1. FUNCIONES

En Python, una función es un bloque de código reutilizable que realiza una tarea específica. Las funciones se utilizan para dividir un programa en fragmentos más pequeños y manejables, lo que facilita el desarrollo, la organización y la reutilización del código. Para usar una función, primero debes definirla y luego puedes llamarla desde otras partes del programa.

Definición básica de una función:

```
def nombre_de_funcion():    #
Código de la función
  # ...
```

def es la palabra clave utilizada para definir una función.

nombre_de_funcion es el nombre que le das a tu función. Puedes elegir cualquier nombre válido de acuerdo con las convenciones de nomenclatura de Python.

Los paréntesis vacíos () se utilizan para indicar que la función no requiere ningún argumento.

El bloque de código de la función se indentará y contendrá las instrucciones que se ejecutarán cuando se llame a la función.

Llamada básica a una función:

```
nombre_de_funcion()  # Llamada a la función
```

Para llamar a una función, simplemente escribe su nombre seguido de paréntesis ().

Ejemplo básico:

```
def saludar():
  print("Hola, ¡bienvenido!") saludar()  # Imprime
```

"Hola, ¡bienvenido!"

En este ejemplo, se define una función llamada saludar() que imprime un mensaje de bienvenida. Luego, se llama a la función utilizando saludar() y se imprime el mensaje.

Ejemplo:

En este ejemplo, se define una función llamada suma() que toma dos argumentos a y b, y devuelve la suma de los dos valores. Luego, se llama a la función con los argumentos 5 y 3, y se asigna el resultado a la variable resultado. Finalmente, se imprime el valor de resultado, que es 8.

```
def suma(a, b):    return a + b

resultado = suma(5, 3)

print(resultado)  # Imprime 8
```

El uso más común de las funciones en Python es la modularización y reutilización del código. Aquí tienes algunas de las aplicaciones más comunes de las funciones en Python:

- Abstracción y organización del código: Las funciones permiten dividir un programa en bloques más pequeños y manejables, lo que facilita la comprensión y el mantenimiento del código. Puedes definir funciones para realizar tareas específicas y luego llamarlas según sea necesario.
- Reutilización del código: Al definir una función, puedes usarla en múltiples lugares de tu programa sin tener que repetir el mismo código una y otra vez. Esto mejora la legibilidad, reduce la duplicación y facilita los cambios y actualizaciones futuras.
- Encapsulamiento de la lógica: Las funciones permiten encapsular la lógica y la funcionalidad en un bloque separado, lo que hace que el código sea más modular y fácil de entender. Esto promueve la separación de preocupaciones y ayuda a crear programas más estructurados y mantenibles.
- Abstracción de tareas repetitivas: Si tienes un conjunto de instrucciones que se repiten en varios lugares de tu programa, puedes encapsular esas instrucciones en una función y llamarla cada vez que necesites ejecutar esa tarea. Esto simplifica el código y evita errores causados por la repetición manual de código.
- Manejo de argumentos y valores de retorno: Las funciones te permiten definir parámetros (argumentos) que pueden ser pasados a la función al llamarla. Además, puedes utilizar la declaración return para devolver valores desde la función. Esto facilita la personalización y la flexibilidad de las funciones.

2. FUNCIONES INCORPORADAS A PYTHON MAS UTILIZADAS

Python ofrece una amplia variedad de funciones incorporadas (built-in functions) que son ampliamente utilizadas en el desarrollo de programas. Algunas de las funciones más comunes y útiles en Python incluyen:

print(): Imprime mensajes en la consola o en la salida estándar.

print("Hola, mundo!") input(): Lee una línea de texto ingresada por el

usuario desde la entrada estándar.

nombre = input("Ingresa tu nombre: ")

len(): Devuelve la longitud (cantidad de elementos) de un objeto, como una cadena, lista, tupla o diccionario.

texto = "Hola" longitud = len(texto) range(): Genera una secuencia

de números dentro de un rango especificado.

for i in range(1, 5):

print(i) type(): Devuelve el tipo de datos

de un objeto.

numero = 10
tipo = type(numero)

str(), int(), float(): Convierten un objeto a una cadena, entero o flotante, respectivamente.

numero_str = str(10) numero_int =

int("15") numero_float = float("3.14")

list(), tuple(), set(), dict(): Crea un objeto de tipo lista, tupla, conjunto o diccionario, respectivamente.

mi_lista = list([1, 2, 3]) mi_tupla = tuple((4, 5, 6))

mi_conjunto = set([7, 8, 9]) mi_diccionario =

dict(nombre="Juan", edad=25)

max(), min(), sum(): Calculan el valor máximo, mínimo y la suma de una secuencia de números, respectivamente.

