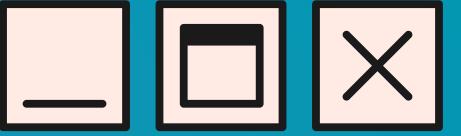




TALLER

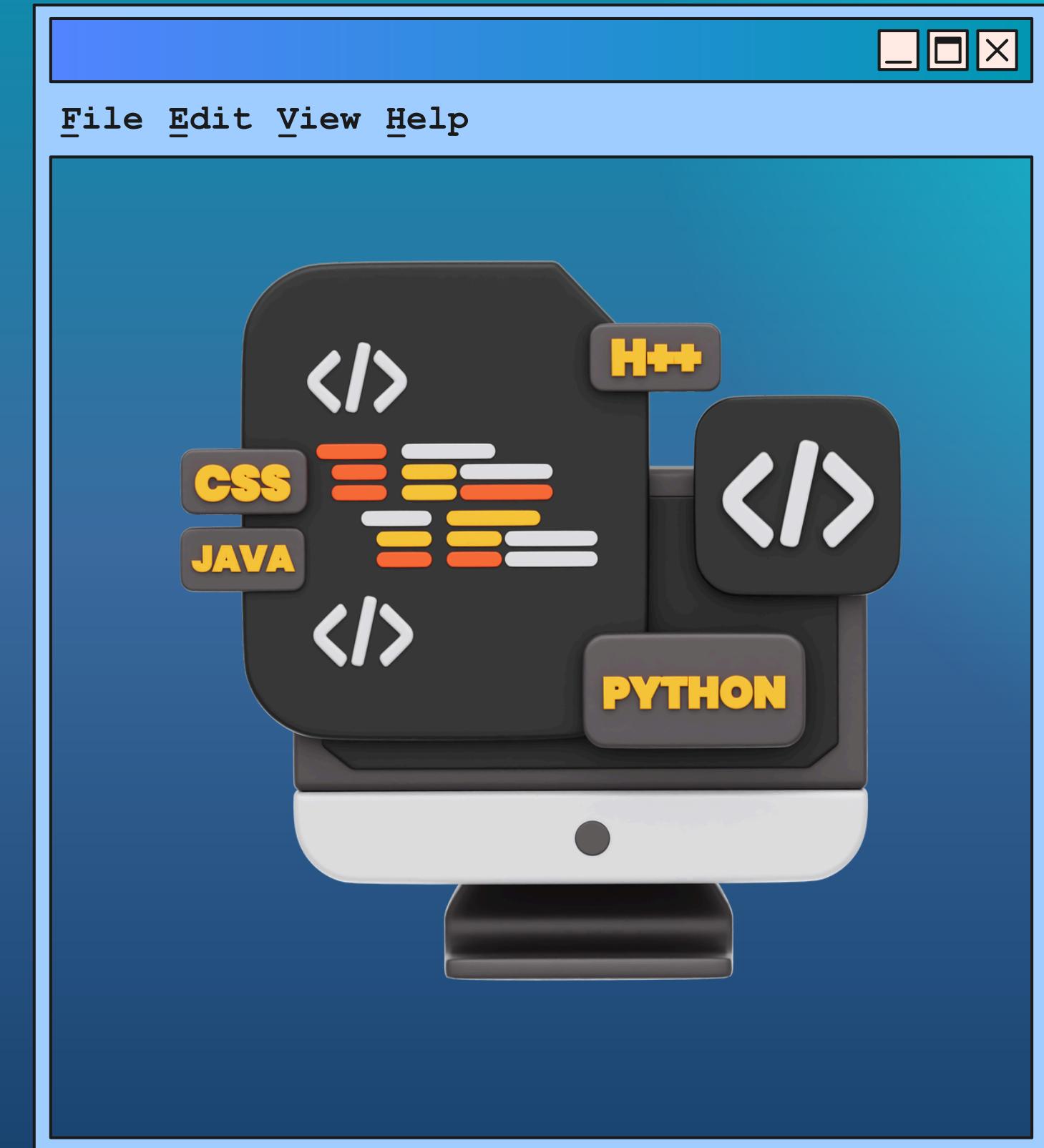
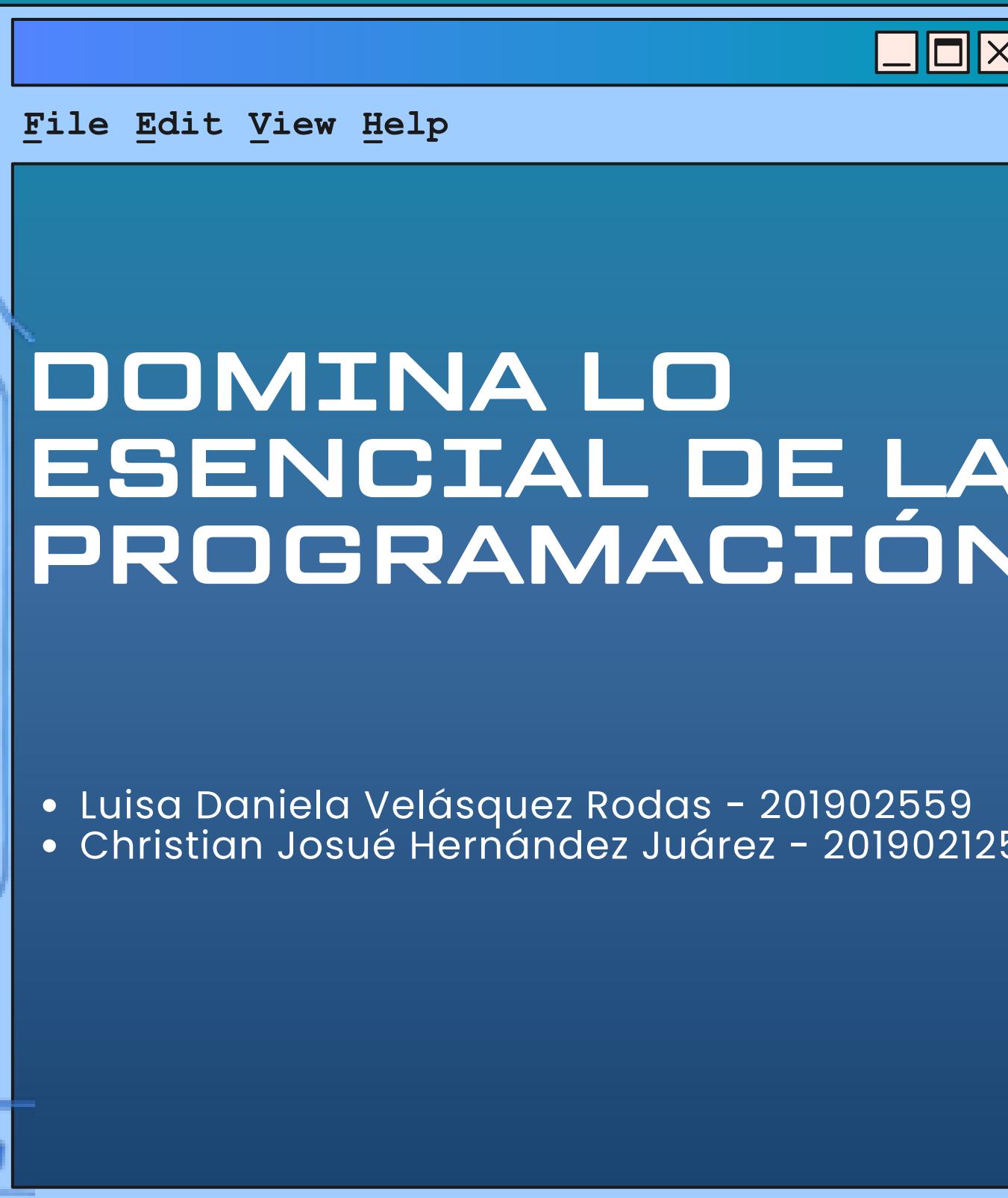


