Programación Básica-Intermedia en Python
Clase 1

Fabián Palacios Pereira, MSc. Ing.[1]

[1] *fpalacios@fiuna.edu.py* Laboratorio de Sistemas Distribuidos,
Dpto. de Ing. Electrónica y Mecatrónica (DIEM),
Centro de Innovación Tecnológica (CITEC),
Facultad de Ingeniería de la UNA (FIUNA), Isla Bogado, Luque, Paraguay.

10 de Mayo del 2025

Resumen

En esta clase se introducen los fundamentos del lenguaje Python, orientados a su aplicación en problemas de ingeniería. Se abordan los conceptos esenciales de sintaxis, definición de variables y tipos de datos básicos, junto con la entrada y salida de información por consola. Además, se presentan los operadores aritméticos y lógicos, y se estudian las estructuras de control fundamentales como condicionales (if, elif, else) y bucles (for, while), permitiendo construir algoritmos simples y estructurados. También se guía al estudiante en la instalación y uso de entornos de desarrollo como Jupyter Notebook o Visual Studio Code.

1. Generalidades de Python

1.1. ¿Qué es Python?

Python es un lenguaje de programación de alto nivel, interpretado, multipropósito, con una sintaxis clara y concisa. Fue creado por Guido van Rossum

y lanzado en 1991, y desde entonces se ha convertido en uno de los lenguajes más populares del mundo.

Es un lenguaje orientado a objetos, aunque también admite programación estructurada y funcional.

1.2. Características principales

- Interpretado: No se compila previamente; el código se ejecuta línea por línea.
- Tipado dinámico: No es necesario declarar tipos de variables.
- Sintaxis clara: Su estructura obliga a escribir código legible (usando indentación).
- Multiparadigma: Admite programación orientada a objetos, estructurada y funcional.
- Gran comunidad y ecosistema: Miles de librerías disponibles para ciencia, ingeniería, IA, etc.
- Portabilidad: Funciona en Windows, macOS, Linux, Raspberry Pi, etc.

1.3. ¿Para qué se usa Python?

Python es extremadamente versátil. Algunos campos donde se destaca:

- Ciencia e ingeniería: análisis de datos, simulación, visualización, automatización.
- Inteligencia artificial y machine learning: con librerías como TensorFlow, PyTorch, scikit-learn.
- Desarrollo web: frameworks como Django y Flask.
- Automatización y scripting: tareas repetitivas, manejo de archivos, scraping web.
- Educación: lenguaje ideal para aprender a programar.
- Internet de las Cosas (IoT): especialmente con Raspberry Pi y microcontroladores.

1.4. Comparación con otros lenguajes

- C/C++: Python es más lento, pero mucho más fácil de escribir y leer. C y C++ requieren manejo de memoria; Python no.
- MATLAB: Python es libre y más flexible. Muchas librerías científicas imitan MATLAB (numpy, scipy, matplotlib).
- Java: Python es más conciso y rápido para prototipado. Java es más rápido y estructurado para sistemas grandes.
- JavaScript: JS más orientado para la web; Python es más general. Se complementan más que competir.

1.5. Ventajas de Python

- Sintaxis simple y limpia (muy parecido al pseudocódigo).
- Comunidad enorme: muchas soluciones ya existen.
- Enorme cantidad de librerías científicas, gráficas, de IA, web, etc.
- Ideal para prototipado rápido.
- Código muy legible y mantenible.
- Código multiplataforma.

1.6. Desventajas de Python

- Velocidad: no es tan rápido como C/C++ o Rust (aunque se puede integrar con ellos).
- No tan adecuado para móviles o sistemas embebidos pequeños (aunque hay versiones como MicroPython).
- El tipado dinámico puede ser fuente de errores difíciles de detectar en proyectos grandes.
- No tiene un entorno de desarrollo estándar (aunque Jupyter y VS Code son los más usados).

1.7. ¿Por qué se recomienda Python para ingeniería?

- Curva de aprendizaje baja: permite enfocarse en los problemas, no en la sintaxis.
- Ideal para prototipos, simulaciones y automatización.
- Librerías potentes: numpy, scipy, matplotlib, pandas, sympy, etc.
- Fácil de integrar con sensores, hardware y sistemas de control.
- Muchos entornos académicos y científicos lo usan en investigación y desarrollo.
- Aplicado en varios campos en auge como ciencia de datos, inteligencia artificial, IoT, entre otros.

1.8. Ejemplo de sintaxis

Ejemplo con C++ y Python: Introducir una palabra y mostrar la palabra en la salida.

```cpp
#include <iostream>
#include <string>
using namespace std;

int main() {
    string palabra;
    cout << "Ingrese una palabra: ";
    cin >> palabra;
    cout << "La palabra ingresada es: " << palabra <<
       endl;
    return 0;
}
```

Listing 1: Código en C++

```python
palabra = input("Ingrese una palabra: ")
print("La palabra ingresada es:", palabra)
```

Listing 2: Código en Python

2. Palabras reservadas, variables y tipos de datos

2.1. Palabras reservadas

Las palabras reservadas (o keywords) son términos que tienen un significado especial en el lenguaje y no pueden ser usadas como nombres de variables o funciones. Ejemplos:

```
if, else, elif, while, for, def, return, import, from
```

No es posible usar:

```
if = 5
for = 'hola'
```

porque *if*, también *for* son palabras reservadas

2.2. Variables

Una variable es un nombre que almacena un valor. Python no necesita declarar el tipo de la variable, lo infiere automáticamente. Ejemplo:

```
1  x = 10         # Entero
2  nombre = "Ana"  # Cadena de texto
3  pi = 3.1416    # Flotante
```

Reglas para nombrar variables:

- Comienzan con una letra, nunca con un número.
- No pueden ser palabras reservadas.
- Se recomienda usar nombres descriptivos, cortos y fáciles de recordar.

