



FUNCIONES

PYTHON

¿QUÉ SON LAS FUNCIONES EN PYTHON?

1.1 DEFINICIÓN Y UTILIDAD.

1.2 SINTAXIS BÁSICA DE UNA
FUNCIÓN EN PYTHON.

ESTRUCTURA DE UNA FUNCIÓN EN PYTHON.

2.1 DECLARACIÓN DE LA FUNCIÓN.

2.2 PARÁMETROS DE ENTRADA.

2.3 CUERPO DE LA FUNCIÓN.

2.4 VALOR DE RETORNO.

DECLARACIÓN Y USO DE FUNCIONES EN PYTHON.

3.1 CREACIÓN DE FUNCIONES PERSONALIZADAS.

3.2 LLAMADO DE FUNCIONES Y USO DE PARÁMETROS.

3.3 EJEMPLOS PRÁCTICOS DE FUNCIONES EN PYTHON.

TIPOS DE VARIABLES DENTRO DE LAS FUNCIONES EN PYTHON.

4.1 VARIABLES LOCALES.

4.2 VARIABLES GLOBALES.

FUNCIONES ANIDADAS Y RECURSIVIDAD EN PYTHON.

5.1 FUNCIONES ANIDADAS.

5.2 FUNCIONES RECURSIVAS.

FUNCIONES PREDEFINIDAS DE PYTHON.



6.1 USO DE FUNCIONES
PREDEFINIDAS.

BIBLIOTECAS Y MÓDULOS EN PYTHON.

7.1 UTILIZACIÓN DE BIBLIOTECAS
Y MÓDULOS.

7.2 EJEMPLOS PRÁCTICOS DE
BIBLIOTECAS Y MÓDULOS EN
PYTHON.

BUENAS PRÁCTICAS EN LA PROGRAMACIÓN DE FUNCIONES EN PYTHON.

8.1 NOMENCLATURA DE
FUNCIONES.

8.2 DOCUMENTACIÓN DE
FUNCIONES.

CONCLUSIONES Y RECOMENDACIONES FINALES.

9.2 RECOMENDACIONES PARA EL
USO DE FUNCIONES EN PYTHON.

¿QUÉ SON LAS FUNCIONES EN PYTHON?

1.1 DEFINICIÓN Y UTILIDAD.

Una función en Python es un bloque de código que **hace una tarea específica** y se puede usar en cualquier momento. Es como una herramienta que te ayuda a **dividir tu programa en partes más pequeñas** y fáciles de entender.

La mejor parte de las funciones es que te permiten **escribir código una vez y usarlo muchas veces**, lo que ahorra tiempo y esfuerzo. Además, **las funciones pueden ser compartidas entre diferentes programas**, lo que las hace muy útiles para los programadores. En resumen, las funciones **hacen que tu código sea más organizado, legible y fácil de mantener**.

En inglés, "def" es una abreviatura de "define" que significa "definir" en español

1.2 SINTAXIS BÁSICA DE UNA FUNCIÓN EN PYTHON.

La sintaxis básica de una función en Python comienza con la palabra clave "def" seguida del nombre de la función, paréntesis y dos puntos.

Dentro de la función, se escribe el código que realiza la tarea deseada. La función puede o no recibir argumentos y puede devolver un valor usando la palabra clave "return". Es importante nombrar a la función de una manera clara y descriptiva, para que su uso sea fácil de entender.

ESTRUCTURA DE UNA FUNCIÓN EN PYTHON.

2.1 DECLARACIÓN DE LA FUNCIÓN.

La **declaración** de una función en Python **es la forma en que se define la función**. Esta declaración comienza con la palabra clave "**def**", seguida del nombre de la función, y **los paréntesis que pueden o no contener argumentos**. La sintaxis básica es la siguiente:

```
python
def nombre_funcion(argumento1, argumento2, ...):
    # cuerpo de la función
    
```

Es importante en python el tener cuidado con los espacios

2.1 DECLARACIÓN DE LA FUNCIÓN.

El nombre de la función debe seguir las mismas reglas que las variables en Python. Los argumentos son variables **opcionales** que pueden ser pasadas a la función para ser utilizadas en su cuerpo. Si no se necesita ningún argumento, los paréntesis todavía deben ser incluidos. El cuerpo de la función es donde se coloca el código que se ejecutará cuando se llame a la función.