numeros = [10, 5, 8, 3] maximo = max(numeros) minimo = min(numeros)

suma = sum(numeros) upper(), lower(): Convierte una cadena a

mayúsculas o minúsculas, respectivamente.

texto = "Hola" mayusculas = texto.upper() minusculas = texto.lower() capitalize():

Convierte el primer carácter de una cadena en mayúscula y el resto en

minúsculas.

texto = "hola mundo" capitalizado =

texto.capitalize()

strip(), lstrip(), rstrip(): Elimina los espacios en blanco al principio y/o al final de una cadena.

texto = " Hola " sin_espacios = texto.strip() sin_espacios_izquierda = texto.lstrip() sin_espacios_derecha = texto.rstrip() split(): Divide una cadena en una lista de subcadenas utilizando un separador especificado.

texto = "Hola,como,estas" subcadenas = texto.split(",") join(): Concatena una lista de cadenas utilizando un separador especificado. subcadenas = ["Hola", "como", "estas"] texto = "-".join(subcadenas) replace(): Reemplaza todas las apariciones de una subcadena por otra en una cadena.

texto = "Hola mundo" nuevo_texto = texto.replace("mundo", "amigo")

3. ESTRUCTURAS DE CONTROL

Una estructura de control en programación se utiliza para alterar el flujo de ejecución de un programa, permitiendo la toma de decisiones y la repetición de bloques de código según ciertas condiciones. En Python, existen varias estructuras de control que te permiten controlar el flujo de ejecución de tu programa.

Las principales estructuras de control en Python son las siguientes:

Estructura if-else: Permite ejecutar un bloque de código si se cumple una condición, y otro bloque de código si no se cumple la condición.

```
if condicion:
    # Bloque de código si se cumple la condición else:
    # Bloque de código si no se cumple la condición
```

Estructura elif (else if): Se utiliza para evaluar múltiples condiciones secuenciales.

```
if condicion1:
    # Bloque de código si se cumple la primera condición elif condicion2:
    # Bloque de código si se cumple la segunda condición else:
    # Bloque de código si no se cumple ninguna de las condiciones anteriores
```

Ejemplo:

En este ejemplo, se define una función llamada calcular_precio_descuento que toma dos parámetros: precio y descuento. La función verifica diferentes

condiciones utilizando la estructura if para calcular y mostrar el precio con descuento.

- La primera condición verifica que el precio sea mayor que cero. Si es así, continúa evaluando las siguientes condiciones.
- La segunda condición verifica si el descuento está en el rango de 0 a 100. Si se cumple, se calcula el precio_final aplicando el descuento proporcionado.
- Si el descuento es igual a cero, se imprime un mensaje indicando que no hay descuento aplicado.
- Si el descuento no está en el rango válido, se imprime un mensaje de error.
- Si el precio no es mayor que cero, se imprime un mensaje indicando que el precio debe ser mayor que cero.
- Luego, se llaman a la función calcular_precio_descuento con diferentes valores de precio y descuento para probar las diferentes condiciones y ver los resultados en función de los valores ingresados.

```
def calcular_precio_descuento(precio, descuento):
  if precio > 0:       if descuento > 0 and descuento <= 100:
      precio_final = precio - (precio * descuento / 100)          print(f"El precio con
descuento es: {precio_final}")       elif descuento == 0:
      print("No hay descuento aplicado.")       else:
      print("El descuento debe ser un valor entre 0 y 100.")    else:
    print("El precio debe ser mayor que cero.")
calcular_precio_descuento(100, 20)  # Precio con descuento: 80.0
calcular_precio_descuento(200, 0)   # No hay descuento aplicado.
calcular_precio_descuento(150, 150) # El descuento debe ser un valor entre 0 y
100.
calcular_precio_descuento(0, 10)    # El precio debe ser mayor que cero.
```

Estructura de bucle for: Permite iterar sobre una secuencia (como una lista, una tupla o una cadena) o sobre un rango de números.

```
for elemento in secuencia:
  # Bloque de código a ejecutar en cada iteración
```

Ejemplo:

Se utiliza un bucle for para iterar sobre una lista de números. Se verifica si cada número es par utilizando la condición numero % 2 == 0. Si el número es par, se agrega a la variable suma.

Después de que el bucle for ha terminado de iterar sobre todos los números de la lista, se imprime el resultado de la suma de los números pares.

numeros = [2, 4, 6, 8, 10] suma = 0

for numero in numeros: if

numero % 2 == 0: suma

+= numero

print(f"La suma de los números pares es: {suma}")

Estructura de bucle while: Se utiliza para repetir un bloque de código mientras se cumpla una condición.

while condicion:

Bloque de código a ejecutar mientras se cumple la condición

Ejemplo:

En este ejemplo, se genera un número secreto aleatorio entre 1 y 100 utilizando el módulo random. Luego, se inicia un bucle while que se ejecuta infinitamente hasta que el usuario adivine el número secreto.