Cuadro 1: Tipos de datos básicos.

Tipo	Ejemplo	Descripción
int	a = 5	Números enteros
float	b = 4.14	Números decimales (coma flotante)
str	c = "Hola"	Texto (cadenas de caracteres)
bool	d = True	Booleanos: True o False

2.3. Ejemplos

Se presentan algunos ejemplos para practicar la declaración de variables en Python:

```
nombre = "Jose"
edad = 30
altura = 1.85
tipo_de_sangre = 'b'
```

Es posible verificar el tipo de los datos mediante la funcion *type()*.

```
nombre = "Jose"
edad = 30
altura = 1.85
tipo_de_sangre = 'b'

print(type(nombre))
print(type(altura))
```

3. Entrada y salida de datos

En Python, la entrada y la salida de datos son tareas esenciales y bastante sencillas de manejar. Las funciones principales que se utilizan para estos fines son *input()* para introducir datos por el usuario y *print()* para mostrar resultados en la terminal.

3.1. Salida de datos con *print()*

La función print() se utiliza para mostrar información en la pantalla o terminal. Puede imprimir texto, variables y resultados de expresiones. print() acepta múltiples argumentos, que se separan por comas, y automáticamente inserta un espacio entre ellos. Ejemplo:

```
nombre = "Ana"
edad = 30
print("Nombre:", nombre, "Edad:", edad)
```

3.2. Entrada de datos con *input()*

La función *input()* permite que el programa reciba información del usuario desde la línea de comandos o terminal. El valor que el usuario ingresa se devuelve como una cadena de texto (string), independientemente de si se introducen números u otros tipos de datos. Ejemplo:

```
variable = input("Mensaje al usuario: ")
```

Ejemplo de entrada y salida de texto:

```
frase = input("Escriba un mensaje: ")
print("El mensaje es: ", frase)
```

4. Operadores

4.0.1. Aritméticos

Los operadores aritméticos permiten realizar operaciones matemáticas básicas. Python soporta los siguientes:

4.0.2. Lógicos

Los operadores lógicos permiten combinar expresiones condicionales. Son clave en estructuras de control (*if*, *while*) y en programación de sistemas y automatización.

Cuadro 2: Operadores aritméticos en Python.

Operador	Ejemplo	Descripción
`+`	`a + b`	Suma
`-`	`a - b`	Resta
`*`	`a * b`	Multiplicación
`/`	`a / b`	División (resultado flotante)
`//`	`a // b`	División entera (descarta decimales)
`%`	`a% b`	Módulo (resto de la división)
`**`	`a ** b`	Potencia (a elevado a la b)

Cuadro 3: Operadores lógicos en Python.

Operador	Ejemplo	Descripción
`and`	`True and False`	Verdadero si ambas condiciones son verdaderas
`or`	`True or False`	Verdadero si al menos una condición es verdadera
`not`	`not True`	Invierte el valor lógico (negación)

4.0.3. De comparación

Los operadores de comparación comparan dos valores y devuelven un resultado booleano (True o False). Son fundamentales para tomar decisiones en los programas.

5. Condicionales

Las estructuras condicionales permiten ejecutar bloques de código dependiendo de si una condición es verdadera o falsa. Son fundamentales para la toma de decisiones en un programa. Sintaxis básica:

```
1    if condicion:
2        # si la condicion es verdadera
3    elif otra_condicion:
4        # si la anterior no se cumple, pero esta si
5    else:
6        # si ninguna condicion anterior se cumple
```

Cuadro 4: Operadores de comparación en Python.

Operador	Ejemplo	Descripción
==	a == b	Igualdad: Verdadero si a es igual a b
!=	a != b	Distinto: Verdadero si a es diferente de b
>	a >b	Mayor que
<	a <b	Menor que
>=	a >= b	Mayor o igual que
<=	a <= b	Menor o igual que

6. Bucles

Los bucles permiten repetir instrucciones múltiples veces. En Python, los más comunes son for y while.

6.0.1. Bucles for

Un bucle for es una estructura de programación que permite ejecutar un bloque de código repetidamente un número determinado de veces. Es útil cuando se conoce de antemano cuántas veces se necesita ejecutar un código.

```
for i in range(5):
print("Repeticion", i)
```

Otro ejemplo:

```
frutas = ["manzana", "banana", "pera"]
for fruta in frutas:
    print(fruta)
```

6.0.2. Bucle While

Un bucle while es una estructura de control de flujo en programación que permite ejecutar repetidamente un bloque de código mientras una condición específica sea verdadera. Una vez que la condición se vuelve falsa, la ejecución del bucle se detiene y el programa continúa con la siguiente instrucción después del bucle.

```
    while condicion:
        # bloque de codigo
```

Ejemplo:

```
    contador = 0
    while contador < 3:
        print("Contador:", contador)
        contador += 1
```

A tener en cuenta:

- Asegurarse de que la condición del while cambie, o el bucle será infinito.
- En for, range(inicio, fin) no incluye el último valor (fin - 1).