Es importante destacar que la definición de una función no la ejecuta, sino que simplemente la crea para ser llamada posteriormente. La ejecución de la función se realiza mediante una llamada a la función.

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

2.1 DECLARACIÓN DE LA FUNCIÓN.

En este ejemplo, la función `saludar()` no tiene ningún argumento. Simplemente imprime el mensaje "Hola, bienvenido!" cuando se llama a la función. Para llamar a esta función, simplemente escribiríamos `saludar()` en nuestro programa principal.

Recordando los espacios! , No llamaras a la funcion dentro de la funcion si no fuera de ella.

¿Como me doy cuenta si estoy fuera o no de la funcion?.

```
# Definición de la función
def saludar():
    print("Hola, bienvenido!")

# Llamando a la función
saludar()
```

2.1 DECLARACIÓN DE LA FUNCIÓN.

Como puedes observar, `saludar()` no esta a la misma distancia que "print" esta alineado a `def saludar()` dentro de la funcion, ahora para llamar a la funcion creada, esta alineado con "`def`", Esto significa que donde esta `def` es la linea de tu codigo principal.

no te preocupes el editor de codigo que estes usando te ayudara con los espacios.

2.2 PARÁMETROS DE ENTRADA.

Los parámetros de entrada en Python **son valores que se pasan a una función para que esta realice una tarea específica**. Los parámetros **se definen dentro de los paréntesis de la definición de la función** y se utilizan dentro del cuerpo de la función para realizar cálculos o realizar operaciones en ellos.

Los parámetros de entrada en Python **son opcionales**, lo que significa que una función puede tener cero o varios parámetros de entrada. Además, los parámetros **pueden tener valores por defecto o ser opcionales**, lo que significa que pueden ser ignorados al llamar la función si se proporciona un valor por defecto.

2.2 PARÁMETROS DE ENTRADA.

```
def sumar(numero1, numero2):  
    resultado = numero1 + numero2  
    return resultado
```

En este caso, la función se llama "sumar" y toma dos parámetros de entrada: "numero1" y "numero2". Dentro del cuerpo de la función, se realiza una operación para sumar los dos números y se devuelve el resultado.

Luego, puedes llamar a esta función de la siguiente manera:

```
resultado_suma = sumar(2, 3)  
print(resultado_suma) # 5
```

2.2 PARÁMETROS DE ENTRADA.

```
def sumar(numero1, numero2):  
    resultado = numero1 + numero2  
    return resultado
```

```
resultado_suma = sumar(2, 3)  
print(resultado_suma) # 5
```

En este ejemplo, se llama a la función "sumar" proporcionando dos valores como argumentos de entrada: 2 y 3. La función realiza la suma de los dos números y devuelve el resultado, que se almacena en la variable "resultado_suma".

Este ejemplo ilustra cómo los parámetros de entrada en Python se definen dentro de los paréntesis de la definición de la función y se utilizan dentro del cuerpo de la función para realizar cálculos o realizar operaciones en ellos.

2.3 CUERPO DE LA FUNCIÓN.

Dentro del cuerpo de la función, se pueden incluir declaraciones de variables, estructuras de control de flujo como "if" o "for", operaciones aritméticas, llamadas a otras funciones, entre otras cosas.

Además, es importante recordar que las funciones pueden devolver valores mediante la declaración "return".

IMPORTANTE, el return sirve para casi darle finalidad a tu función , ¿por que casi? por que si usas condicionales en este caso el que cerraria tu función seria un else.

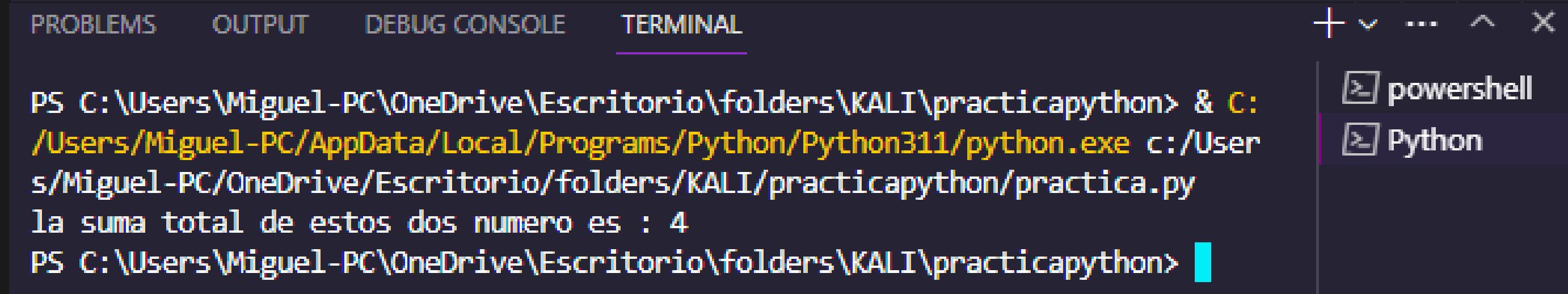
```
def dividir(a, b):
    if b == 0:
        return "Error: no se puede dividir entre cero"
    else:
        return a / b
```