En cada iteración del bucle while, se le pide al usuario que ingrese un número y se compara con el número secreto. Dependiendo de la comparación, se imprime un mensaje indicando si el número ingresado es mayor o menor que el número secreto.

Si el usuario adivina el número, se imprime un mensaje de felicitaciones junto con la cantidad de intentos realizados, y se utiliza la instrucción break para salir del bucle while.

import random intentos = 0 numero_secreto =

random.randint(1, 100) print("¡Adivina el número

secreto!") while True:

intentos += 1 numero_adivinar = int(input("Ingresa un número: "))

if numero_adivinar == numero_secreto:

print(f"¡Felicidades! Adivinaste el número en {intentos} intentos.") break elif

numero_adivinar < numero_secreto:

print("El número ingresado es menor al número secreto.") else:

print("El número ingresado es mayor al número secreto.") Estructura de bucle do-while (no está disponible de forma directa en Python): Aunque Python no tiene una estructura de bucle do-while incorporada, se puede simular utilizando un bucle while con una condición de salida al final.

```
while True:
    # Bloque de código a ejecutar al menos una vez     if not condicion:
        break  # Condición de salida del bucle
```

4. USO DE IF – ElIf

La elección entre usar if-else o elif depende de la lógica que deseas implementar en tu programa. Aquí tienes algunas consideraciones para ayudarte a decidir cuál usar en diferentes situaciones:

if-else:

- Se utiliza cuando solo tienes dos posibles caminos o resultados en función de una condición.
- El bloque de código dentro del if se ejecuta si la condición es verdadera, mientras que el bloque de código dentro del else se ejecuta si la condición es falsa.
- Si tienes más de dos posibilidades y solo necesitas evaluar una vez, puedes utilizar varios if-else encadenados.

Ejemplo de uso de if-else:

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

- Se utiliza cuando tienes múltiples condiciones y deseas evaluarlas secuencialmente.
- Evita evaluar todas las condiciones una vez que se encuentra una que sea verdadera.
- Se ejecuta el bloque de código correspondiente a la primera condición verdadera y luego se sale del conjunto de condiciones.
- Ejemplo de uso de elif:

```
puntaje = 85 if
puntaje >= 90:
    print("Obtuviste una A") elif puntaje >=
80:
    print("Obtuviste una B") elif puntaje >=
70:
    print("Obtuviste una C")
```

else:

print("Obtuviste una D")

En general, si solo necesitas evaluar una condición simple y tienes dos resultados posibles, utiliza if-else. Por otro lado, si tienes múltiples condiciones y quieres evaluarlas secuencialmente, utiliza elif.

Recuerda que puedes combinar if-else y elif según tus necesidades para implementar lógica más compleja y realizar diferentes acciones en función de múltiples condiciones.

5. USO DE FOR O WHILE

En Python, tanto el bucle for como el bucle while son útiles para diferentes situaciones. La elección entre utilizar uno u otro depende de la tarea específica que deseas realizar. Aquí hay algunas consideraciones para ayudarte a decidir cuándo usar cada uno:

Bucle for:

- Se utiliza cuando se conoce la cantidad de elementos a iterar, como recorrer una lista, tupla, cadena o rango de números.
- Es especialmente útil cuando deseas realizar una tarea para cada elemento de una secuencia.
- La sintaxis es más concisa y legible.
- No necesitas controlar manualmente el contador o la condición de finalización del bucle.

Ejemplo de uso de bucle for:

numeros = [1, 2, 3, 4, 5] for numero

in numeros:

print(numero)

Bucle while:

- Se utiliza cuando no se conoce la cantidad exacta de iteraciones necesarias y se basa en una condición para continuar o salir del bucle.
- Es útil cuando deseas repetir un bloque de código hasta que se cumpla una determinada condición.
- Se puede utilizar para implementar lógica más compleja que requiere un mayor control sobre el flujo de ejecución.

Ejemplo de uso de bucle while:

contador = 0 while

contador < 5:

```
print(contador)

contador += 1
```

En resumen, si sabes cuántas veces necesitas iterar o quieres realizar una tarea para cada elemento de una secuencia conocida, el bucle for es la elección adecuada. Por otro lado, si no sabes cuántas veces necesitas repetir un bloque de código y deseas controlar manualmente la condición de finalización, el bucle while es más apropiado.

En algunos casos, también puedes combinar ambos bucles para lograr el resultado deseado, utilizando el bucle for para iterar sobre una secuencia conocida y el bucle while para realizar tareas adicionales o realizar una repetición basada en una condición específica dentro de cada iteración del bucle for.