más profesional para trabajar con análisis de datos e IA

Para usarla primero debemos instalarla

import pandas as pd  #pd es la sigla universal con la que se conoce a la librería pandas

#usamos la función read\_csv para leer el archivo.CSV

df = pd.read\_csv("Archivos.csv\\datos.csv")

#ponemos df (DATA FRAME), son estructuras de datos bidimensionales parecidos a una hoja de cálculo

#por ende tiene sólo dos valores: Filas y Columnas, las cuales podemos usar para mostrar datos específicos del csv

**Mostrar la columna que querramos**

#mostrando la columna nombre

print(df["nombre"])

Escribimos el nombre que tiene la columna y de esa forma llamamos a todos los elementos de esa columna

**Slicing (mostrar desde el índice inferior al superior de un string o iterable)**

**Nos permite mostrar en una cadena o iterable desde el índice que queremos que inicie hasta el índice que queremos que termine (sin incluir este último)**

Para eso usamos **print(iterable o cadena[límite inferior : límite superior])**

Ej:

lista = ["Emi", "Marcos", "Raúl", "Uli", "José"]

print(lista[0:3])

#en este caso nos muestra del índice 0 al índice 2 (ya que el límite superior no se toma)

cadena = "123456789"

print(cadena[2:5])

#en este caso nos muestra del índice 2 al índice 4 (ya que el límite superior no se toma)

Lo que se muestra en pantalla es:

['Emi', 'Marcos', 'Raúl']

345

***Ordenar por la columna que querramos del archivo.csv***

De forma **ASCENDENTE**:

Usamos el **df.sort\_values(“columna”)**

#ordenando por la columna edad de forma ASCENDENTE

df\_ordenado = df.sort\_values("edad")

print(df\_ordenado)

**IMPORTANTE: Para poder usar ésta función, NO debemos dejar ningún espacio en blanco luego de cada coma en el archivo.csv, sino nos devolverá errores de la librería pandas**

De forma **DESCENDENTE**:

Usamos el **df.sort\_values(“columna”, ascending=False)**

#ordenando por la columna edad de forma DESCENDENTE

df\_ordenado\_descendente = df.sort\_values("edad", ascending=False)

print(df\_ordenado\_descendente)

**Concatenar 2 df (dataframes)**

Usamos la función concat ([df,df2])

**df\_concatenado=pd.concat ([df, df2])**

**Acceder a las primeras filas de un dataframe**

Usamos la función head (cant\_filas)

**Primeras\_filas=df.head (3)**

#en este caso nos mostrará las primeras 3 filas del dataframe

**Acceder a las últimas filas de un dataframe**

Usamos la función tail (cant\_filas)

**Ultimas\_filas=df.tail (2)**

#en este caso nos mostrará las últimas 2 filas del dataframe

**Saber la cantidad de filas y columnas del dataframe**

Usamos el atributo shape

**Shape nos devuelve una tupla con los dos valores (cant\_filas, cant\_columnas)**

filas\_y\_columnas\_totales = df.shape

filas\_totales = filas\_y\_columnas\_totales[0]

columnas\_totales = filas\_y\_columnas\_totales[1]

**Otra forma es usar el desempaquetado**

filas\_totales,columnas\_totales = df.shape

**Acceder a un dato específico del dataframe**

Podemos usar **loc (fila, “nombre de columna”)** o **iloc (fila, columna)**

elemento\_especifico\_loc = df.loc[2,"apellido"]

#o

elemento\_especifico\_iloc = df.iloc[2,2]

**Acceder a todas las filas de una columna del dataframe**

**Usamos el Slicing para indicar que accedemos a todas las filas de esa columna**

apellidos = df.loc[:,"apellido"]

#o

apellidos = df.iloc[:,2]

**Acceder a todas las columnas de una fila del dataframe**

fila\_3 = df.loc[2,:]

#o

fila\_3 = df.iloc[2,:]  #en este caso no varía

**Mostrar todas las filas que cumplan una condición**

#fila con edad mayor que 25

mayor\_que\_25 = df.loc[df["edad"], :]

**Working with Files (Trabajando con Archivos)**

**Ingresar dos listas en un archivo.txt de forma óptima**

#Tenemos dos listas, una con nombres y otra con apellidos

#Debemos escribir los datos en un archivo de texto de forma óptima

nombres = ["Emiliano","José","Eric","Matías","Carmen"]

apellidos = ["Schonhals","Paradela","Schechtel","Abdala","García"]

