Introduction_to_Python

June 13, 2024

Autor: Gustavo Alonso Gomez Morales

1 Estado del Arte

1.1 Google Colab

Google Colab, o "Colaboratory", es una plataforma de desarrollo basada en la nube que permite a los usuarios escribir y ejecutar código Python en un navegador sin ninguna configuración previa. Es un entorno que se basa en Jupyter Notebook, una herramienta ya establecida y muy popular entre ingenieros, científicos de datos, investigadores y educadores que necesitan combinar código, texto narrativo y visualizaciones en un solo documento interactivo. La integración de Google Colab con Jupyter es profunda, ya que Colab utiliza la infraestructura de Jupyter para ofrecer una interfaz de usuario familiar y accesible que facilita la experimentación, el análisis de datos y la colaboración entre usuarios.

1.2 Python

Creado por Guido van Rossum y lanzado por primera vez en 1991, es uno de los lenguajes de programación más populares y de rápido crecimiento en el mundo. Este crecimiento se debe en gran parte a su diseño, que enfatiza la legibilidad del código, la simplicidad y la sintaxis que permite a los programadores expresar conceptos en menos líneas de código en comparación con otros lenguajes como C++ o Java. Python es altamente extensible, lo que permite a los usuarios y desarrolladores añadir módulos funcionales desarrollados en otros lenguajes de programación, aumentando así su funcionalidad y eficiencia. El principal objetivo detrás de su creación era desarrollar un lenguaje de programación que fuera tanto fácil de entender como de leer, haciendo hincapié en la legibilidad del código y reduciendo el coste de mantenimiento del programa. Van Rossum quería un lenguaje que pudiera ser usado por programadores profesionales y no profesionales por igual, y que permitiera a los usuarios escribir código claro para proyectos pequeños y grandes.

El verdadero poder de Python radica en su extensa variedad de bibliotecas y frameworks, que ha sido un factor importante en su adopción en una amplia gama de disciplinas desde la web y desarrollo de software hasta la ciencia de datos, la inteligencia artificial y robótica. Bibliotecas como NumPy y Pandas han transformado la manipulación de datos y las operaciones matemáticas (básicas y complejas) asi como Matplotlib es una excelente herramienta para la visualizacion de estos datos.

Python ha hecho incursiones significativas tanto en la industria como en el mundo académico. En la industria, desde startups hasta gigantes tecnológicos como Google y Facebook, Python se utiliza para una variedad de aplicaciones, desde el desarrollo web hasta el análisis de sistemas complejos. Su capacidad para integrarse con otras tecnologías y su facilidad de uso lo convierten

en una elección popular para prototipos rápidos y desarrollo de productos a escala. En el ámbito académico, Python es frecuentemente el lenguaje de programación preferido para la investigación computacional y la enseñanza de cursos de informática debido a su sintaxis clara y su poderosa funcionalidad. Universidades de todo el mundo han adoptado Python para enseñar conceptos de programación, análisis de datos, y machine learning, entre otros, debido a su sencillez y la profundidad de sus bibliotecas disponibles.

2 Alcance de Python

Facilidad de Uso y Flexibilidad Su sintaxis clara y legible permite que los ingenieros y científicos, que quizás no son programadores expertos, puedan implementar y probar rápidamente algoritmos de control sin tener que lidiar con la complejidad de lenguajes de programación más bajos como C o C++.

Amplia Gama de Bibliotecas Existen numerosas bibliotecas en Python que son especialmente útiles en el campo del control, como: - **control y control.matlab**: es un paquete de Python que implementa operaciones básicas para el análisis y diseño de sistemas de control realimentados. Su modulo control.matlab permite escribir con la sintaxis de MATLAB en cualquier entorno que use Python. Este modulo resulta muy util para programadores que usan MATLAB. - **NumPy y SciPy**: son librerías especializadas en operaciones matemáticas y cálculos científicos. - **Matplotlib**: Para visualización de datos, fundamental para el análisis de comportamientos de sistemas. - **Pandas**: es la librería mas famosa en manejo eficiente de grandes conjuntos de datos. - **Scikit-learn y TensorFlow**: Para implementar técnicas de aprendizaje automático en el control predictivo y adaptativo de sistemas.

Simulación y Modelado Python es ampliamente utilizado en la simulación y el modelado de sistemas de control debido a su capacidad para integrar y manipular diferentes tipos de datos y algoritmos rápidamente. Herramientas como SimPy (simulación en procesos) y PyDSTool (análisis dinámico de sistemas) son ejemplos de cómo Python facilita la modelación y análisis de sistemas complejos en control.