2.4 VALOR DE RETORNO.

Aqui les dejo un ejemplo de VALOR DE RETORNO que pasaria si una funcion no tiene un return a comparacion de que si tenga un return.

```
def suma(num1,num2):
    resultado = num1 + num2
    return resultado

suma_total = suma(2,2)
print(f"la suma total de estos dos numero es : {suma_total}")
```



The screenshot shows a terminal window with the following content:

```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL + v ... ^ x
PS C:\Users\Miguel-PC\OneDrive\Escritorio\folders\KALI\practicapython> & C:/Users/Miguel-PC/AppData/Local/Programs/Python/Python311/python.exe c:/Users/Miguel-PC/OneDrive/Escritorio/folders/KALI/practicapython/practica.py
la suma total de estos dos numero es : 4
PS C:\Users\Miguel-PC\OneDrive\Escritorio\folders\KALI\practicapython>
```

The terminal interface includes tabs for PROBLEMS, OUTPUT, DEBUG CONSOLE, and TERMINAL, with the TERMINAL tab currently selected. To the right of the terminal window, there is a dropdown menu for selecting a terminal profile, with "Python" highlighted.

Usamos el return para retornar la operacion que hicimos dentro de la funcion , esto nos da el resultado esperado.

2.4 VALOR DE RETORNO.

```
def suma(num1,num2):  
    resultado = num1 + num2  
  
    return resultado  
  
suma_total = suma(2,2)  
print(f"la suma total de estos dos numero es : {suma_total}")
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

```
PS C:\Users\Miguel-PC\OneDrive\Escritorio\folders\KALI\practicapython> & C:/Users/Miguel-PC/AppData/Local/Programs/Python/Python311/python.exe c:/Users/Miguel-PC/OneDrive/Escritorio/folders/KALI/practicapython/practica.py  
la suma total de estos dos numero es : None  
PS C:\Users\Miguel-PC\OneDrive\Escritorio\folders\KALI\practicapython> □
```

Si no usamos el return , no habra nada para retornar,devolver.

DECLARACIÓN Y USO DE FUNCIONES EN PYTHON.

3.1 CREACIÓN DE FUNCIONES PERSONALIZADAS.

Crear funciones personalizadas es una **técnica importante** para escribir código de manera **eficiente**, modular y reutilizable en proyectos de programación. En Python, las funciones personalizadas son útiles para agrupar bloques de código que hacen una tarea específica y se pueden reutilizar en diferentes partes del programa. Al crear una función personalizada, se le da un nombre y un conjunto de parámetros, lo que hace que el código sea más fácil de leer y mantener.

Aquí te muestro un ejemplo sencillo de una función personalizada que calcula la suma de dos números y devuelve el resultado:

3.1 CREACIÓN DE FUNCIONES PERSONALIZADAS.

```
def sumar(a, b):  
    resultado = a + b  
    return resultado  
  
# Llamado a la función y asignación del resultado a una variable  
resultado_suma = sumar(5, 3)  
  
# Impresión del resultado  
print(resultado_suma)
```

En este ejemplo, la función sumar toma dos parámetros (a y b) y los suma para obtener un resultado, que se almacena en la variable resultado. Luego, la función devuelve este resultado usando la declaración return. Finalmente, la función es llamada en el programa principal pasándole los valores de los parámetros 5 y 3, y el resultado se almacena en la variable resultado_suma, que luego se imprime en la consola.

3.1 CREACIÓN DE FUNCIONES PERSONALIZADAS.

Este es un ejemplo muy simple, pero las funciones personalizadas pueden ser mucho más complejas y realizar tareas más sofisticadas. Al crear funciones personalizadas, se puede aumentar la modularidad y la legibilidad del código, lo que facilita la resolución de problemas y el mantenimiento del programa a medida que crece en complejidad.

3.3 EJEMPLOS PRÁCTICOS DE FUNCIONES EN PYTHON.

Ejemplos prácticos

1-Cálculo del área de un rectángulo:

```
def calcular_area_rectangulo(base, altura):  
    return base * altura  
  
area = calcular_area_rectangulo(4, 5)  
print(area) # Imprime 20
```

2-Para calcular el área de un rectángulo, debes multiplicar su base por su altura. La fórmula para el área de un rectángulo es:

$$\text{Área} = \text{base} \times \text{altura}$$

```
def convertir_celsius_a_fahrenheit(celsius):  
    fahrenheit = (celsius * 9/5) + 32  
    return fahrenheit  
  
temperatura_celsius = 25  
temperatura_fahrenheit = convertir_celsius_a_fahrenheit(temperatura_celsius)  
print(temperatura_fahrenheit) # Imprime 77.0
```

3.3 EJEMPLOS PRÁCTICOS DE FUNCIONES EN PYTHON.

Para convertir grados Celsius a grados Fahrenheit, se puede usar la siguiente fórmula:

$$F = (C * 9/5) + 32$$

donde "C" representa la temperatura en grados Celsius, y "F" representa la temperatura en grados Fahrenheit.

3-Función que calcula el área de un círculo dado su radio:

```
def area_circulo(radio):
    area = 3.1416 * radio ** 2
    return area
```

Para calcular el área de un círculo dado su radio, se puede utilizar la siguiente fórmula matemática:

$$\text{área} = \pi * \text{radio}^2$$

Donde "pi" es una constante matemática que se aproxima a 3.14159 y "radio" es la longitud del radio del círculo.

TIPOS DE VARIABLES DENTRO DE LAS FUNCIONES EN PYTHON

4.1 VARIABLES LOCALES.

Las variables locales en una función de Python son aquellas que se definen dentro de la función y solo existen dentro de ella. Esto significa que no se pueden acceder a ellas fuera de la función y no afectan a otras variables con el mismo nombre que puedan existir fuera de la función.

Veamos un ejemplo:

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

4.1 VARIABLES LOCALES.

En esta función, resultado es una variable local ya que se define dentro de la función suma y solo existe dentro de ella. Cuando se llama a la función suma con dos valores para a y b, se realiza la operación de suma y se almacena el resultado en resultado. Luego, la función devuelve este valor y la variable resultado deja de existir.

Es importante tener en cuenta que, si se intenta acceder a la variable resultado fuera de la función suma, se producirá un error ya que no existe fuera de la función.

Las variables locales son útiles para limitar el alcance de una variable y evitar posibles conflictos con variables que puedan tener el mismo nombre en otras partes del programa.

4.2 VARIABLES GLOBALES.

Las variables globales en Python son variables que se definen fuera de una función y pueden ser accedidas desde cualquier parte del programa. Estas variables son útiles para almacenar valores que necesitan ser compartidos entre diferentes funciones.

Es importante tener en cuenta que si se modifica una variable global dentro de una función, la modificación será reflejada en todas las partes del programa que usen esa variable global. Esto puede ser conveniente en algunas situaciones, pero también puede llevar a errores si no se manejan adecuadamente.

Por esta razón, se recomienda tener cuidado al usar variables globales y tratar de mantener su uso al mínimo necesario. En su lugar, se deben utilizar variables locales dentro de las funciones siempre que sea posible, para evitar conflictos y errores en el código.

4.2 VARIABLES GLOBALES.

```
x = 10

def mi_funcion():
    global x
    x += 5
    print("El valor de x dentro de la funcion es :", x)