#Creamos el archivo.txt con permiso de escritura

with open("Archivos\_problemas\_resueltos\\nombres\_y\_apellidos.txt", "w", encoding="utf-8") as archivo :

    archivo.writelines("LOS DATOS SON \n\n")

    for nom,ape in zip(nombres,apellidos) :

        archivo.writelines(f"Nombre: {nom} \nApellido: {ape} \n-------------------\n")

**Cambiando el tipo de datos o los valores específicos de una columna**

#COMO CAMBIAR EL TIPO DE DATO DE UNA COLUMNA

#Siempre en archivos.cv debemos usar la librería pandas

import pandas as pd

df = pd.read\_csv("Archivos\_problemas\_resueltos\datos.csv")

#Convertir a string los datos de una columna

df['edad'] = df['edad'].astype(str)

#mostrar el tipo de dato del primer elemento de la columna edad

#print(type(df['edad'][0]))

#Reemplazando los datos "Emi" por "Nahuel" en la columna nombres

df['nombre'].replace("Emi","Nahuel",inplace=True)

print(df['nombre'])

print("\n")

**Eliminando líneas vacías o líneas duplicadas del df**

#Eliminando las filas con datos vacíos

df\_no\_vacio = df.dropna()

print("Dataframe sin filas con elementos vacíos")

print(df\_no\_vacio)

print("\n")

#Eliminando las filas con datos duplicados

df\_no\_duplicado = df.drop\_duplicates()

print("DataFrame sin filas duplicadas")

print(df\_no\_duplicado)

print("\n")

FORMA ÓPTIMA

#Si queremos un df totalmente limpio de filas vacías o filas duplicadas

df = df.dropna()

df = df.drop\_duplicates()

print("DF totalmente clean")

print(df)

**CREAR UN CSV LIMPIO SIN DATOS VACÍOS NI DUPLICADOS**

#CREANDO UN CSV CON EL DF LIMPIO DE LÍNEAS VACÍAS Y DUPLICADAS

df.to\_csv("Archivos\_problemas\_resueltos\\datos\_limpios.csv")

**Working with Graphics (Trabajando con gráficos)**

Además de usar la librería **Pandas** que usamos para los archivos.csv, ahora debemos usar dos librerías más:

**Matplotlib.pyplot** y **seaborn**

Las podemos llamar plt y sns, respectivamente

**Crear *Gráfico Lineal***

**Sns.lineplot(x=”nom\_columna1”, y=”nom\_columna2”, data=df)**

**Crear *Gráfico de Barras***

**Sns.barplot(x=”nom\_columna1”, y=”nom\_columna2”, data=df)**

**Crear *Gráfico de Dispersión***

**Sns.scatterplot(x=”nom\_columna1”, y=”nom\_columna2”, data=df)**

**Crear un *Gráfico de Bigotes***

**Sns.boxplot(x=”nom\_columna1”, y=”nom\_columna2”, data=df)**

**Crear punto manual en el gráfico**

**Plt.plot(“valor\_columna1”, “valor\_columna2”, “o”)**

#la “o” indica un punto en el gráfico

**Mostrar el gráfico**

**Plt.show()**

Ej 1: **gráfico lineal**

#Creando un gráfico usando un csv con las fechas y cantidad de bostezos de ese día

import pandas as pd

import matplotlib.pyplot as plt   #es una librería para visualizar datos de forma gráfica

import seaborn as sns   #es una librería que nos permite armar gráficos estadísticos

#abriendo el csv

df = pd.read\_csv("Archivos\_problemas\_graficos\_resueltos\\bostezos.csv")

#creando el gráfico lineal

sns.lineplot(x="Fecha",y="Bostezos",data=df)

#creando un punto de forma manual, en este caso en el momento de más bostezos

plt.plot("05/03",7,"o")

#mostrando el gráfico

plt.show()

Ej 2: **gráfico de barras**

#Creando un gráfico de barras usando un csv con las fechas y cantidad de bostezos de ese día

import pandas as pd

import matplotlib.pyplot as plt   #es una librería para visualizar datos de forma gráfica

import seaborn as sns   #es una librería que nos permite armar gráficos estadísticos

#abriendo el csv

df = pd.read\_csv("Archivos\_problemas\_graficos\_resueltos\\ingresos.csv")

