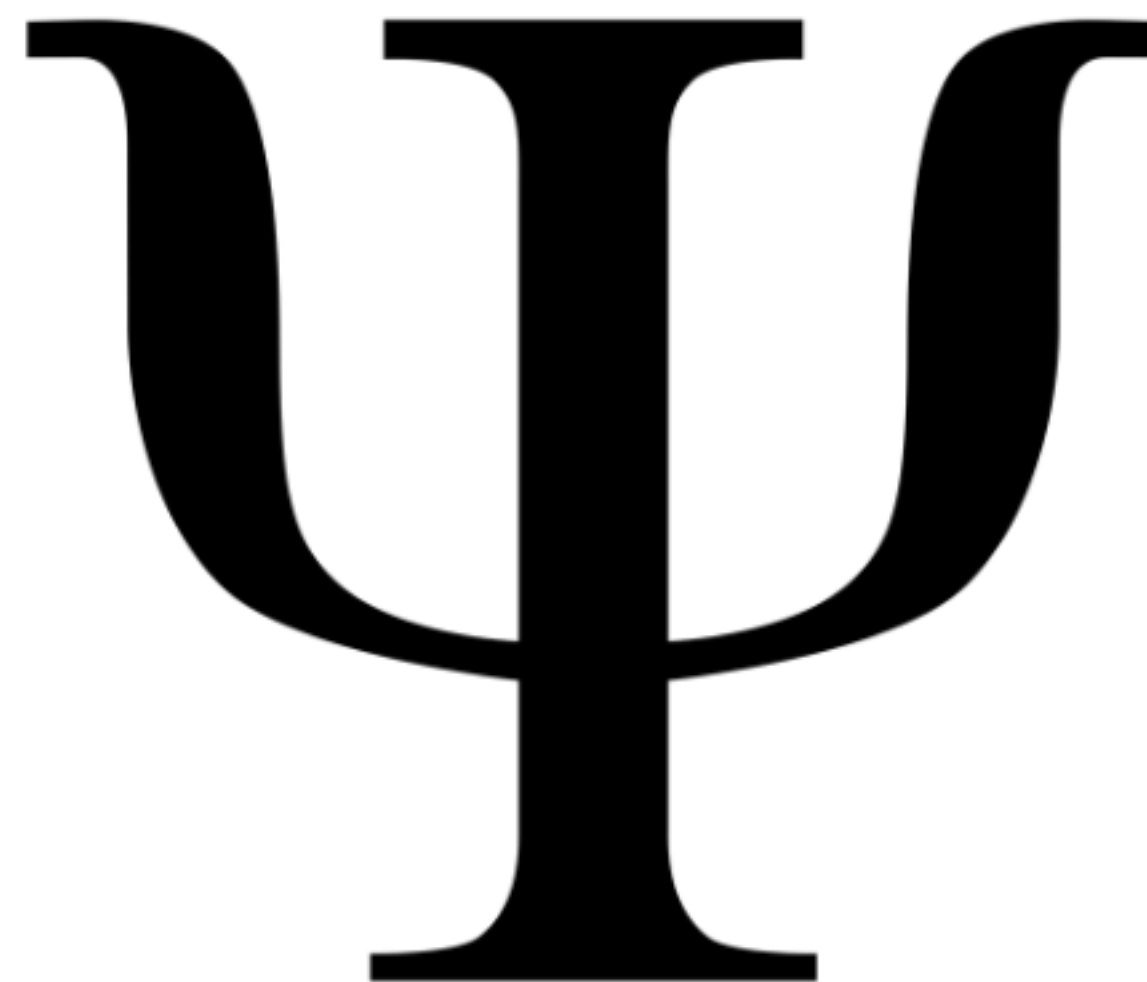




# Reutilización de Código en Python

## Introducción

- El desarrollo moderno exige soluciones modulares, reutilizables y fáciles de mantener.
- Python facilita este enfoque mediante:
  - Funciones definidas por el usuario (def)
  - Organización modular (import)
  - Expresiones lambda para operaciones breves
- La correcta aplicación de estas herramientas mejora la claridad, escalabilidad y productividad.

