**Ejecutar Python**

En la terminal debo escribir Python y ahí ya estoy usando el asistente (Shell) de python, para salir de ese, escribo exit ()

**Variable declaration:** Es el proceso de asignar un nombre a una variable y reservar espacio en memoria para almacenar datos. Se puede realizar simplemente escribiendo el nombre de la variable seguido de un signo de igual (=) y el valor que deseas asignarle.

**Log statement:** Es una declaración de registro que se utiliza para imprimir información o mensajes en la consola o en un archivo de registro. En Python, puedes usar la función **print()** para imprimir mensajes en la consola.

**Type check:** Es el proceso de verificar el tipo de dato de una variable. Puedes utilizar la función **type()** para obtener el tipo de dato de una variable.

numero = 10

print(type(numero))  # Imprime <class 'int'>, que indica que la variable 'numero' es de tipo entero (integer).

**Length check:** Se utiliza para verificar la longitud de una secuencia, como una cadena, una lista o una tupla. Puedes utilizar la función **len()** para obtener la longitud de una secuencia.

cadena = "Hola"

print(len(cadena))  # Imprime 4, que es la longitud de la cadena "Hola".

**Comment:**

* Single line: Un comentario de una sola línea se utiliza para agregar anotaciones o explicaciones en el código. Se indica utilizando el símbolo numeral (#) seguido del comentario. Por ejemplo:
* Multiline: Un comentario multilineal se utiliza para comentarios extensos o para desactivar temporalmente bloques de código. Se puede hacer utilizando triple comilla (""" o ''') al comienzo y al final del comentario.

**Data Types:**

* Primitive:

Boolean: Representa un valor de verdad, que puede ser True (verdadero) o False (falso).

Numbers: Pueden ser enteros (int) o de punto flotante (float) para representar valores numéricos.

Strings: Representan secuencias de caracteres y se pueden definir utilizando comillas simples o dobles.

* Composite:

Listas (lists): Secuencias ordenadas y mutables de elementos.

Tuplas (tuples): Secuencias ordenadas e inmutables de elementos.

Diccionarios (dictionaries): Colecciones no ordenadas de pares clave-valor.

En Python, las tuplas, las listas y los diccionarios son ***tipos de datos*** utilizados para almacenar y organizar información de diferentes formas. A continuación, se explica cada uno de ellos:

Tuplas (tuples):

* Definición: Las tuplas son secuencias ***ordenadas e inmutables*** de elementos. Se crean ***utilizando paréntesis ()*** y los elementos ***se separan por comas.***
* Uso:

Almacenar datos que no deben modificarse, como coordenadas geográficas o fechas.

Devolver múltiples valores desde una función.

Utilizar tuplas como claves en un diccionario (debido a su inmutabilidad).

* Funciones:

Inicializar(initialize): Se pueden inicializar utilizando paréntesis () o simplemente separando los elementos por comas.

Acceder a un valor(acces value): Se puede acceder a los valores de una tupla mediante su índice utilizando corchetes [].

Cambiar un valor(change value): Las tuplas son inmutables, por lo que no se pueden cambiar valores individuales en una tupla después de su creación.

Agregar un valor(add value): No se pueden agregar valores a una tupla existente, ya que son inmutables.

Eliminar un valor(delete value): No se pueden eliminar valores de una tupla existente, ya que son inmutables.

Ejemplo: coordenadas = (10, 20)

meses\_del\_ano = ("enero", "febrero", "marzo")

Listas (lists):

* Definición: Las listas son ***secuencias ordenadas y mutables de elementos***. Se crean ***utilizando corchetes []*** y los elementos ***se separan por comas.***
* Uso:

Almacenar una colección de elementos que pueden modificarse a lo largo del programa.

Iterar sobre los elementos de una lista.

Realizar operaciones como búsqueda, inserción y eliminación de elementos.

* Funciones

Inicializar: Se pueden inicializar utilizando corchetes **[]** o utilizando la función **list()**.

Acceder a un valor: Se puede acceder a los valores de una lista mediante su índice utilizando corchetes **[]**.

Cambiar un valor: Se puede cambiar un valor de una lista asignando un nuevo valor a un índice específico.

Agregar un valor: Se puede agregar un valor a una lista utilizando el método **append()** o utilizando la concatenación de listas.

Eliminar un valor: Se puede eliminar un valor de una lista utilizando el método **remove()** o la palabra clave **del**.

Ejemplo: numeros = [1, 2, 3, 4, 5]

nombres = ["Juan", "María", "Luisa"]

Diccionarios (dictionaries):

* Definición: Los diccionarios son ***colecciones no ordenadas de pares clave-valor.*** Se crean utilizando ***llaves {}*** y los ***pares clave-valor se separan por comas.***
* Características:

Mutabilidad: Los valores asociados a las claves se pueden modificar, agregar o eliminar.

No tiene orden definido: Los elementos de un diccionario no mantienen un orden específico.