DE CERO A PYTHON



201902559

201902125



PROGRAMACION

Programar es el proceso de crear instrucciones detalladas que una computadora puede entender y ejecutar para resolver problemas o realizar tareas específicas. Es como escribir una receta muy precisa que la máquina seguirá paso a paso.



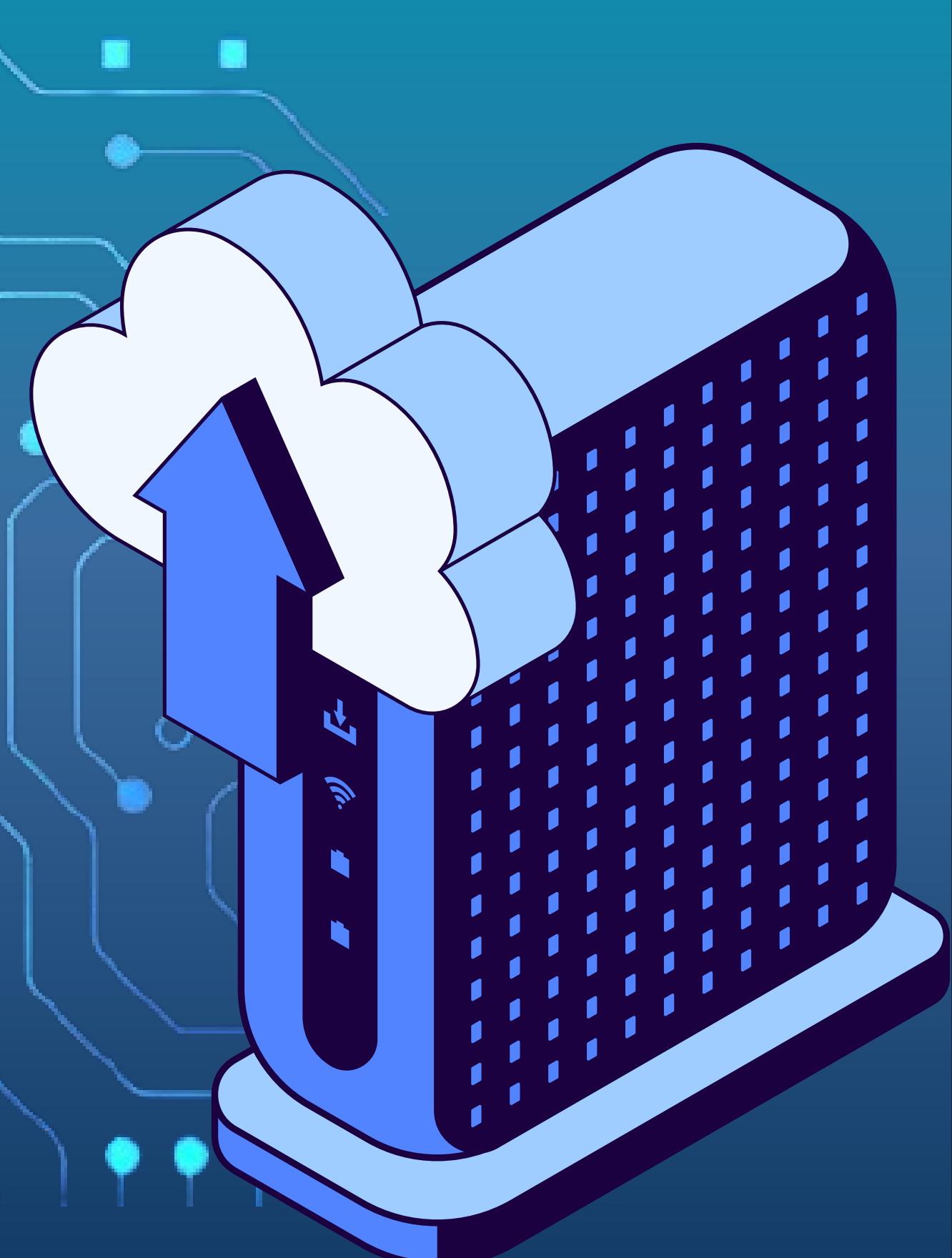


Photo Kiosk

LENGUAJES DE PROGRAMACION

QUE ES ?

es un sistema de comunicación que permite a los humanos escribir instrucciones que la computadora puede interpretar. Cada lenguaje tiene su propia sintaxis (reglas de escritura) y características específica

PYTHON

- Lenguaje interpretado, de alto nivel y propósito general.
- Filosofía: legibilidad y productividad.
- Comunidad y ecosistema enormes.