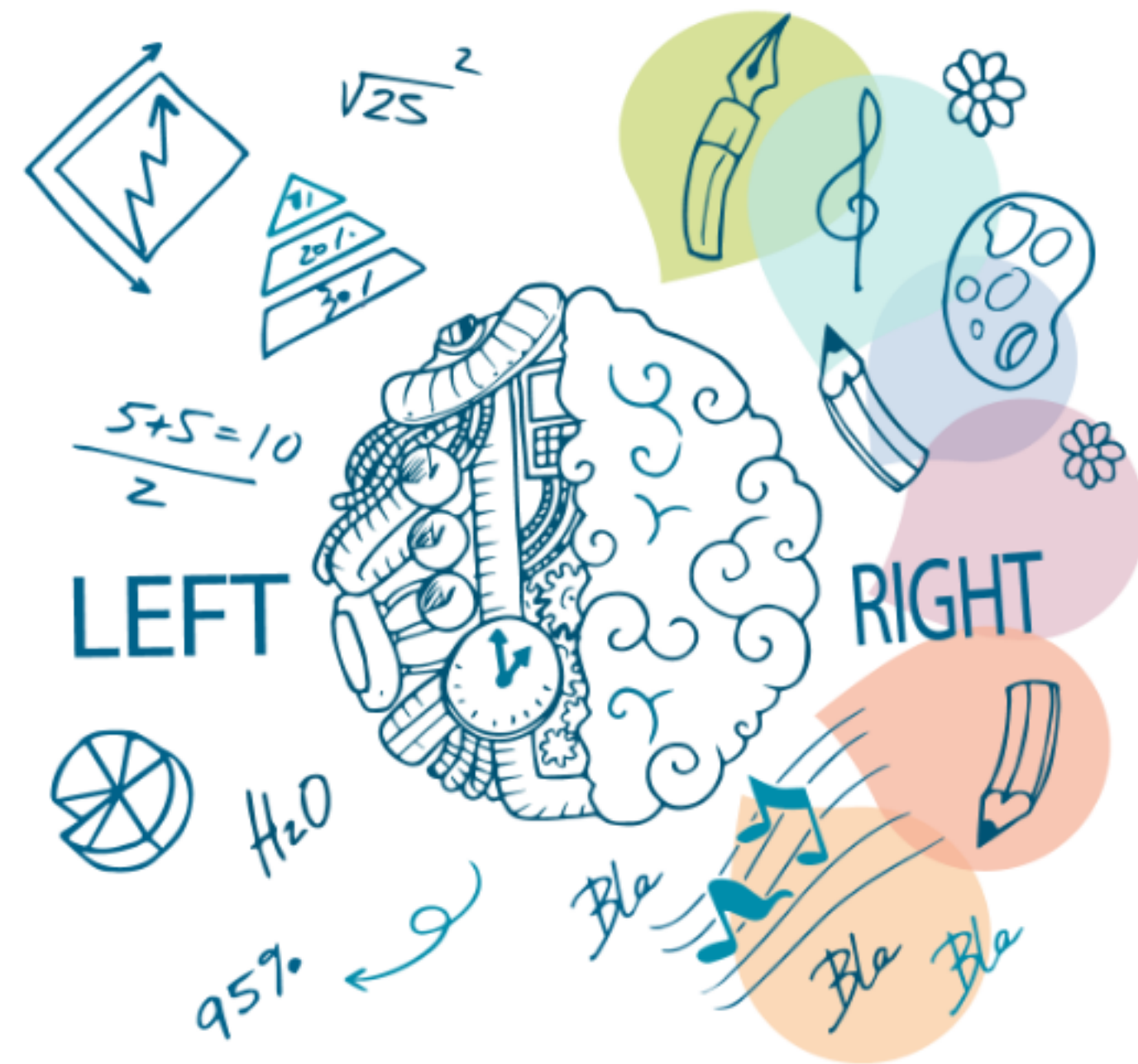


## ¿Qué es una Función?

- Una función es un bloque de código que realiza una tarea específica.
- Se define usando def seguido del nombre, parámetros, y cuerpo:

```
def saludar(nombre):  
    return f"Hola, {nombre}!"
```

- Las funciones mejoran la legibilidad, reducen la redundancia y permiten la reutilización.



# Ejemplo Básico de Función

## Ejemplo Básico de Función

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

```
resultado = suma(10, 15)
print(f"Resultado: {resultado}")
```

◆ Salida esperada:  
Resultado: 25

- Puede tener uno o varios parámetros.
- Puede retornar un valor o ejecutarse sin retorno explícito.