Interfaz con Hardware Python también puede interfazarse directamente con hardware a través de bibliotecas como PySerial o PyVISA para la automatización y el control de dispositivos de medición y actuadores. Esto es esencial en áreas donde el control en tiempo real y la recolección de datos son cruciales.

Integración y Automatización Python se integra bien con otras plataformas y tecnologías, lo que permite automatizar completamente los sistemas de control y los flujos de trabajo de datos. Además, la capacidad de Python para funcionar en varios sistemas operativos y su integración con sistemas embebidos y microprocesadores (como Raspberry Pi) lo hace ideal para proyectos de IoT (Internet de las Cosas) relacionados con control y automatización.

Educación y Comunidad La vasta comunidad de Python significa que existe un gran número de recursos de aprendizaje y foros para solucionar problemas. Esto es especialmente útil en campos técnicos como el control, donde compartir conocimientos y soluciones puede acelerar significativamente el desarrollo de proyectos.

3 Introducción

Python es un lenguaje de programación de alto nivel y de propósito general. Es popular por su sintaxis simple y legible, lo que lo hace fácil de aprender y usar. Cuenta con una amplia gama de librerías y módulos disponibles, lo que permite su uso en una variedad de aplicaciones, desde Data Science y Machine Learning hasta desarrollo web y automatización de tareas. Es un lenguaje de código abierto y multiplataforma, lo que significa que puede ser utilizado en diferentes sistemas operativos.

3.0.1 Aspectos generales:

- Todas las variables en Python están asociadas a sus respectivas clases y pueden ser **mutables** o **inmutables**. Las variables **mutables** permiten la variación de su valor una vez asignado y no se modifica su ubicación en la memoria RAM cada vez que se usa. Por el contrario, las variables **inmutables** no se les puede cambiar su valor una vez asignado y se crea una nueva ubicación en la memoria RAM cada vez que se usa.
- Se recomienda emplear **snake case** para nombrar las variables en Python. Esto significa que se utilizan guiones bajos para separar palabras dentro de un nombre. Por ejemplo, podemos nombrar nuestras variables como: mi_variable, funcion_transferencia, vector_tiempo, senal_cosenoidal.
- Los valores de las variables se pueden mostrar en pantalla con el comando **print()**, haciendo uso de **f-string (f' ')** para facilitar la interpolación de variables dentro de una cadena. Esto permite que sea más sencillo y legible insertar variables en una cadena. Además, la aparición de `\n` en una cadena representa un salto de línea en ésta.
- Con el atajo **Ctrl** + **Space** activamos la ayuda de sugerencias de autocompletado que brinda el entorno colab. Esto significa que se mostrará una lista con las sugerencias asociadas a lo que se está escribiendo, lo cual nos permite ahorrar tiempo y esfuerzo al momento de programar. Esto hace que la escritura del código sea mucho más ágil y eficiente.
- Cuando se presiona **Ctrl** + **Enter**, la celda actual se ejecutará y se mostrarán los resultados. Esta es una forma rápida de probar y ejecutar código sin necesidad de ejecutar todo el código del documento.

---

[47]:
```
print("Hello world")
```

```
Hello world
```

3.1 Librerías y sus métodos

Las librerías son "complementos" que permiten añadir funciones a Python que no vienen en la interfaz por defecto.

Los nombres "np" y "plt" son abreviaciones (apodos) de las librerías y se pueden llamar como tú prefieras. Por convención, se suele usar estás abreviaciones para fácilitar la escritura del código. Vamos a importar las librerias de interes para el curso de la siguiente manera:

[48]:
```
import numpy as np                  # Importamos la librería Numpy
import matplotlib.pyplot as plt     # Importamos la libreria MatPlotLib
import scipy as sp                  # Importamos la libreria SciPy
```

Cada librería tiene un catálogo de códigos que nos permiten realizar diversas funciones y trabajar en conjunto con otras librerías para aprovechar el entorno de Python.

Para llamar, invocar o usar una función de una librería tenemos que colocar el identificador (o apodo) primero, un punto (.) y luego el nombre de esta

La función "arange" de numpy permite crear un arreglo de números (vectores). Esta función tiene la siguiente estructura:

`arange([inicio],[final],[paso])`

Donde el inicio y el final marcan los límites del arreglo y el paso determina la cantidad de muestras que se van a tomar entre estos límites.