mi_funcion() #el valor de x dentro de la funcion es : 15
```

En este ejemplo, definimos una variable global x con un valor inicial de 10. Luego, definimos una función mi_funcion() que utiliza la variable global x. Para poder modificar el valor de la variable global dentro de la función, usamos la palabra clave global. Dentro de la función, aumentamos el valor de x en 5 y luego imprimimos el valor actualizado. Finalmente, llamamos a la función y vemos el valor actualizado de x. Después de llamar a la función, imprimimos el valor de x fuera de la función para demostrar que ha sido actualizado.

4.2 VARIABLES GLOBALES.

En resumen, una variable global en una función de Python es una variable que se define fuera de la función y se puede acceder desde cualquier lugar dentro del programa. Cuando se modifica el valor de una variable global dentro de una función, el cambio se reflejará en todas las partes del programa que utilizan esa variable. Es importante tener cuidado al usar variables globales para evitar errores y asegurarse de que el código sea fácil de entender y mantener.

FUNCIONES ANIDADAS Y RECURSIVIDAD EN PYTHON.

5.1 FUNCIONES ANIDADAS.

Las funciones anidadas son simplemente una función dentro de otra función. Esto puede ser útil para separar la lógica del código en módulos más pequeños y manejables. Por ejemplo:

En este ejemplo, `funcion_secundaria` es una función anidada dentro de `funcion_principal`. Al llamar a `funcion_principal`, también se ejecutará la función `funcion_secundaria`.

```
def funcion_principal():
    def funcion_secundaria():
        print("Esta es una función anidada")
    funcion_secundaria()
```

5.1 FUNCIONES ANIDADAS.

En este ejemplo, la función calcular_edad contiene dos funciones anidadas: obtener_anio_actual y calcular_edad_actual.

```
def calcular_edad(nombre, anio_nacimiento):

    def obtener_anio_actual():
        return 2023

    def calcular_edad_actual(anio_actual):
        return anio_actual - anio_nacimiento

    edad = calcular_edad_actual(obtener_anio_actual())
    return f"{nombre} tiene {edad} años"

yo = calcular_edad("juan",1999)
print(yo)
```

La función obtener_anio_actual simplemente retorna el año actual, mientras que la función calcular_edad_actual calcula la edad actual restando el año de nacimiento del año actual.

La función principal, calcular_edad, llama a ambas funciones anidadas para calcular la edad de la persona y devuelve un mensaje con el nombre y la edad.

5.2 FUNCIONES RECURSIVAS.

La recursion en funciones se manda a llamar la misma funcion adentro de la funcion.

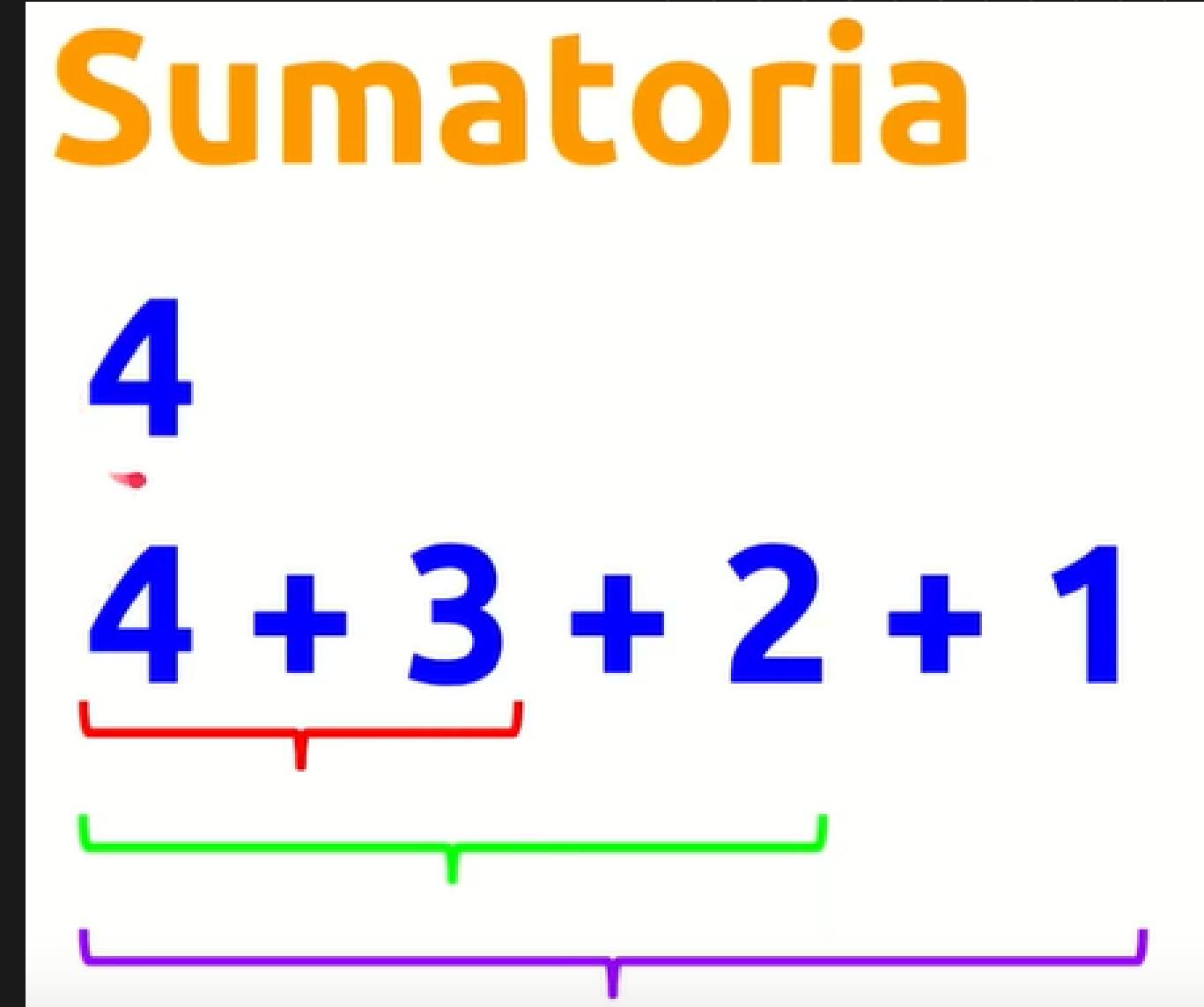
Vamos a trabajar el ejemplo de una sumatoria

Comenzamos escribiendo el 4 y le vamos a ir sumando hacia atras hasta llegar al 1 , este uno sera el encargado de detener la funcion

