

# Sesión 1

May 27, 2025

Universidad Autónoma Metropolitana Unidad Iztapalapa (UAM-I)

Maestría en Matemáticas Aplicadas e Industriales (MCMIAI)

Tópicos Selectos de Matemáticas Aplicadas II - Análisis de Datos con Python

Trimestre 25-P

Profesor: Dr. Juan Alberto Martínez Cadena

Alumno Alan Badillo Salas

## 1 Introducción Python

### 1.1 Ciencia de Datos

Intersección entre:

- **Estadística** - Herramientas de análisis
- **Programación** - Técnicas de procesamiento
- **Conocimiento de Campo** - Modelos y Fenómenos

### 1.2 Tipos de Datos

Básicos:

Representan los valores fundamentales de para operar datos

- **Enteros** - Clase `int` representan valores numéricos enteros, por ejemplo, 123
- **Decimales** - Clase `float` representan valores numéricos decimales, por ejemplo, 3.1415
- **Lógicos** - Clase `bool` representan valores lógicos *booleanos* de verdadero o falso, por ejemplo, `True` o `False`
- **Textos** - Clase `str` representan valores de textos, por ejemplo, "Hola mundo", 'Adiós mundo'

Collecciones:

Representan objetos complejos capaces de almacenar múltiples valores

- **Listas** - Clase `list` representan colecciones lineales, indexadas y dinámicas, por ejemplo, [1, 2, 3] o ["manzana", "pera", "kiwi"]
- **Tuplas** - Clase `tuple` representan colecciones lineales, indexadas e inmutables, por ejemplo, (1, "manzana", 45.6) o (2, "pera", 38.7)

- **Diccionarios** - Clase dict representan colecciones no lineales, indexadas y dinámicas, por ejemplo, { "id": 1, "nombre": "manzana", "precio": 45.6 } o { "id": 2, "nombre": "pera", "precio": 38.7 }

## 1.3 Manejo de listas

### 1.3.1 Ejemplo de listas

```
[63]: lista1 = [3, 4, 5]
      lista2 = ["uno", "dos", "tres"]
      lista3 = ["uno", 2, 3.14, [10, 20, 30]]
```

### 1.3.2 Acceso por índice

```
[64]: lista1[0]
```

```
[64]: 3
```

```
[65]: lista2[1]
```

```
[65]: 'dos'
```

```
[66]: lista3[3]
```

```
[66]: [10, 20, 30]
```

```
[67]: lista1[0:2]
```

```
[67]: [3, 4]
```

```
[68]: lista1[1:3]
```

```
[68]: [4, 5]
```

```
[69]: lista1[1:]
```

```
[69]: [4, 5]
```

```
[70]: lista1[:]
```

```
[70]: [3, 4, 5]
```

### 1.3.3 Reemplazar elementos

```
[71]: lista1[0] = 7

      lista1
```

```
[71]: [7, 4, 5]
```

```
[72]: lista2[0:1] = ["one", "two"]  
  
lista2
```

```
[72]: ['one', 'two', 'dos', 'tres']
```

### 1.3.4 Concatenación

```
[73]: lista1 + lista2
```

```
[73]: [7, 4, 5, 'one', 'two', 'dos', 'tres']
```

### 1.3.5 Añadir de elementos

```
[74]: lista1.append(6)  
lista1
```

```
[74]: [7, 4, 5, 6]
```

### 1.3.6 Eliminar elementos

```
[75]: lista1.remove(4)  
  
lista1
```

```
[75]: [7, 5, 6]
```

```
[76]: lista1.pop()  
  
lista1
```

```
[76]: [7, 5]
```

```
[77]: print(lista2)  
  
lista2.pop(2)  
  
lista2
```

```
['one', 'two', 'dos', 'tres']
```

```
[77]: ['one', 'two', 'tres']
```

## 1.4 Manejo de textos

### 1.4.1 Cadenas multi-línea

```
[78]: reporte = """  
Hola mundo  
  
Esta es una cadena más compleja  
  
Bienvenido a Python  
"""  
  
reporte
```

```
[78]: '\nHola mundo\n\nEsta es una cadena más compleja\n\nBienvenido a Python \n'
```

```
[79]: print(reporte)
```