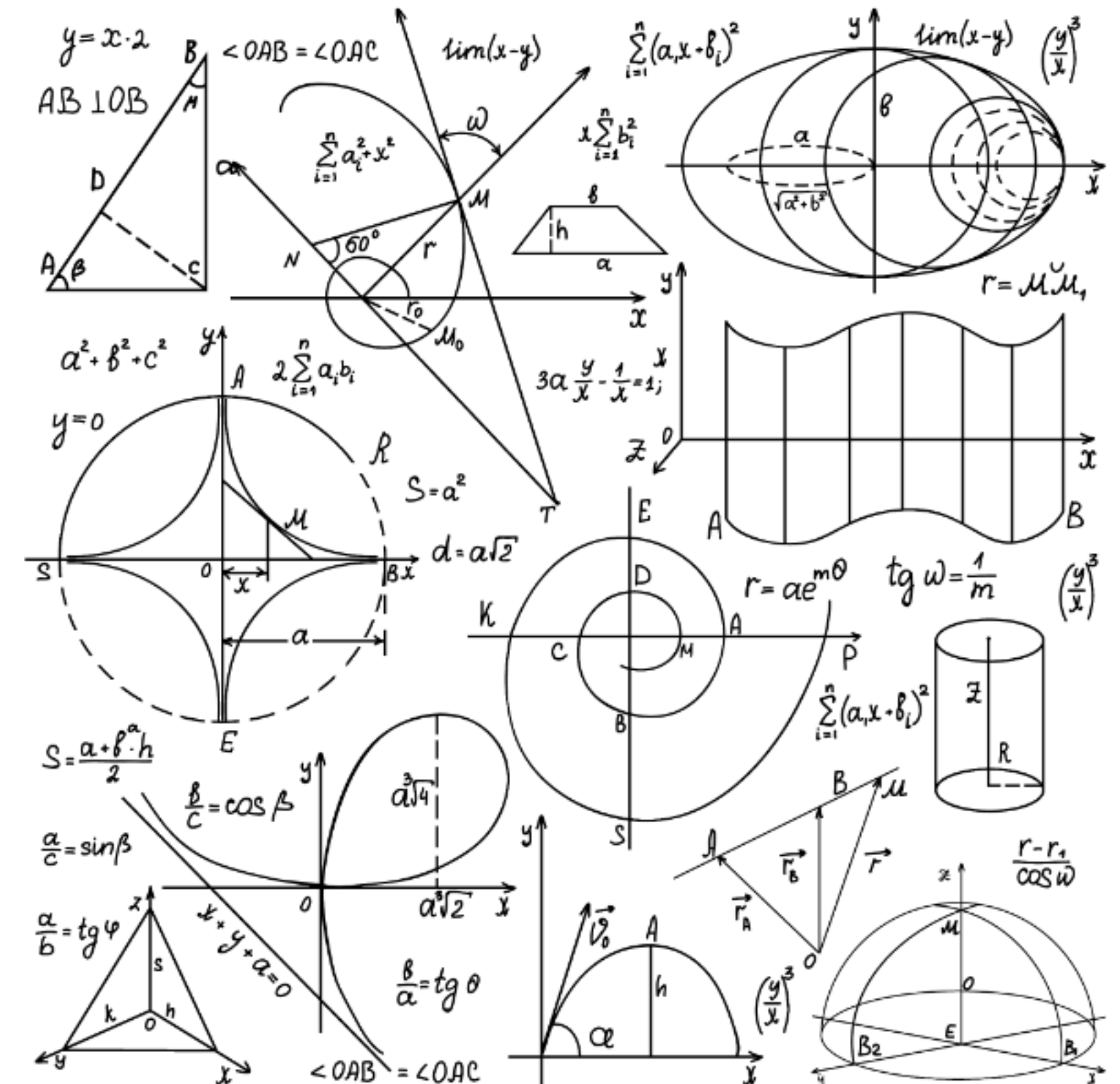
## Importación y Uso de Módulos

- Un módulo es un archivo .py que contiene funciones reutilizables.
- Se importa con import:

```
import math
```

```
print(math.sqrt(36)) # Salida: 6.0
```

- Permite mantener el código organizado y cumplir con principios como SRP (Responsabilidad Única).



## Scope: Ámbito de Variables

- **Ámbito Local:** dentro de una función, solo visible ahí.
- **Ámbito Global:** fuera de funciones, visible en todo el script.