PAGE 4

VARIABLES



Las variables son contenedores que almacenan datos en la memoria de la computadora. Puedes pensar en ellas como "cajas" etiquetadas donde guardas información que puedes usar y modificar durante la ejecución del programa.

 **ENTEROS**

Números sin decimales
(1, 5, -10)

 **FLOTANTES**

Números con
decimales (3.14, -2.5)

 **CADENAS**

Texto entre comillas
("Hola", "123")

 **BOOLEANAS**

Valores de verdadero o
falso (true/false)



 **ARRAY**

Colecciones de
elementos ([1, 2, 3] o
["a", "b", "c"])

Awesome Web Browser X



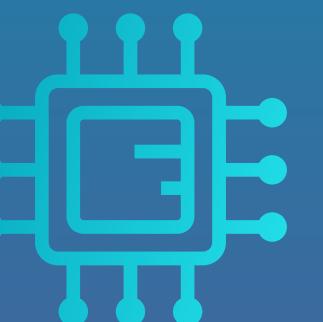
OPERADORES

Son símbolos que realizan operaciones sobre variables y valores:

ARITMÉTICOS

```
a = 10
b = 3

print("== OPERADORES ARITMÉTICOS ==")
print(f"{a} + {b} = {a + b}")      # Suma: 13
print(f"{a} - {b} = {a - b}")      # Resta: 7
print(f"{a} * {b} = {a * b}")      # Multiplicación: 30
print(f"{a} / {b} = {a / b}")      # División: 3.333...
print(f"{a} % {b} = {a % b}")      # Módulo (resto): 1
print(f"{a} ** {b} = {a ** b}")    # Potencia: 1000
```