#creando el gráfico de barras

sns.barplot(x="Fuente",y="Ingresos",data=df)

#mostrando el gráfico

plt.show()

#obteniendo el total de ingresos

total\_ingresos = df['Ingresos'].sum()

#mostrando el total de ingresos

print(f"El total de ingresos es de: ${total\_ingresos} USD")

Ej 3: **gráfico de dispersión**

#Creando un gráfico de dispersión usando un csv

import pandas as pd

import matplotlib.pyplot as plt   #es una librería para visualizar datos de forma gráfica

import seaborn as sns   #es una librería que nos permite armar gráficos estadísticos

#abriendo el csv

df = pd.read\_csv("Archivos\_problemas\_graficos\_resueltos\\dispersion.csv")

#creando el gráfico de dispersión

sns.scatterplot(x="Tiempo",y="Dinero",data=df)

#mostrando el gráfico

plt.show()

**#los gráficos de dispersión son muy utilizados en IA, nos sirven para hallar relaciones entre dos o más variables**

#Es muy útil cuando tenemos datos csv con muchos puntos

#En el ejemplo podemos observar que a mayor tiempo invertido, más dinero generamos

Ej 4: **gráfico de bigotes**

#Creando un gráfico de bigotes usando un csv

import pandas as pd

import matplotlib.pyplot as plt   #es una librería para visualizar datos de forma gráfica

import seaborn as sns   #es una librería que nos permite armar gráficos estadísticos

#abriendo el csv

df = pd.read\_csv("Archivos\_problemas\_graficos\_resueltos\\bigotes.csv")

#creando el gráfico de bigotes

sns.boxplot(x="Categoría",y="Valor",data=df)

#mostrando el gráfico

plt.show()

**#Los gráficos de bigotes muestran la distribución de datos en cuartiles, resaltando el promedio y los valores atípicos (mayor valor, menor valor, etc)**

**Crear un QR**

Usamos la librería **qrcode**

**En data ponemos el link que deseamos que esté enlazado al QR**

Guardamos el QR como un **archivo.PNG**

import qrcode

data = "https://www.instagram.com/emischonhals/"

img = qrcode.make(data)

img.save("QR\\qr.png")

**Hacer que el usuario ingrese el link que desea asociar al QR**

import qrcode

link = input("Ingrese el link de lo que quiera enlazar al QR: ")

data = link

img = qrcode.make(data)

img.save("QR\\qr\_usuario.png")

**Breve resumen de Programación Orientada a Objetos (POO)**

Es una forma más de programar

Mediante Clases y Objetos podemos construir y diseñar cualquier software que querramos

Cada objeto es una instancia de una clase (una clase es como una plantilla que define las propiedades y los comportamientos de los objetos que se crean a partir de esas mismas clases)

Esto sirve para organizar la estructura y la codificación, además de para crear programas complejos pero a la vez flexibles

En resumen, **la POO es un estilo de programación que se centra en desarrollar software en base a Clases y objetos, permitiendo una mayor seguridad y encapsulamiento del código, también permite reutilizarlo**

Es de los paradigmas más utilizados del mundo

**Exceptions (Excepciones)**

Las Excepciones **son eventos que ocurren durante la ejecución del programa**, pero lo que hacen las excepciones es **interrumpir el flujo normal de ejecución**, haciendo que se detenga toda acción en el programa

Por ende nuestra labor como programadores es saber cómo manejar estas excepciones para poder continuar con el resto del programa de forma óptima

**Las excepciones las podremos encontrar si ejecutamos el programa en modo “depurar archivo de Python” (donde lo ejecutamos normalmente elegimos esa opción)**

def sumar\_dos() :

    #usamos el bucle while con la condición True para que no termine de ejecutarse hasta que se ingresen bien los datos

    while True :

        num1 = input("número 1: ")

        num2 = input("número 2: ")

        #ponemos try donde pudiera existir alguna excepción, en este caso por un dato mal ingresado del usuario

        try :

            resultado = int(num1) + int(num2)

            #si no ocurren excepciones salimos del bucle while

            break

        #si la excepción ocurre mostramos el texto en pantalla y se repetirá el bucle while hasta que ingrese bien los datos

        except :

            print("Te pedi un entero, volvé a ingresar los datos")

    #una vez salga del bucle while teniendo los datos correctos se devuelve el resultado

    return resultado

print("Inicio del programa")

print("Ingrese dos números para sumarlos")