**x = 10**

```
def mostrar():
```

$$y = 5$$

```
print(x, y)
```

- Una mala gestión del scope puede causar errores o colisiones de nombres.





## Funciones como Módulos

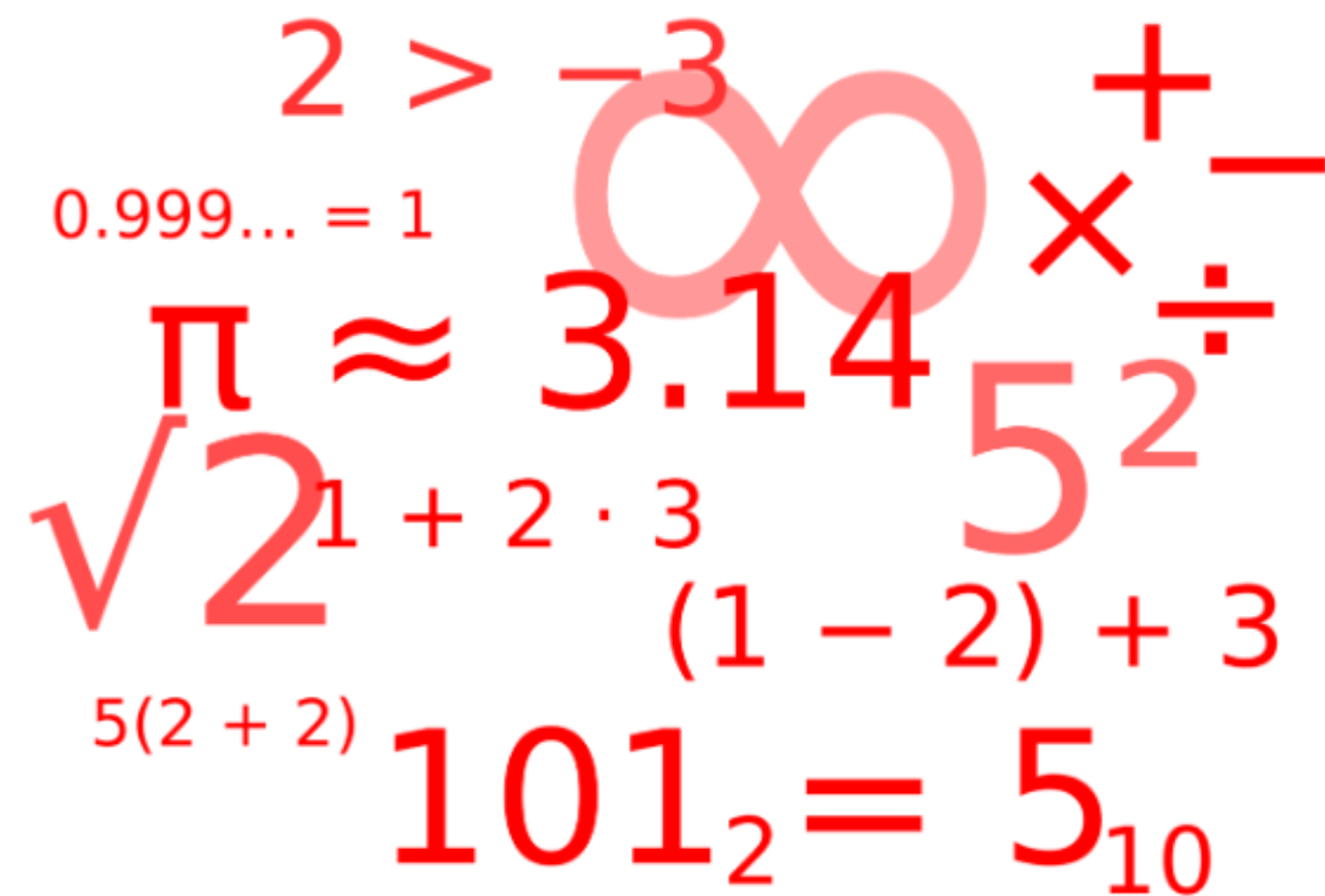
- Las funciones pueden organizarse en archivos separados para reutilizar en otros programas.