```
#comprendiendo La recursividad

def sumatoria(num):
    if num==1:
        return 1
    else:
        return num+sumatoria(num-1)

num =int(input("numero de la sumatoria"))
print(sumatoria(num))
```



5.2 FUNCIONES RECURSIVAS.

Explicaremos paso a paso que estamos haciendo.

1-definimos nuestra funcion , en este caso def sumatoria(num): y va a depender de la variable que la llamaremos num.

2-Vamos a poner la condicion que anteriormente mencione,

If ==1:

 return 1

cuando se llegue al 1 la funcion terminara regresara en 1.

3-Despues en caso de que no sea 1 , lo que hara es regresar el numero + la funcion que nombramos como sumatoria y pondremos que numero es -1
Esta resta lo que hara es que vayamos hacia atras sumando los valores , entonces en cuanto llegue al 1 terminara la funcion.

 return num + sumatoria(num-1)

Esto es lo que pareceria como funciona un ciclo while

5.2 FUNCIONES RECURSIVAS.

EJEMPLO PRACTICO

Si yo te preguntara como harias para contar cuantas paginas tienen todos tus libros juntos ,y te pido que me definas todos los pasos de ese proceso.
Tal vez tu solucion seria algo asi como en lo siguiente.

Para encontrar el total de paginas en estos libros

1-voy al primer libro y tomo cuantas paginas tiene "50" y lo guardo en una variable total. Total = 50

2-Ahora voy al segundo y sumo sus paginas al total, que nos quedaria asi
TOTAL = 150 y asi sucesivamente.

Esto se podria hacer
simplemente con un ciclo for.

```
1 libros = [50, 100, 150, 70, 250]
2
3 total = 0
4 for libro in libros:
5     total += libro
6
7 print(total)
```

5.2 FUNCIONES RECURSIVAS.

Pero hay una manera mucho mas facil de hacer mas eficiente, usando la recursividad , por que utilizando el for tenemos que iterar encambio la recursividad lo hace automaticamente. [#comprendiendo La recursividad](#)

```
def suma_naturales(lista):
```

```
if len(lista) == 1:
```

```
return lista[0]
```

else:

```
return lista[0] + suma_naturales(lista[1:])
```

```
numero = [50,100,150,200,250]
```

```
resultado = suma_naturales(numero)
```

```
print(resultado)
```

ERORIFI

OUTPUT

DFRUG CONSOL

TERMINAL

Python + ⌂ ⌄ ⌁ ⌂ ⌁