[49]:
```
# Prueba correrlo, cambia los parámetros y mira el resultado!

t = np.arange(0,10,2);
print(f't = {t}')
```

```
t = [0 2 4 6 8]
```

La función "linspace" es muy parecida a la función "arange", ambas de numpy, la diferencia es que mediante la función linspace podemos crear arreglos de números cuyos datos están equiespaciados entre si. Esta función tiene la siguiente estructura:

`linspace([inicio],[final],[paso])`

Podemos observar que la estructura es igual a la de la función "arange", sin embargo, para el caso de linspace el paso marca la separación entre los datos.

[50]:
```
# Prueba cambiar los límites y el paso y mira lo que ocurre!

t = np.linspace(0,1000,5);
print (f't = {t}')
```

```
t = [   0.  250.  500.  750. 1000.]
```

La librería numpy también cuenta con funciones famosas como lo son:

[51]:
```
np.pi;    # El número pi
np.sin;   # La función seno
np.cos;   # La función coseno
```

Existen otras muchas funciones y complementos con los que cuenta esta librería

Veamos algunos comandos de la librería MatPlotLib:

- **plt.plot()**: Nos permite gráficar funciones de numpy u otras librerías
- **plt.grid()**: Cóloca una cuadrícula en la gráfica de las funciones

- **plt.xlim()**: Nos permite definir límites para la gráfica en el eje de las abcisas
- **plt.ylim()**: Nos permite definir límites para la gráfica en el eje de las ordenadas
- **plt.show()**: Nos permite visualizar varias funciones en una sola gráfica

3.2 Tipos de variables.

Entero (int): Estas variables almacenan números enteros, como -1, 0, 1, 2, etc.

[52]:
```
numero_entero = 4
print(f'Número int: {numero_entero}.\nClase de la variable:␣
  ↪{type(numero_entero)}')
```

```
Número int: 4.
Clase de la variable: <class 'int'>
```

Flotante (float): Estas variables almacenan números con decimales, como 3.14, -2.5, etc.

[53]:
```
numero_flotante = 3.1415
print(f'Número float: {numero_flotante}.\nClase de la variable:␣
  ↪{type(numero_flotante)}')
```

```
Número float: 3.1415.
Clase de la variable: <class 'float'>
```

Cadena o String (str): Estas variables almacenan cadenas de caracteres, como “Hello world”, “Python”, “I love programming in Python”, etc.

[54]:
```
cadena = "Soy una variable que almacena caracteres ;)"
print(f'Cadena str: {cadena}\nClase de la variable: {type(cadena)}')
```

```
Cadena str: Soy una variable que almacena caracteres ;)
Clase de la variable: <class 'str'>
```

Booleano (bool): Estas variables almacenan valores booleano, como **True o False**

[55]:
```
booleano = False
print(f'Boolean: {booleano}\nClase de la variable: {type(booleano)}')
```

```
Boolean: False
Clase de la variable: <class 'bool'>
```

Una variable **booleana** tambien se puede expresar como una operación lógica, comparación y evaluacíon

[56]:
```
mi_variable_booleana = 1 > 5
print(f'La clase de la variable es: {type(mi_variable_booleana)} la variables␣
  ↪es: {mi_variable_booleana}')
# Imprime False, ¡pruebalo!
```

```
La clase de la variable es: <class 'bool'> la variables es: False
```

3.3 Operadores y asignación de variables

3.3.1 Operadores aritméticos

Son símbolos especiales (+, -, *, /, //, %, **) que se utilizan para realizar operaciones matemáticas como suma, resta, multiplicación, división, entre otras.

[75]:
```
num1 = 11
num2 = 7

print(f'Operaciones aritmetica con los siguientes numeros: {num1} y {num2}')

# Suma
suma = num1 + num2
print(f'Resultado de la suma {num1} mas {num2} es {suma}')

# Resta
resta = num1 - num2
print(f'Resultado de la resta {num1} menos {num2} es {resta}')

# Multiplicación
multiplicacion = num1 * num2
print(f'Resultado de la multiplicación entre {num1} y {num2} es␣
 ↪{multiplicacion}')

# División
division = num1 / num2
print(f'Resultado de la división {num1} sobre {num2} es {division}')

# División entera
division_entera = num1 // num2
print(f'Resultado de la división entera {num1} sobre {num2} es␣
 ↪{division_entera}')

# Módulo
modulo = num1 % num2
print(f'Resultado del módulo (residuo) entre {num1} y {num2} es {modulo}')

# Exponente
exponenciacion = num1 ** num2
print(f'Resultado de la exponenciación {num1} elevado a {num2} es␣
 ↪{exponenciacion}')
```

```
Operaciones aritmetica con los siguientes numeros: 11 y 7
Resultado de la suma 11 mas 7 es 18
Resultado de la resta 11 menos 7 es 4
Resultado de la multiplicación entre 11 y 7 es 77
Resultado de la división 11 sobre 7 es 1.5714285714285714
Resultado de la división entera 11 sobre 7 es 1
```

```
Resultado del módulo (residuo) entre 11 y 7 es 4
Resultado de la exponenciación 11 elevado a 7 es 19487171
```