# operaciones.py

```
def dividir(a, b):  
    return a / b if b != 0 else "Error"
```

# main.py

```
import operaciones  
print(operaciones.dividir(10, 2)) # Salida: 5.0
```



2 > -3  
0.999... = 1  
 $\pi \approx 3.14$   
 $\sqrt{2}$   
5(2 + 2)  
 $101_2 = 5_{10}$   
+  
-  
x  
÷  
5<sup>2</sup>  
(1 - 2) + 3

## Expresiones Lambda

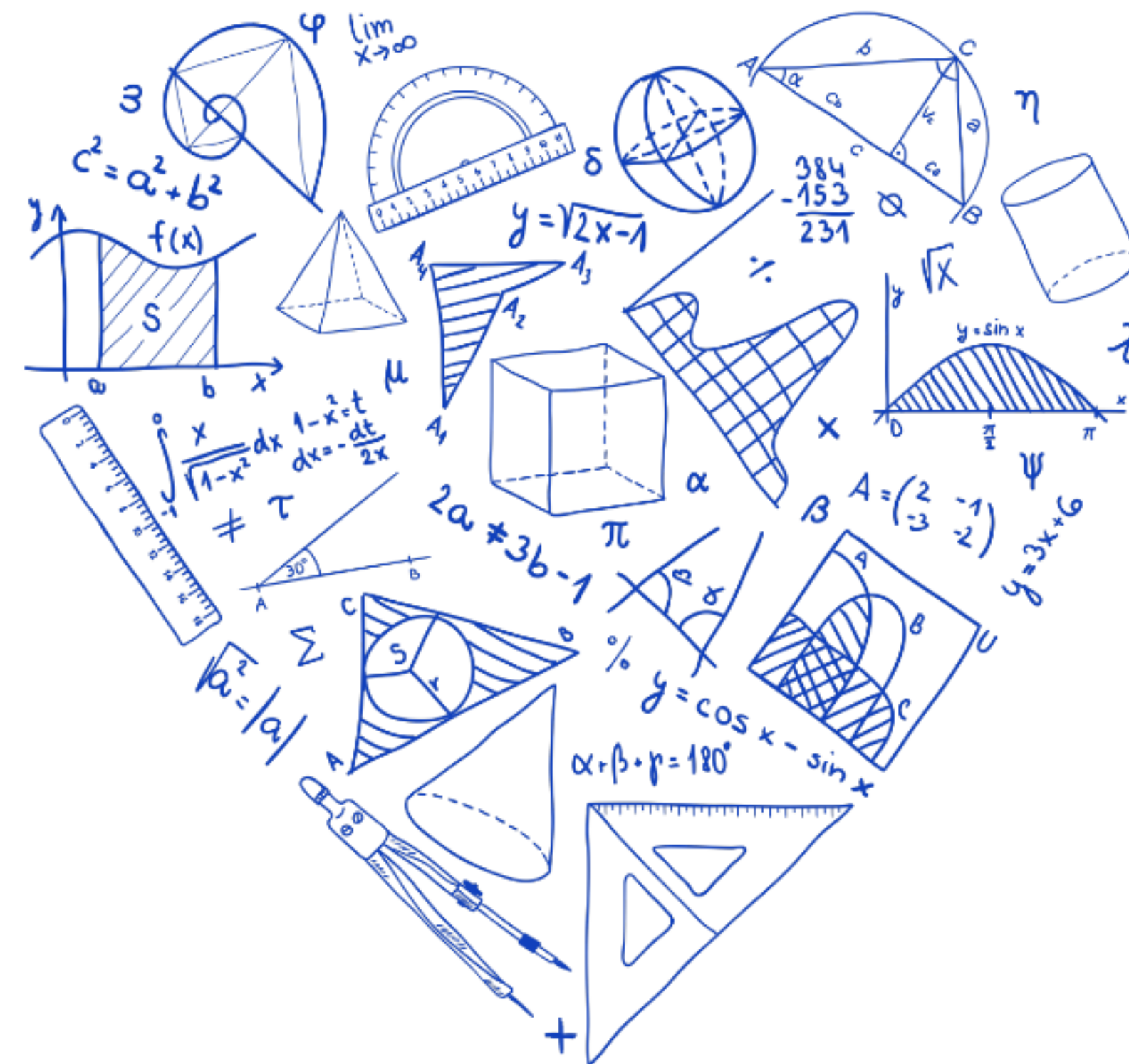
- Una lambda es una función anónima de una sola línea:

```
doble = lambda x: x * 2  
print(doble(4)) # Salida: 8
```

- Útil con `map()`, `filter()`, `reduce()`:

```
cuadrados = list(map(lambda x: x**2, [1,2,3]))
```

- Ideal para funciones breves, sin necesidad de nombre.