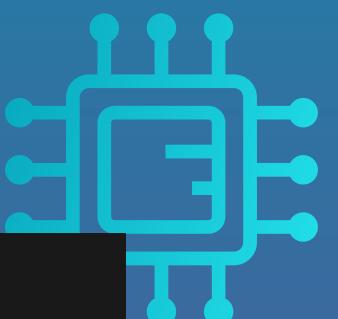




Awesome Web Browser X

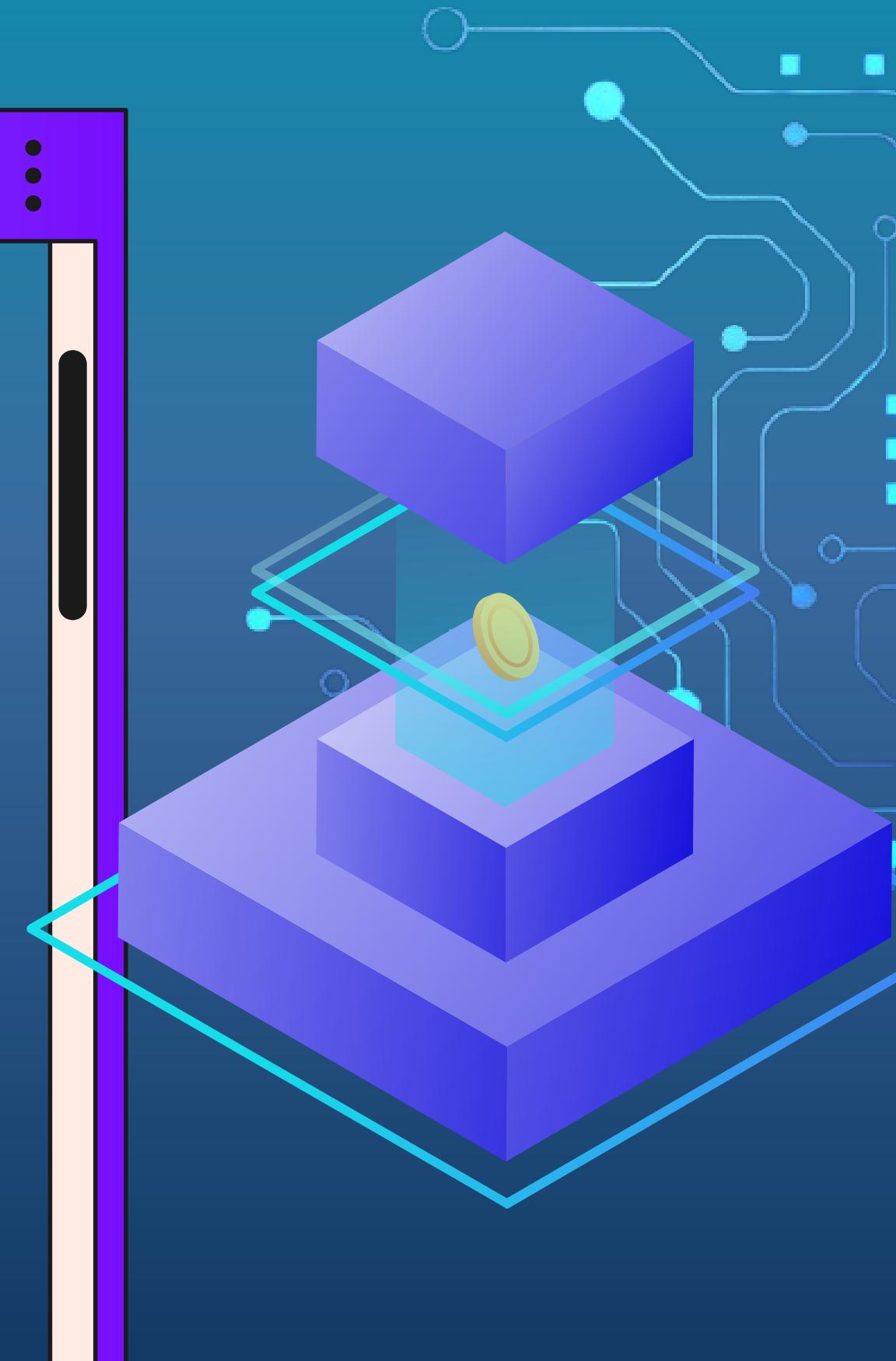


COMPARACION



```
# Ejemplos en Python
x = 5
y = 8

print("\n==== OPERADORES DE COMPARACIÓN ===")
print(f"{x} == {y} : {x == y}")      # Igual: False
print(f"{x} != {y} : {x != y}")      # Diferente: True
print(f"{x} > {y} : {x > y}")       # Mayor: False
print(f"{x} < {y} : {x < y}")       # Menor: True
print(f"{x} >= {y} : {x >= y}")     # Mayor o igual: False
print(f"{x} <= {y} : {x <= y}")     # Menor o igual: True
```





Awesome Web Browser X



OPERADORES LÓGICOS

```
# Ejemplos en Python
es_mayor = True
tiene_licencia = False
edad = 20

print("\n== OPERADORES LÓGICOS ==")
print(f"AND: {es_mayor} and {tiene_licencia} = {es_mayor and tiene_licencia}")
print(f"OR: {es_mayor} or {tiene_licencia} = {es_mayor or tiene_licencia}")
print(f"NOT: not {es_mayor} = {not es_mayor}")
```

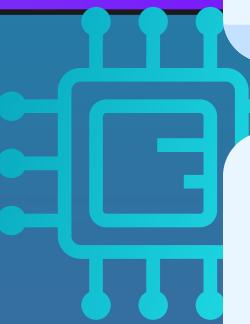
and



or



not





ESTRUCTURAS DE CONTROL

- CONDICIONALES
- CICLO/BUCLES

CONDICIONALES

Un condicional es una estructura que permite que un programa tome decisiones. Le dice al programa: "Si se cumple esta condición, entonces ejecuta este código; si no, haz otra cosa"

```
if (condición):
    # Código que se ejecuta si la condición es VERDADERA
else:
    # Código que se ejecuta si la condición es FALSA
```

```
edad = 20
tiene_licencia = True

if edad >= 18:
    if tiene_licencia:
        print("Puedes conducir")
    else:
        print("Necesitas licencia")
else:
    print("Eres muy joven para conducir")
```

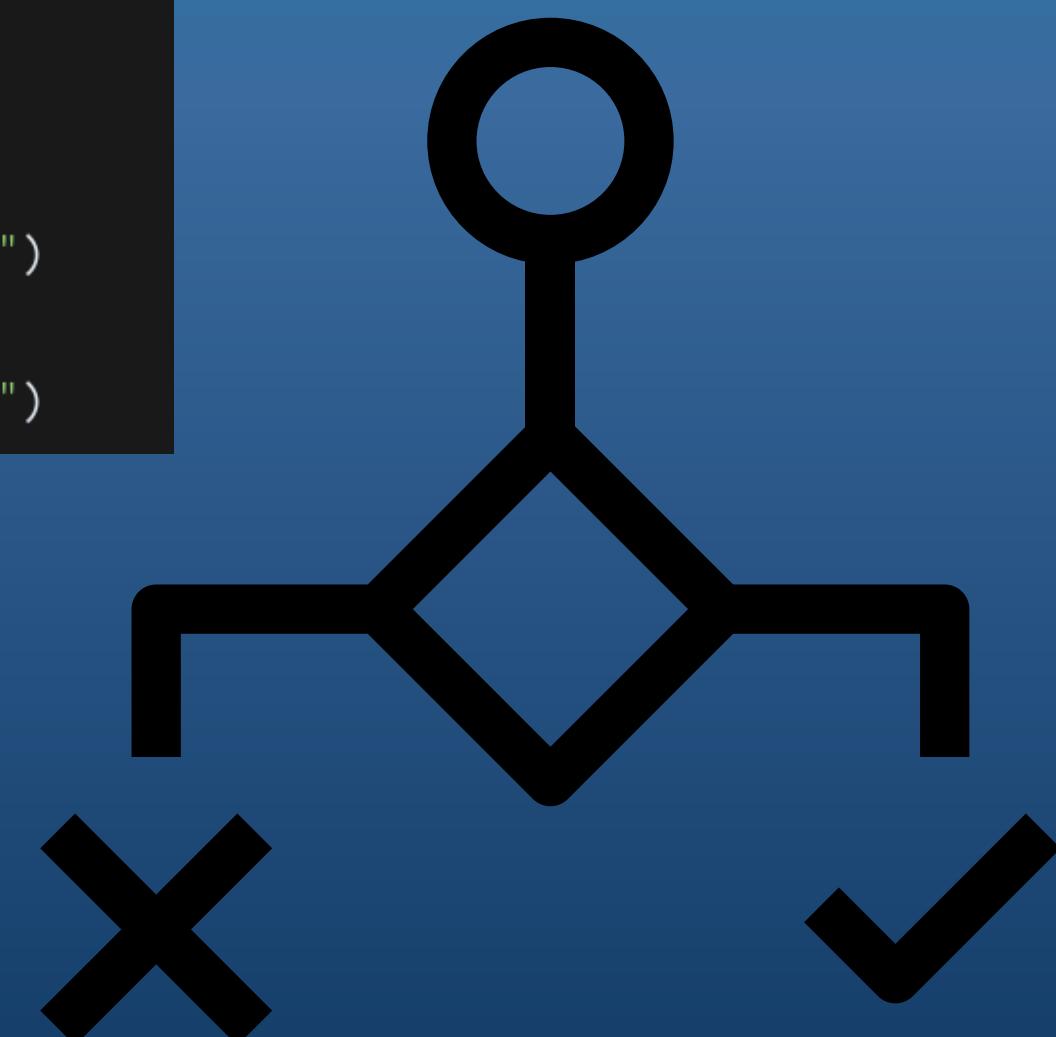
If/Else (Si/Sino):