```
PS C:\Users\Miguel-PC\OneDrive\Escritorio\folders\KALI\practicapython> & C:/Users/Miguel-PC/AppData/Local/Programs/Python/Python311/python.exe c:/Users/Miguel-PC/OneDrive/Escritorio/folders/KALI/practicapython/practica.py  
750  
PS C:\Users\Miguel-PC\OneDrive\Escritorio\folders\KALI\practicapython>
```

FUNCIONES PREDEFINIDAS DE PYTHON.

6.1 USO DE FUNCIONES PREDEFINIDAS.

Las funciones predefinidas en Python son funciones integradas en el lenguaje que puedes utilizar sin tener que definirlas previamente. Estas funciones están disponibles en Python sin necesidad de importar bibliotecas externas. Por ejemplo, la función `len()` se utiliza para obtener la longitud de un objeto, como una cadena de caracteres o una lista. Para utilizar esta función, simplemente la llamas y le pasas el objeto como argumento:

```
cadena = "Hola, mundo!"  
longitud = len(cadena)  
print(longitud)
```

6.1 USO DE FUNCIONES PREDEFINIDAS.

En este ejemplo, la función len() devuelve la longitud de la cadena de caracteres "Hola, mundo!" y la guarda en la variable longitud. Luego, se imprime el valor de longitud en la consola.

Otro ejemplo de función predefinida es sum(), que se utiliza para sumar los elementos de una lista:

```
numeros = [1, 2, 3, 4, 5]
suma = sum(numeros)
print(suma)
```

6.1 USO DE FUNCIONES PREDEFINIDAS.

En este caso, la función `sum()` recibe la lista `numeros` como argumento y devuelve la suma de sus elementos, que se guarda en la variable `suma`. Luego, se imprime el valor de `suma` en la consola.

Existen muchas funciones predefinidas en Python que puedes utilizar para simplificar tus programas. Algunas de las más comunes son `print()`, `input()`, `str()`, `int()`, `float()`, `bool()`, `max()`, `min()`, `range()`, `abs()`, `round()`, entre otras. Es importante mencionar que también puedes definir tus propias funciones en Python para realizar tareas específicas que necesites y poder reutilizarlas en tus programas.

FUNCIONES PREDEFINIDAS DE PYTHON.

7.1 UTILIZACIÓN DE BIBLIOTECAS Y MÓDULOS.

En Python, una biblioteca o módulo es un conjunto de funciones y objetos que se pueden importar en un programa para realizar tareas específicas. Estas bibliotecas están diseñadas para hacer que la programación en Python sea más fácil y eficiente.

Para utilizar una biblioteca o módulo en un programa de Python, primero debe importarse en el archivo de código fuente. Hay varias formas de hacer esto, pero la forma más común es usar la instrucción `import`. Por ejemplo, para utilizar el módulo de matemáticas de Python, puede importarlo en el siguiente código:

```
import math

print(math.pi) # Imprime el valor de pi
```

7.1 UTILIZACIÓN DE BIBLIOTECAS Y MÓDULOS.

En este ejemplo, la biblioteca de matemáticas de Python se importa en el programa con la instrucción "import math". A continuación, se puede acceder a las funciones y constantes del módulo utilizando el prefijo "math.". Hay muchas bibliotecas y módulos disponibles en Python que pueden ser utilizados para realizar una amplia gama de tareas. Algunas bibliotecas populares incluyen:

- Numpy: biblioteca de cálculo numérico que se utiliza para trabajar con matrices y vectores.
- Pandas: biblioteca utilizada para el análisis de datos y la manipulación de datos tabulares.
- Matplotlib: biblioteca de visualización de datos para la creación de gráficos y diagramas.
- Scikit-learn: biblioteca utilizada para el aprendizaje automático y la minería de datos.

7.2 EJEMPLOS PRÁCTICOS DE BIBLIOTECAS Y MÓDULOS EN PYTHON.

En este ejemplo, importamos numpy como np. Luego creamos dos matrices x y y utilizando la función array de numpy. Por último, sumamos ambas matrices utilizando el operador + de numpy y guardamos el resultado en la matriz z. Finalmente, imprimimos z en la consola.