Hola mundo

Esta es una cadena más compleja

Bienvenido a Python

### 1.4.2 Limpiar los blancos al principio y final de una cadena

```
[80]: reporte.strip()
```

```
[80]: 'Hola mundo\n\nEsta es una cadena más compleja\n\nBienvenido a Python '
```

```
[81]: print(reporte.strip())
```

Hola mundo

Esta es una cadena más compleja

Bienvenido a Python

### 1.4.3 Concatenar textos

```
[82]: nombre = "Coca Cola"  
  
precio = 45.5  
  
reporte = nombre + " " + str(precio)  
  
reporte
```

```
[82]: 'Coca Cola 45.5'
```

#### 1.4.4 Formato de textos

```
[83]: f"Producto: {nombre} | Precio: {precio:.2f}"
```

```
[83]: 'Producto: Coca Cola | Precio: 45.50'
```

### 1.5 Operaciones binarias

```
[84]: 2 ** 10
```

```
[84]: 1024
```

#### 1.5.1 Ejemplo: Resolver la ecuación cuadrática

Ecuación cuadrática general:

$$ax^2 + bx + c = 0$$

Soluciones generales:

$$x_1 = \frac{-b + \sqrt{b^2 - 4ac}}{2a}$$

$$x_2 = \frac{-b - \sqrt{b^2 - 4ac}}{2a}$$

Caso particular:

$$x^2 + x - 1 = 0$$

```
[85]: from math import sqrt

a = 1
b = 1
c = -1

x1 = (-b + sqrt(b ** 2 - 4 * a * c)) / (2 * a)
x2 = (-b - sqrt(b ** 2 - 4 * a * c)) / (2 * a)

x1, x2
```

```
[85]: (0.6180339887498949, -1.618033988749895)
```

Nota: `from <paquete> import <función>`

```
[86]: a = -1
      b = 1
      c = 11

      x1 = (-b + sqrt(b ** 2 - 4 * a * c)) / (2 * a)
      x2 = (-b - sqrt(b ** 2 - 4 * a * c)) / (2 * a)

      x1, x2
```

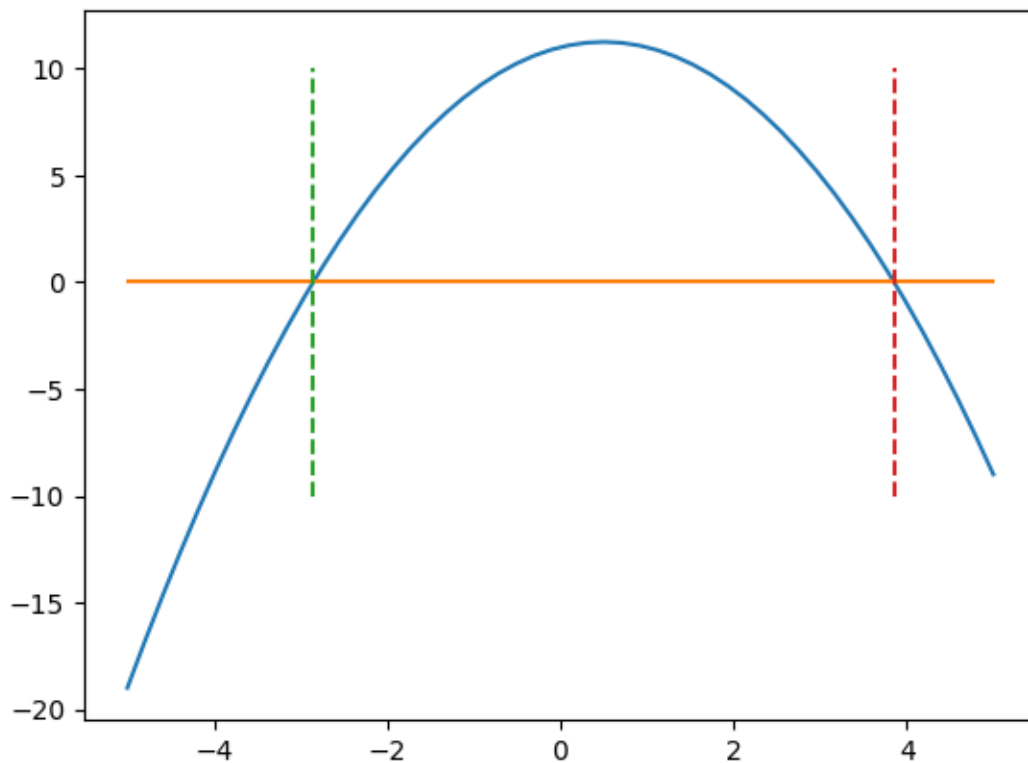
```
[86]: (-2.8541019662496847, 3.8541019662496847)
```