- Si una condición es verdadera, ejecuta un bloque de código
- Si es falsa, puede ejecutar otro bloque alternativo

```
edad = 18

if edad >= 18:
    print("Eres mayor de edad")
else:
    print("Eres menor de edad")
```

Los condicionales anidados son condicionales que se colocan dentro de otros condicionales. Es como tener una decisión dentro de otra decisión.





USA

CICLOS II BUCLES

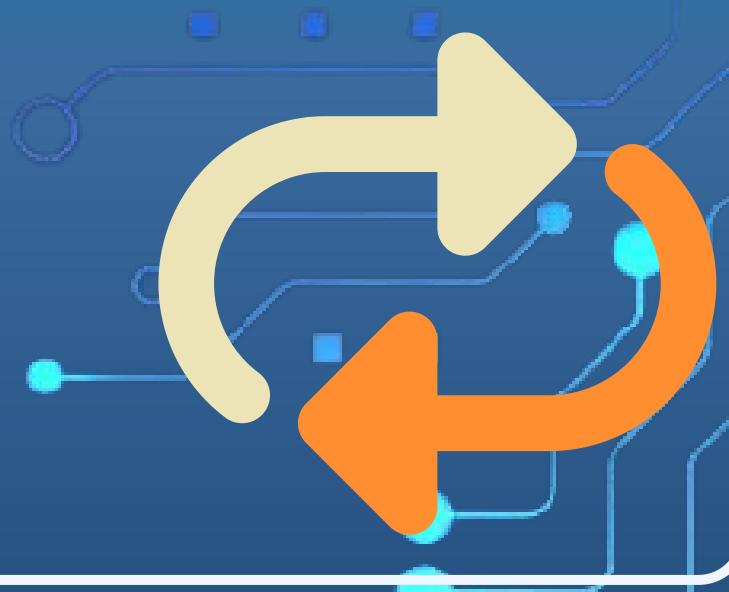


Los ciclos o bucles son estructuras que permiten repetir un bloque de código múltiples veces, hasta que se cumpla una condición específica.

- Automatizan tareas repetitivas
- Procesan colecciones de datos
- Eliminan código duplicado
- Ahorran tiempo y líneas de código