#llamada y muestra de la función, en este caso no le pasamos parámetros ya que ingresamos los datos dentro de la misma

print(sumar\_dos())

**El except solo se va a ejecutar si try nos lanza una excepción**

**El break dentro del try hace que si no nos lanza ninguna excepción, salga del bucle while True**

**Siempre debemos poner el try y el except dentro de un While True para que si nos lanza una excepción no nos deje salir del bucle hasta que se ingresen los datos correctos**

**También podemos agregar un finally** de la misma forma que el try y el except, pero **esta sentencia se ejecuta si o si, sin depender si lanza una excepción o no**

***Regular Expressions (Expresiones regulares)***

Al trabajar con expresiones regulares **el objetivo es encontrar coincidencias y patrones**

Podemos devolver la coincidencia, devolver todas las coincidencias, reemplazarla por otra, usarla como separador, etc

**Las tres funciones más utilizadas en expresiones regulares son Search, FindAll y Sub**

**Search** encuentra si hay una coincidencia y la devuelve

Ej: **resultado = re.search(“coincidencia”, texto)**

**FindAll** encuentra todas las coincidencias y las devuelve en forma de arrays

Ej: **resultado =** **re.findall(“coincidencia”, texto)**

**El problema surge cuando están las mismas coincidencias pero varían las mayúsculas**

En ese caso lo arreglamos con:

**resultado =** **re.findall(“coincidencia”, texto, flags=re.IGNORECASE)**

Ignorecase ignora las mayúsculas y minúsculas

**Sub** encuentra la cadena especificada y la reemplaza por otra en el texto seleccionado

Ej: **resultado = re.sub(cadena, reemplazo, texto)**

**Principales expresiones regulares en Python**

**Aclaración importante:** **Python utiliza las expresiones regulares en mayúsculas y minúsculas para hacer todo lo contrario (al menos en las expresiones más comunes)**

Ej: \d busca dígitos numéricos y \D busca todo lo que NO sea dígito numérico

**\d: busca todos los dígitos numéricos del 0 al 9**

resultado = re.findall(r"\d", texto)

**\D: busca todo MENOS dígitos numéricos del 0 al 9**

resultado2 = re.findall(r"\D", texto)

**\w: busca todos los caracteres alfanuméricos (a-z A-Z 0-9 \_)**

resultado3 = re.findall(r"\w", texto)

**\W: busca todo MENOS caracteres alfanuméricos**

resultado4 = re.findall(r"\W", texto)

**\s: busca todos los espacios en blanco (espacios, tabs, saltos de línea)**

resultado5 = re.findall(r"\s", texto)

**\S: busca todo MENOS espacios en blanco**

resultado6 = re.findall(r"\S", texto)

**\n: busca todos los saltos de línea**

resultado7 = re.findall(r"\n", texto)

**.: Busca todo MENOS los saltos de línea**

resultado8 = re.findall(r".", texto)

**\: cancela todos los caracteres especiales (los que NO son alfanuméricos)**

#cancelando la función del punto y buscando puntos

resultado9 = re.findall(r"\.", texto)

**^: Busca el comienzo de una línea**

El acento circunflejo (o v invertida) se hace usando **Alt + 94**

**Comienzo de la primera línea (lo toma como si todo el string tuviera un solo comienzo de línea)**

#buscando si "Esta" está al principio de la primer línea

resultado11 = re.findall(r"^Esta", texto)

**Comienzo de cada línea (toma cada comienzo de línea dentro del string por separado)**

#si queremos saber si está en alguna de TODAS las líneas debemos poner flags= re.M

resultado11 = re.findall(r"^Esta", texto, flags=re.M)

**$: busca el final de una línea o de todas las líneas** (en el caso de que usemos el parámetro flags= re.M)

resultado12 = re.findall(r"adios$", texto, flags=re.M)

**{n}: busca n cantidad de coincidencias consecutivas del valor que pongamos a su izquierda**

#en este caso buscamos en el texto 3 números consecutivos

resultado13 = re.findall(r"\d{3}", texto)

**{n,m}: busca mínimo n y máximo m coincidencias**

#en este caso buscamos que haya entre 1 y 4 números juntos en el texto

resultado14 = re.findall(r"\d{1,4}", texto)

**|: Con que alguna de las dos opciones que le demos se cumpla nos la devolverá**

resultado15 = re.findall(r"\d{1,4}|Hola", texto)