Claves únicas: Las claves en un diccionario deben ser únicas.

* Uso:

Almacenar y recuperar datos utilizando claves personalizadas.

Representar estructuras de datos complejas y anidadas.

Mapear información relacionada de manera eficiente.

* Funciones:

Inicializar: Se pueden inicializar utilizando llaves {} o utilizando el constructor dict().

Acceder a un valor: Se puede acceder a los valores de un diccionario utilizando una clave específica entre corchetes [] o utilizando el método get().

Cambiar un valor: Se puede cambiar el valor asociado a una clave en un diccionario asignándole un nuevo valor.

Agregar un valor: Se puede agregar un nuevo par clave-valor a un diccionario utilizando una nueva clave y asignándole un valor.

Eliminar un valor: Se puede eliminar un par clave-valor de un diccionario utilizando la palabra clave del o el método pop().

Ejemplo: persona = {"nombre": "Juan", "edad": 30, "ciudad": "Madrid"}

estudiantes = {"A001": "María", "A002": "Pedro", "A003": "Luisa"}

**Condicional:**

* **if** (si): Es una estructura condicional que permite ejecutar un bloque de código si una condición es verdadera. Si la condición no se cumple, el bloque de código no se ejecuta

if condicion:

    # Bloque de código si la condición es verdadera

* **else** (si no): Se utiliza junto con **if** para proporcionar una alternativa en caso de que la condición sea falsa.

if condicion:

    # Bloque de código si la condición es verdadera

else:

    # Bloque de código si la condición es falsa

* **elif (sino si):** Se utiliza para evaluar condiciones adicionales después de if. Si la condición anterior no se cumple, se evalúa la siguiente condición.

if condicion1:

    # Bloque de código si la condición1 es verdadera

elif condicion2:

    # Bloque de código si la condicion2 es verdadera

else:

    # Bloque de código si ninguna condición es verdadera

**Bucle for:**

* **start** (inicio): Es el valor inicial desde el cual comenzará el bucle.
* **stop** (parada): Es el valor en el cual el bucle dejará de ejecutarse.
* **increment** (incremento): Especifica cómo cambiará la variable de control en cada iteración del bucle.
* **break**: Se utiliza para salir del bucle antes de que se complete normalmente.
* **continue**: Se utiliza para omitir el resto del bloque de código en una iteración particular y pasar a la siguiente iteración.
* **sequence** (secuencia): Es una secuencia iterable (como una lista, tupla o cadena) que se utilizará en el bucle.

for variable in sequence:

    # Bloque de código a repetir

**Bucle while:**

* **start** (inicio): Es la condición inicial o valor de inicio antes de comenzar el bucle.
* **stop** (parada): Es la condición que se evalúa en cada iteración y determina cuándo se debe detener el bucle.
* **increment** (incremento): Se utiliza para actualizar o modificar la condición en cada iteración para evitar bucles infinitos.

while condicion:

    # Bloque de código a repetir mientras la condición sea verdadera

**Función:**

* **parameter** (parámetro): Es una variable utilizada para recibir valores o información en una función. Los parámetros se definen en la declaración de la función.
* **argument** (argumento): Es el valor real o dato proporcionado a un parámetro al llamar a una función.
* **return**: Es una palabra clave utilizada para devolver un resultado o valor desde una función al lugar donde se llamó. El valor devuelto puede ser utilizado en el código posterior.

def nombre\_de\_funcion(parametro1, parametro2):

    # Bloque de código de la función

    return resultado

**Para llamar a la función y pasar los argumentos:**

nombre\_de\_funcion(argumento1, argumento2)

**Errores:**

* **NameError**: Se produce cuando intentas utilizar una variable que no ha sido definida.
* **TypeError**: Ocurre cuando se realiza una operación o función en un tipo de dato incorrecto.
* **KeyError**: Sucede cuando intentas acceder a una clave que no existe en un diccionario.
* **IndexError**: Ocurre cuando intentas acceder a un índice que está fuera del rango válido en una lista o tupla.
* **IndentationError**: Se produce cuando hay un error en la indentación del código, como una falta de sangría o un nivel incorrecto de sangría.
* **AttributeError**: Sucede cuando intentas acceder a un atributo o método que no existe para un objeto en particular.

**Tuplas:**

* Cambiar un valor: Las tuplas son inmutables, por lo que no se pueden cambiar valores individuales después de su creación.
* Agregar un valor: No se pueden agregar valores a una tupla existente, ya que son inmutables.
* Eliminar un valor: No se pueden eliminar valores de una tupla existente, ya que son inmutables.

PYTHON clase repaso general del 11/07

Recomendaciones: ejecutar print despues de cada función para ver como va funcionando mi código y detectar el código. Print es el console.log

Para crear una función se usa def nombre(parámetro):

Print(f”HOLA” {nombre}”)

Print(f”HOLA” {nombre}”)

Return 10, 12

Age1, age2 =

El bloque de código lo define la identacion