TIPOS DE BUCLES

- Ciclo For
- Ciclo While





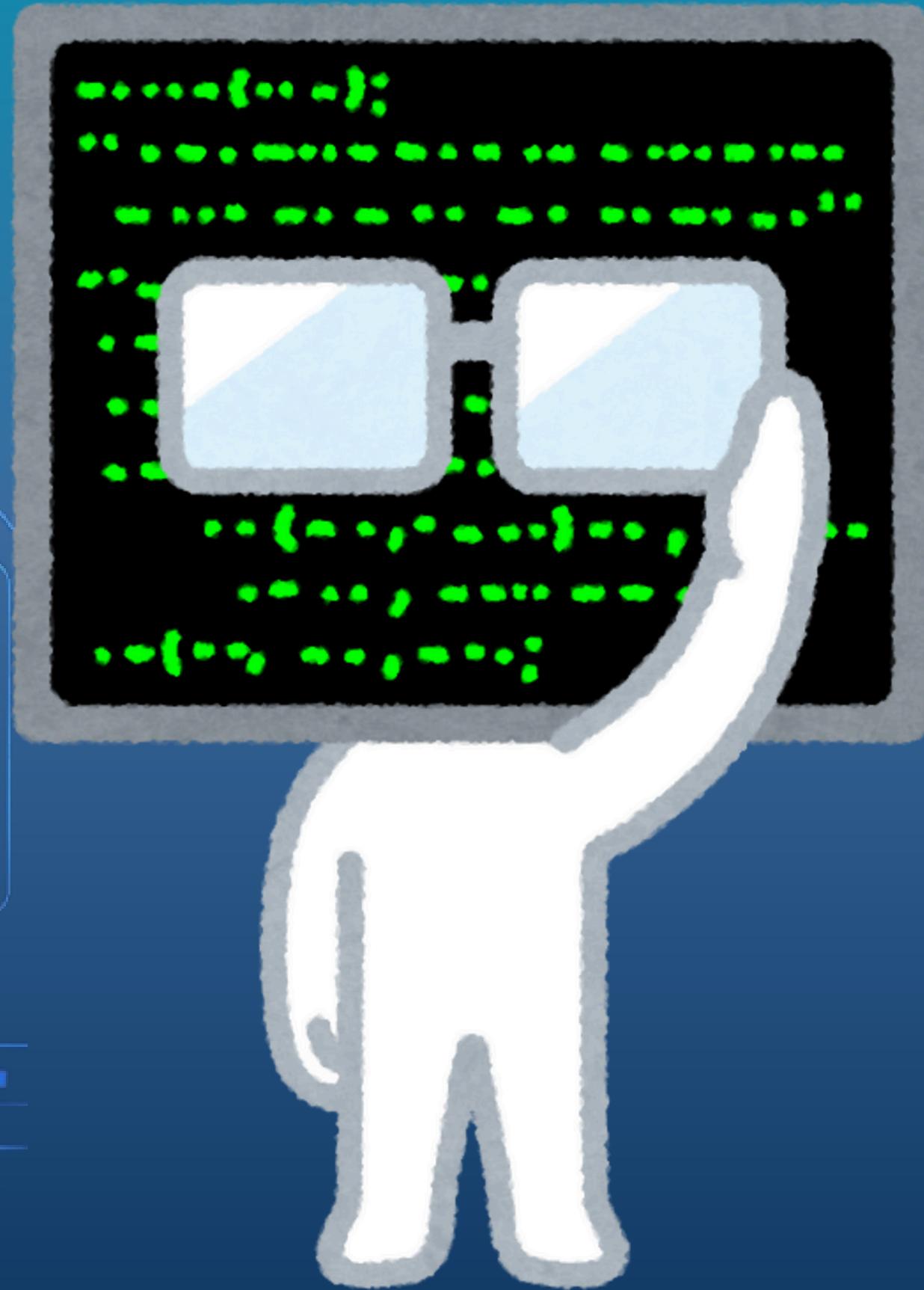
CICLO FOR

El ciclo for es una estructura de control que permite repetir un bloque de instrucciones un número determinado de veces

- Conoces el número exacto de iteraciones
- Vas a recorrer listas, rangos, o colecciones
- Necesitas un contador automático

```
# Perfecto para for  
for i in range(10):  
    for elemento in lista:  
        for caracter in texto:
```

```
# 10 veces exactas  
# Recorrer colección  
# Procesar texto
```



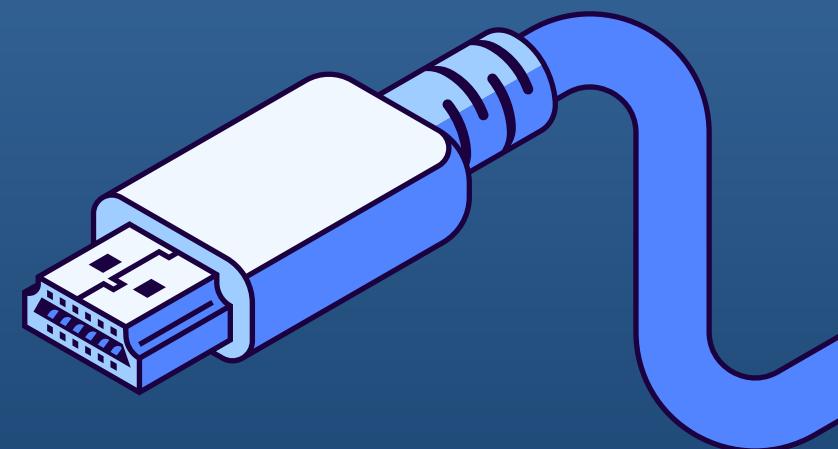
CICLO WHILE

El ciclo while es una estructura de control repetitiva que ejecuta un bloque de instrucciones mientras se cumpla una condición lógica.

- No sabes cuántas veces se repetirá
- Depende de entrada del usuario
- Esperas un evento externo
- La condición puede cambiar impredeciblemente

```
while (condición) {  
    // Código que se repite  
}
```

CONTROL DE BUCKLES



 **BREAK**

Detener el bucle inmediatamente

```
for numero in range(1, 11):
    if numero == 5:
        break # Sale del bucle cuando número es 5
    print(numero)
# Resultado: 1 2 3 4
```

 **CONTINUE**

Saltar a la siguiente iteración

```
for numero in range(1, 11):
    if numero % 2 == 0: # Si es par
        continue # Salta los números pares
    print(numero)
# Resultado: 1 3 5 7 9
```

 **ELSE**

en bucles - Se ejecuta si el bucle termina normalmente

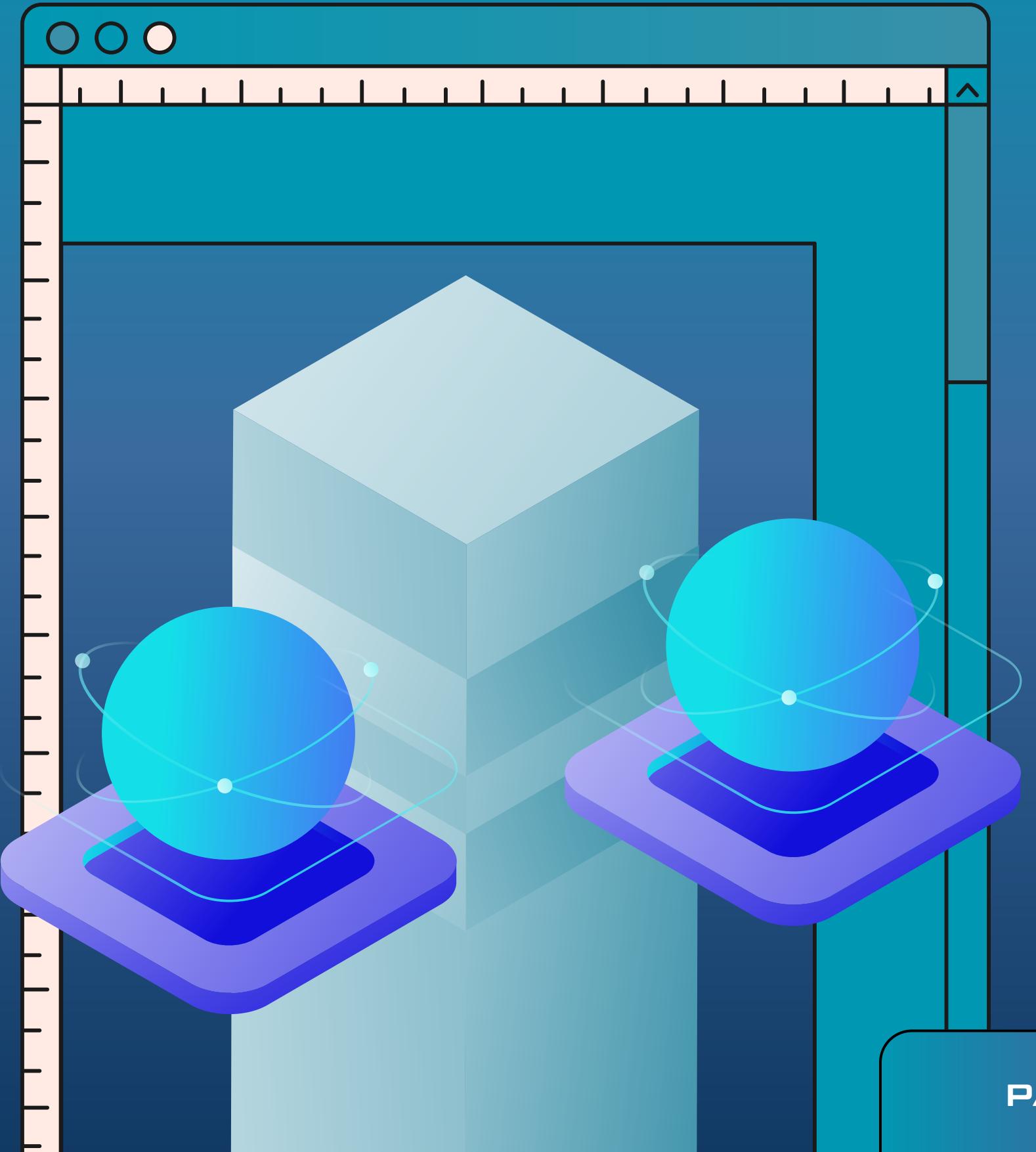
```
for numero in range(1, 4):
    print(numero)
else:
    print("Bucle completado ✓")
```