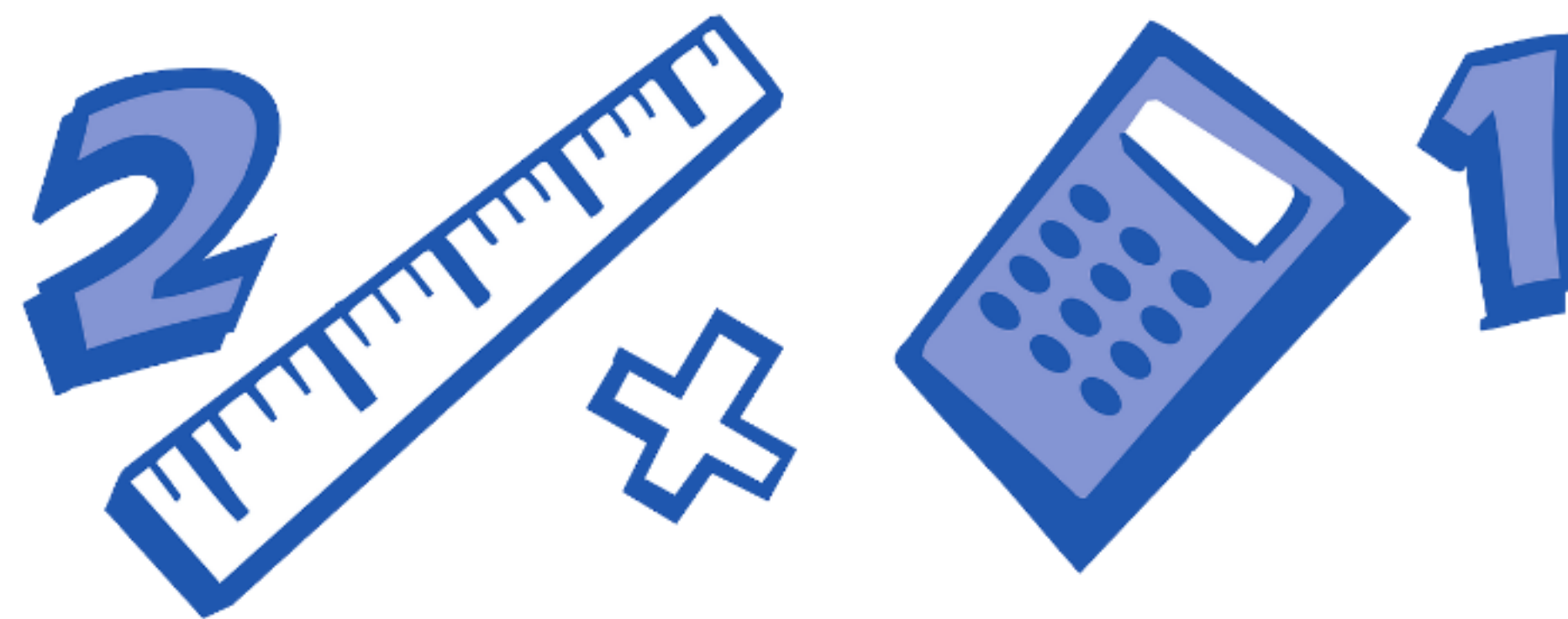


## Caso de Uso Combinado

- En un flujo real, es común combinar módulos, funciones y lambdas:

```
# estadisticas.py  
def promedio(lista):  
    return sum(lista) / len(lista)
```

```
# programa principal  
import estadisticas  
datos = [8.5, 9.0, 6.7]  
media = estadisticas.promedio(datos)  
pares = list(filter(lambda x: x % 2 == 0, range(10)))
```



## Beneficios del Enfoque Modular

- **Claridad:** código más legible
- **Reutilización:** funciones reutilizables en múltiples scripts
- **Mantenibilidad:** cambios en un módulo no afectan al resto
- **Escalabilidad:** estructura profesional y limpia



## Conclusión

- Las funciones son más que sintaxis: representan lógica estructurada.
- Lambdas y módulos complementan la eficiencia del desarrollo.
- Dominar estas herramientas te permite crear software más **limpio, profesional y sostenible**.