3.3.2 Operadores de asignación

Son símbolos (como =, +=, -=, *=, /=, entre otros) que se utilizan para asignar valores a variables y actualizar su valor al mismo tiempo con una operación específica.

[58]:
```
# Asignación de una variable
my_variable = 10
print(f'La variable inicial tiene un valor de {my_variable}\n')

# Asignación de incremento (my_variable + 7) [Reasignación]
my_variable += 7
# my_variable = my_variable + 7
print(f'El incremento fue de 7')
print(f'La asignación de incremento da como resultado: {my_variable} ya que se␣
 ↪incrementó 7\n')

# Asignación de disminución (my_variable - 2) [Reasignación]
my_variable -= 2
# my_variable = my_variable + 7
print(f'La disminucion fue de 2')
print(f'La asignación de disminución da como resultado: {my_variable} ya que␣
 ↪disminuyó 2\n')

# Asignación de multiplicación (my_variable * 2) [Reasignación]
my_variable *= 2
# my_variable = my_variable * 2
print(f'La asignación de multiplicacíon da como resultado: {my_variable} ya que␣
 ↪se multiplicó por 2')

# Asignación de división (my_variable / 3) [Reasignación]
# my_variable = my_variable / 3
my_variable /= 3
print(f'La asignación de división da como resultado: {my_variable} ya que se␣
 ↪dividió por 3')

# Asignación de división entera (my_variable // 4) [Reasignación]
my_variable //= 4
# my_variable = my_variable // 4
print(f'La asignación de división entera da como resultado: {my_variable} ya␣
 ↪que se dividió por 4')
```

```
La variable inicial tiene un valor de 10

El incremento fue de 7
La asignación de incremento da como resultado: 17 ya que se incrementó 7
```

```
La disminucion fue de 2
La asignación de disminución da como resultado: 15 ya que disminuyó 2

La asignación de multiplicacíon da como resultado: 30 ya que se multiplicó por 2
La asignación de división da como resultado: 10.0 ya que se dividió por 3
La asignación de división entera da como resultado: 2.0 ya que se dividió por 4
```

3.3.3 Asignación booleana

Implica asignar valores verdaderos o falsos (True o False) a variables según el resultado de una evaluación condicional (como ==, !=, >, >=, <, <=).

[59]:
```python
# Definimos los siguientes números
num1 = 3; num2 = 4

print(f'Los números en cuestion son: {num1} y {num2}')

booleano = num1 == num2
print(f'Resultado con el operador == es : {booleano}')

booleano = num1 != num2
print(f'Resultado con el operador != es : {booleano}')

booleano = num1 > num2
print(f'Resultado con el operador > es : {booleano}')

booleano = num1 >= num2
print(f'Resultado con el operador >= es : {booleano}')

booleano = num1 < num2
print(f'Resultado con el operador < es : {booleano}')

booleano = num1 <= num2
print(f'Resultado con el operador <= es : {booleano}')
```

```
Los números en cuestion son: 3 y 4
Resultado con el operador == es : False
Resultado con el operador != es : True
Resultado con el operador > es : False
Resultado con el operador >= es : False
Resultado con el operador < es : True
Resultado con el operador <= es : True
```

3.4 Sentencias condicionales

Son estructuras de control que permiten la ejecución de un código basado en una condición:

- **if:** Se utiliza para ejecutar un bloque de código si la condición evaluada es verdadera.
- **elif:** Se utiliza para evaluar varias condiciones y ejecutar un bloque de código si alguna de las condiciones es verdadera.
- **else:** Se utiliza para ejecutar un bloque de código si la condición es falsa.