INSTALLACION PYTHON



PASOS

- Pasos 1 Visita el sitio oficial:
Abre tu navegador y ve a
python.org
- Descargar la versión más reciente
- Paso 2: Instalar Python
- Paso 3: instalar visual studio code
- Parte 4: Configurar VS Code para Python
- Parte 5: Crear tu primer programa Python



ESTRUCTURAS DE DATOS

- Listas
- Tuplas
- Diccionarios

```
mi_lista = [1, 2, 3]
mi_lista.insert(1, "nuevo")
# Ahora, mi_lista es [1, "nuevo", 2, 3]
```





LISTAS

En Python, las listas son un tipo de contenedor compuesto, que se utilizan para almacenar conjuntos de elementos relacionados del mismo tipo o de tipos distintos. Es uno de los tipos de secuencia que existen en Python, con la particularidad de que son mutables. Esto quiere decir que su contenido se puede modificar después de haber sido creada.

Para crear una lista en Python, simplemente hay que encerrar una secuencia de elementos separados por comas entre corchetes [].

#Lista con elementos del mismo tipo

```
numeros = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

lista de elementos de diferente tipo

```
elementos = [3, 'a', true, 3.14, "hola mundo"]
```

Para acceder a los elementos de una lista en Python se utilizan los índices. El índice es el número entero que indica la posición de un elemento en una lista. El primer elemento de una lista siempre comienza con 0.

length = 5					
	'p'	'r'	'o'	'b'	'e'
index	0	1	2	3	4
negative index	-5	-4	-3	-2	-1



OPERACIONES BÁSICAS

**APPEND**

Agrega un elemento al final de la lista

**INSERT**

Agrega el elemento en la posición indicada.

**REMOVE**

Elimina la primera coincidencia del elemento indicado.

**POP**

Elimina el último y lo retorna

**SORT**

Ordena alfabeticamente o numéricamente.

**REVERSE**

Invierte el orden.

**CLEAR**

Vacia la lista.

**LEN**

Retorna el tamaño de la lista



RECORRER LISTAS

Para recorrer listas en python se utiliza un ciclo for

```
#webdebe.xyz

mylist = [1,4,7,3,21,34,43,55,66,87,2345,435,345,254,435,2354]

for x in mylist:
    print(x)
```





LISTAS ANIDADAS

Listas dentro de listas

```
# Lista de listas
matriz = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]

# Lista con diferentes tipos de datos
datos_personales = [["Alice", 30], ["Bob", 25], ["Charlie", 35]]

# Anidación más profunda
lista_compleja = ['a', 'b', ['cc', 'dd', ['eee', 'fff']], 'g', 'h']
```





Awesome Web Browser X

← → ⌂



TUPLAS

Las tuplas en Python son estructuras de datos ordenadas e inmutables, lo que significa que no se pueden modificar una vez creadas. A diferencia de las listas, que son mutables, las tuplas tienen un tamaño y contenido fijos.

Características principales

- Inmutables: No se pueden añadir, eliminar ni cambiar elementos después de la creación.
- Ordenadas: Los elementos tienen un orden definido y se puede acceder a ellos mediante un índice, al igual que en las listas.
- Permiten duplicados: Pueden contener elementos con el mismo valor.
- Múltiples tipos de datos: Una sola tupla puede contener elementos de diferentes tipos (enteros, cadenas, listas, etc.).
- Mayor eficiencia: Las tuplas son generalmente más rápidas y consumen menos memoria que las listas, ya que Python optimiza el espacio de memoria para ellas al saber que no cambiarán de tamaño.