### 1.5.2 Graficar las soluciones

```
[87]: import matplotlib.pyplot as pyplot
      import numpy

      x = numpy.linspace(-5, 5)
      y = a * x ** 2 + b * x + c

      pyplot.plot(x, y)
      pyplot.plot(x, numpy.zeros(x.shape))
      pyplot.plot([x1, x1], [-10, 10], linestyle="--")
      pyplot.plot([x2, x2], [-10, 10], linestyle="--")
      pyplot.show()
```



## 1.6 Estructuras de control

Permiten modificar la ejecución de las operaciones para condicionar, repetir o llamar a bloques de instrucciones.

Un bloque se establece por líneas de código correctamente indentadas y de manera uniforme.

Ejemplo de un bloque de instrucciones:

```
<bloque>:
    <línea 1>
    <línea 2>
    <...>
    <línea n>
```

Podemos dentro de un bloque anidar otro bloque:

```
<bloque 1>:
    <línea 1.1>
    <línea 1.2>
    <...>

    <bloque 2>:
        <línea 2.1>
        <línea 2.2>
        <...>
        <línea 2.m>

    <...>
    <línea 1.n>
```

Cada nivel de anidación requiere aumentar los niveles de indentación.

### 1.6.1 Condicional if

Permite evaluar una o varias condiciones previo la ejecución del primer bloque que cumpla la condición u opcionalmente el bloque finalizador:

```
if <condición 1>:
    <segmento 1>
elif <condición 2>:
    <segmento 2>
elif <condición ...>:
    <segmento ...>
elif <condición n>:
    <segmento n>
else:
    <segmento falso>
```

Los componentes `elif` y `else` son opcionales y sólo puede haber un segmento falso que atrape el código cuando no se cumple ninguna condición.

```
[88]: a = 1
      b = 1
      c = 1

      d = b ** 2 - 4 * a * c

      if d == 0:
          x1 = -b / (2 * a)
          print("Hay una única raíz:", x1)
      elif d > 0:
          x1 = (-b + sqrt(d)) / (2 * a)
          x2 = (-b - sqrt(d)) / (2 * a)
          print("Existen dos raíces:", x1, x2)
      else:
          x1 = (-b + sqrt(-d) * 1j) / (2 * a)
          x2 = (-b - sqrt(-d) * 1j) / (2 * a)
          print("La raíces son complejas:", x1, x2)
```

La raíces son complejas: (-0.5+0.8660254037844386j) (-0.5-0.8660254037844386j)

## 1.7 Captura de datos

Podemos capturar datos y convertirlos al formato adecuado:

```
<valor> = <tipo>( input(<mensaje>) )
```

### 1.7.1 Calcular el IMC

```
[89]: estatura = float(input("Ingresa la estatura (metros):"))
      peso = float(input("Ingresa el peso (kilogramos):"))

      imc = peso / estatura ** 2

      print(f"El IMC (Índice de Masa Corporal) para una persona de {estatura:.2f}m y {peso:.2f}kg es: {imc:.2f}")
```

El IMC (Índice de Masa Corporal) para una persona de 1.70m y 34.00kg es: 11.76