[60]:
```
# Valor actual de la temperatura del sistema (podría provenir de un sensor)
temperatura_actual = 75

# Rango de temperatura deseado
temperatura_minima = 70
temperatura_maxima = 80

# Verifica si la temperatura está por debajo del mínimo
if temperatura_actual < temperatura_minima:
    print("La temperatura está por debajo del mínimo. Encender calentador.")
elif temperatura_actual > temperatura_maxima:
    print("La temperatura está por encima del máximo. Encender aire␣
 ↪acondicionado.")
else:
    print("La temperatura está dentro del rango. No se requiere acción.")
```

```
La temperatura está dentro del rango. No se requiere acción.
```

3.5 Ciclos

Los ciclos se usan para realizar un conjunto de acciones repetidamente bajo ciertas condiciones y parámetros.

3.5.1 Ciclo for

El ciclo **for** se usa para iterar sobre una secuencia objetos, como listas, tuplas. Los ciclos **for** se usan definiendo parámetros de iteración, así, veamos un ejemplo sencillo:

[61]:
```
colores = ['rojo', 'verde', 'azul', 'amarillo', "naranja"]
colores_invalidos = ['naranja', 'morado']

for color in colores:
    if color in colores_invalidos:
        print(f'{color}: ¡No es un color válido!')
    else:
        print(f'El color es: {color}')
```

```
El color es: rojo
El color es: verde
El color es: azul
El color es: amarillo
naranja: ¡No es un color válido!
```

Ahora, aquí tenemos un ejemplo sencillo de un código que utiliza un ciclo **for** junto con la función

range().

[76]:
```
# Definirmos el número de términos a sumar
n = 100

# Creamos una variable llamada suma y le asignamos el valor 0
suma = 0

# Ciclo for para iterar sobre una secuencia de números desde 1 hasta n
for i in range(1, n+1):
    suma += i

print(f'La suma de los primeros {n} números enteros es {suma}')
```

```
La suma de los primeros 100 números enteros es 5050
```

3.5.2 Ciclo while

Por otro lado, el ciclo while se usa para ejecutar repetidamente un conjunto de instrucciones bajo unos condicionales o de manera infinita (esto último no es muy práctico).

[63]:
```
# Veamos un ejemplo de como funcionan los ciclos while:

# importamos especificamente las funciones que vamos a usar
from numpy import arange, sin, pi, sqrt

# Generamos una onda seno con frecuencia de 100 Hz y amplitud de 1

fs = 100
t = arange(0, 1, 1/fs)
x = sin(2*pi*2*t)

# Vamos a calcular el valor RMS de la señal usando un ciclo while

i = 0
N = len(x)
sum_cuadrada = 0

while i < N:
    sum_cuadrada += x[i]**2
    i += 1
rms = sqrt(sum_cuadrada/N)

print(f'El valor RMS de la señal es: {rms}')
```

```
El valor RMS de la señal es: 0.7071067811865476
```

3.6 Lista y tuplas

3.6.1 Listas

Una lista es un conjunto de elementos (variables). Las listas son mutables.

[64]:
```
# Nombres --> list ['Juan', 'Karla', 'Maria', 'Ricardo']
#                    (0)      (1)      (2)       (3)

print(' LISTAS (mutables) '.center(80,'-'))

name = ['Juan', 'Karla', 'Maria', 'Ricardo', 0, 100, 13.123, True]
print(f'''
Lista original: {name}
Elementos de la lista name[1:5] sin incluir el indice: {name[1:5]}
''')

# Cambiar un valor
name[3] = 'Ivone'
print(name)

# Iterar una lista
for i in name:
    print(f'Iterando... {i}')
else:
    print('No existen mas elementos en la lista\n')

# Saber la cantidad de elementos que contiene una lista
print(f'La cantidad de elementos de la lista es: {len(name)}\n')

# Agregar elementos al final de una lista
name.append('Lorenzo')
print(name)

# Insertar elemento en índice; desplaza los demás a la derecha
name.insert(1, 'Octavio')
print(name)

# Remover un elemento
name.remove('Maria')
print(name)

# Remover el ultimo valor agregado
name.pop()
print(name)

# Eliminar un indice (los demas elementos se mueven a la izquierda)
del name[0]
```

```
print(name)

# Eliminar todos los elementos de la lista
name.clear()
print(name)
```

```
------------------------------ LISTAS (mutables) ------------------------------

Lista original: ['Juan', 'Karla', 'Maria', 'Ricardo', 0, 100, 13.123, True]
Elementos de la lista name[1:5] sin incluir el indice: ['Karla', 'Maria',
'Ricardo', 0]

['Juan', 'Karla', 'Maria', 'Ivone', 0, 100, 13.123, True]
Iterando… Juan
Iterando… Karla
Iterando… Maria
Iterando… Ivone
Iterando… 0
Iterando… 100
Iterando… 13.123
Iterando… True
No existen mas elementos en la lista

La cantidad de elementos de la lista es: 8

['Juan', 'Karla', 'Maria', 'Ivone', 0, 100, 13.123, True, 'Lorenzo']
['Juan', 'Octavio', 'Karla', 'Maria', 'Ivone', 0, 100, 13.123, True, 'Lorenzo']
['Juan', 'Octavio', 'Karla', 'Ivone', 0, 100, 13.123, True, 'Lorenzo']
['Juan', 'Octavio', 'Karla', 'Ivone', 0, 100, 13.123, True]
['Octavio', 'Karla', 'Ivone', 0, 100, 13.123, True]
[]
```