```
43 - mifamilia = {  
44 -     "hijo1" : {  
45 -         "nombre" : "Dennise",  
46 -         "año" : 2004  
47 -     },  
48 -     "hijo2" : {  
49 -         "nombre" : "Fernando",  
50 -         "año" : 2007  
51 -     },  
52 -     "hijo3" : {  
53 -         "nombre" : "Omar",  
54 -         "año" : 2011  
55 -     }  
56 - }  
57 - for x, y in mifamilia.items():  
58 -     print(x, y)
```



```
hijo3 {'año': 2011, 'nombre': 'Omar'}  
hijo1 {'año': 2004, 'nombre': 'Dennise'}  
hijo2 {'año': 2007, 'nombre': 'Fernando'}
```

DICCIONARIOS

Un diccionario es una colección en Python que guarda pares clave–valor.

- Clave → el identificador único (como el “nombre del dato”).
- Valor → la información asociada a esa clave.
- Son mutables (se pueden modificar).
- Las claves deben ser únicas e inmutables, pero los valores pueden ser cualquier cosa.

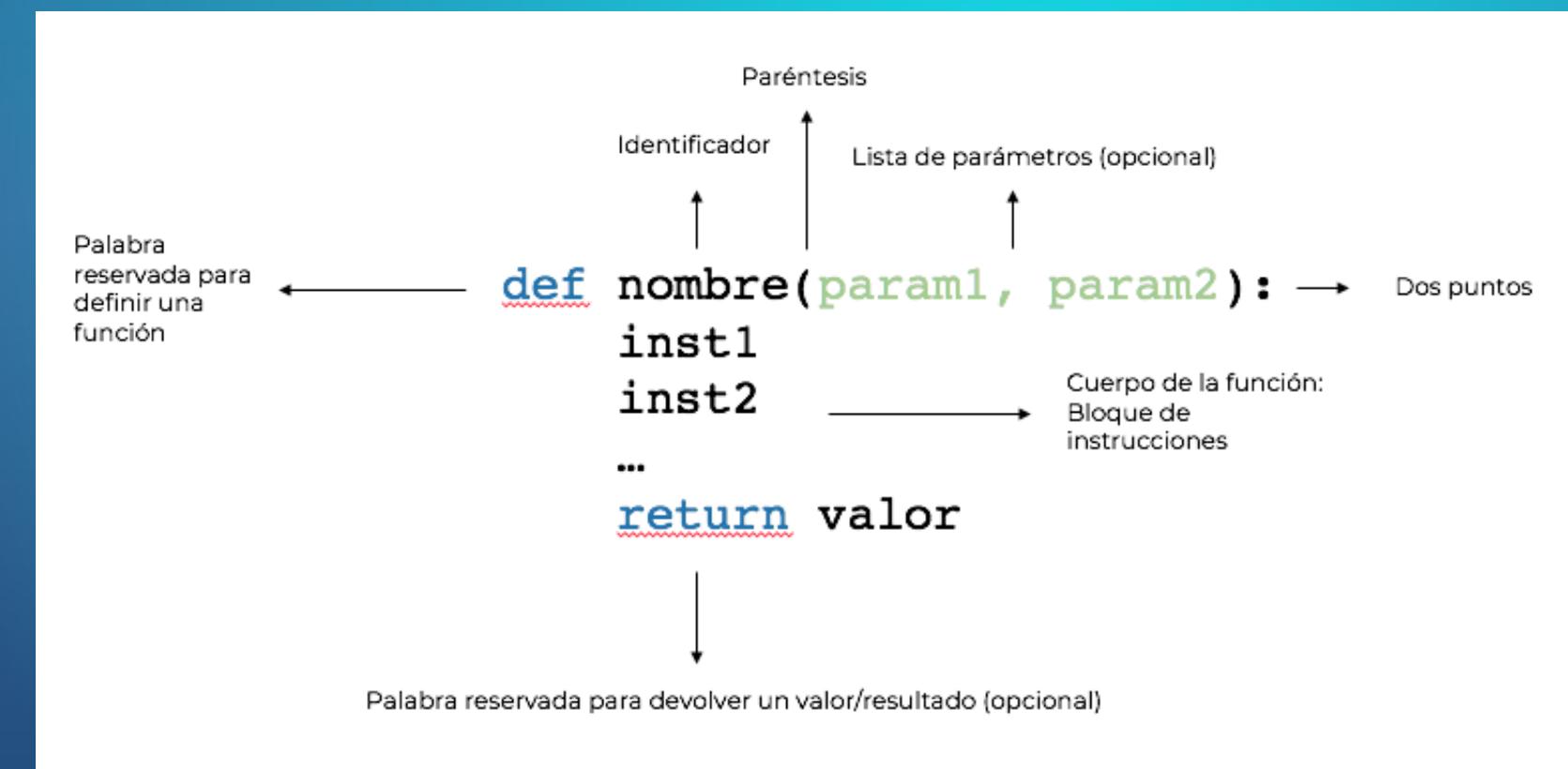




FUNCIONES

Una función es un bloque de código reutilizable que realiza una tarea específica.
Permite organizar el código y evitar repeticiones.
Se define con la palabra clave def.

- ➔ Sin parámetros y sin retorno
- ➔ Con parámetros pero sin retorno
- ➔ Con parámetros y con retorno
- ➔ Con múltiples parámetros
- ➔ Con múltiples retornos





ÁMBITO ENTRE VARIABLES

El ámbito o scope de una variable se refiere a dónde en el código una variable es accesible y puede ser utilizada.

LOCAL

Solo existe dentro de una función o bloque específico

GLOBAL

Existe en todo el programa, accesible desde cualquier parte.

```
def mi_funcion():
    variable_local = "Solo existo aquí dentro"
    print(variable_local) # ✓ Funciona

mi_funcion()
# print(variable_local) # ✗ ERROR: No existe fuera de la función
```

```
variable_global = "Soy accesible desde cualquier lugar"

def funcion1():
    print(f"Dentro de funcion1: {variable_global}") # ✓ Acceso

def funcion2():
    print(f"Dentro de funcion2: {variable_global}") # ✓ Acceso

funcion1()
funcion2()
print(f"Fuera: {variable_global}") # ✓ Acceso
```



MUCHAS
GRACIAS POR
SU ATENCIÓN!