```
import numpy as np
x = np.array([1, 2, 3])
y = np.array([4, 5, 6])
z = x + y
print(z)
```

En este ejemplo, importamos numpy como np. Luego creamos dos matrices x y y utilizando la función array de numpy. Por último, sumamos ambas matrices utilizando el operador + de numpy y guardamos el resultado en la matriz z. Finalmente, imprimimos z en la consola.

Este es solo un ejemplo muy simple, pero numpy tiene muchas más funciones y características que te permiten realizar cálculos más complejos y sofisticados.

BUENAS PRÁCTICAS EN LA PROGRAMACIÓN DE FUNCIONES EN PYTHON.

8.1 NOMENCLATURA DE FUNCIONES.

La nomenclatura de funciones en Python se refiere a las convenciones de nombrar funciones de manera que sean fáciles de leer y entender para otros programadores.

Algunas reglas generales de la nomenclatura de funciones en Python incluyen:

- Las funciones deben tener nombres en minúsculas y, si el nombre consta de múltiples palabras, estas deben separarse por guiones bajos.
- El nombre de la función debe ser lo más descriptivo posible para indicar su función.
- Se recomienda que el nombre de la función comience con un verbo, ya que las funciones en Python son acciones que realizan una tarea específica.
- Si la función es parte de un módulo, puede ser útil incluir el nombre del módulo como prefijo en el nombre de la función, separado por un guión bajo.

8.1 NOMENCLATURA DE FUNCIONES.

Un ejemplo de función con una buena nomenclatura sería "calcular_precio_total()" que indica claramente la tarea que realiza la función. Es importante seguir estas convenciones para que otros programadores puedan leer y entender fácilmente el código que escribimos.

8.2 DOCUMENTACIÓN DE FUNCIONES.

La documentación de funciones en Python se refiere a la práctica de incluir información adicional sobre la función en el código, para ayudar a los programadores que la utilizan a entender su uso y comportamiento. Esta información adicional puede incluir una descripción de lo que hace la función, los parámetros que acepta, el tipo de valores que espera y los valores que devuelve. La documentación de funciones en Python se hace utilizando "docstrings", que son cadenas de texto que se colocan inmediatamente después de la definición de la función y antes del cuerpo de la función. Estos docstrings pueden ser accesibles mediante el uso de la función `help()` o leyendo el código fuente. Es una buena práctica documentar las funciones en Python para que otros programadores puedan entender cómo se usan y para ayudar en el mantenimiento del código. Además, muchos editores de código y entornos de desarrollo integrados (IDE) pueden mostrar la documentación de la función en una ventana emergente o como parte de una función de autocompletado, lo que facilita el trabajo con las funciones.

A continuación, se presenta un ejemplo sencillo de cómo documentar una función en Python utilizando docstrings:

8.2 DOCUMENTACIÓN DE FUNCIONES.

```
def suma(a, b):
    """
    Esta función toma dos números como argumentos y devuelve su suma.
    Los argumentos deben ser números enteros o de punto flotante.
    """
    return a + b
```

En este ejemplo, se utiliza un docstring de varias líneas para proporcionar información sobre la función. El docstring incluye una descripción de lo que hace la función, los parámetros que acepta y el tipo de valores que espera.

CONCLUSIONES Y RECOMENDACIONES FINALES.

9.1 RECOMENDACIONES PARA EL USO DE FUNCIONES EN PYTHON.

hay varios consejos de programadores profesionales sobre el uso de funciones en Python. Algunos de los más importantes son:

1. Mantener las funciones cortas y simples: las funciones deben hacer una sola cosa y hacerla bien. Si una función es demasiado larga o compleja, puede ser difícil de entender y mantener.
2. Usar nombres de funciones descriptivos: los nombres de las funciones deben ser descriptivos y explicar claramente lo que hace la función.
3. Documentar las funciones: es importante documentar las funciones para que otras personas puedan entender lo que hace la función y cómo se usa.
4. Usar argumentos por defecto: si una función tiene argumentos opcionales, es una buena práctica proporcionar valores predeterminados para estos argumentos. Esto hace que sea más fácil usar la función y reduce la cantidad de código necesario.

9.1 RECOMENDACIONES PARA EL USO DE FUNCIONES EN PYTHON.

- 7.Evitar el uso de variables globales: las variables globales pueden causar problemas de legibilidad y mantenimiento. Es mejor pasar cualquier variable necesaria como argumento a la función.
- 8.Usar el retorno de valores: es importante que las funciones devuelvan un valor si es posible, para que se puedan utilizar en otras partes del programa.
- 9.Usar la modularidad: las funciones deben ser escritas de manera que puedan ser utilizadas en diferentes partes del programa. Esto hace que el código sea más modular y fácil de mantener.
Estos son solo algunos consejos básicos