3.6.2 Tuplas

Una tupla sigue guardando el orden de los elementos, al igual que las lista, pero estos elementos no se pueden modificar, eliminar o agregar (variable inmutable). Las funciones append(), remove(), insert(), etc. no son validas para las tuplas.

[65]:
```
# frutas = tuple ('Naranja', 'Platano', 43000, True)
#                    (0)        (1)      (2)   (3)

print(' TUPLAS (inmutables) '.center(80,'-'))

frutas = ('Naranja', 'Platano', 43000, True)
print(f'Tupla: {frutas}')

# Saber la cantidad de elementos de una tupla
print(f'La cantidad de elementos de la tupla es: {len(frutas)}\n')
```

```python
# Acceder a un elemento en particular
print('Indice 0 de la tupla:',frutas[0])

# Acceder a un rango (Sin incluir el ultimo indice)
print('Rango [0:2] de la tupla (sin incluir el ultimo indice):', frutas[0:2],␣
 ↪'\n')

# Recorrer elementos
for fruta in frutas:
    print('Iterando',fruta, end = ' - ')

# "Cambiar un valor a una tupla"
fruta_lista = list(frutas)
fruta_lista[0] = 'Pera'
frutas = tuple(fruta_lista)
print(f'''
"Cambiando" el valor de una tupla: {frutas}
''')
```

```
----------------------------- TUPLAS (inmutables) -----------------------------
Tupla: ('Naranja', 'Platano', 43000, True)
La cantidad de elementos de la tupla es: 4

Indice 0 de la tupla: Naranja
Rango [0:2] de la tupla (sin incluir el ultimo indice): ('Naranja', 'Platano')

Iterando Naranja - Iterando Platano - Iterando 43000 - Iterando True -
"Cambiando" el valor de una tupla: ('Pera', 'Platano', 43000, True)
```

3.6.3 Funciones

Las funciones en Python son bloques de código que se diseñan para realizar una tarea específica, y pueden ser reutilizadas a lo largo de tu programa.

1. Definición de una función en Python:

- Se utiliza la palabra clave `def` seguida del nombre de la función y paréntesis que pueden incluir parámetros. Los bloques de código dentro de la función deben estar indentados. Ejemplo: `def my_function(param1, param2):`
- Los parámetros son variables que actúan como los valores que pasas a la función.
- El cuerpo de la función es donde escribe la lógica principal de la función. Ejemplo:

```python
def my_function(param1, param2):
  print(f'El parametro 1 es: {param1}') # Este es el cuerpo
  print(f'El parametro 2 es: {param2}') # de la función
```

- La palabra clave `return` se utiliza para devolver un resultado desde la función al lugar donde fue llamada. Una función puede retornar cualquier tipo de dato y hasta múltiples valores. Ejemplo:

```
def add(number1, number2):
  return number1 + number2
```

2. Llamada a la Función Para usar una función, simplemente llama su nombre seguido de paréntesis que pueden contener argumentos específicos. Ejemplo:

```
resultado_suma = add(2,5)
print(resultado_suma)
```

```
my_function(2,5)
```

Ventajas de usar funciones:

- ***Modularidad:*** Permite dividir tu programa en secciones pequeñas y manejables.
- ***Reutilización de código:*** Una vez definida, una función puede ser utilizada muchas veces, en diferentes partes del programa o incluso en diferentes programas.

[77]:
```
# Función que calcula el área del triángulo

def area_triangulo(base, altura):
    return abs((base * altura) / 2)

base = 5.0
altura = 3.0
area = area_triangulo(base, altura)
print(f"El área del triángulo con base {base} y altura {altura} es {area}")
```

```
El área del triángulo con base 5.0 y altura 3.0 es 7.5
```