Capítulo 1: Introducción a la Programación y Python

1.1 Bienvenida al Mundo del Backend

¡Bienvenido/a a este curso de Desarrollo Backend con Python! Si estás aquí, es probable que tengas curiosidad sobre cómo funcionan las aplicaciones web y móviles "detrás de escena", o quizás buscas iniciar una carrera en el emocionante campo del desarrollo de software. Sea cual sea tu motivación, ¡estás en el lugar correcto!

En este curso, partiremos desde los fundamentos absolutos. No necesitas ningún conocimiento previo de programación. Nuestro objetivo es que, al finalizar, seas capaz de diseñar, construir y desplegar servicios backend utilizando Python, uno de los lenguajes más populares y demandados en la industria, junto con herramientas y frameworks esenciales como Django, FastAPI, bases de datos SQL (PostgreSQL) y NoSQL (MongoDB).

Este primer capítulo sienta las bases: entenderemos qué es programar, qué hace un desarrollador backend, por qué Python es una excelente elección y, lo más importante, prepararemos nuestras herramientas e escribiremos nuestras primeras líneas de código. ¡Empecemos!

1.2 ¿Qué Significa Programar?

En esencia, **programar es dar instrucciones a una computadora**. Imagina que quieres enseñarle a un robot cómo preparar café. No puedes simplemente decirle "haz café". Debes darle una secuencia de pasos extremadamente detallados y sin ambigüedades:

1. Toma la cafetera.
2. Ábrela.
3. Toma el filtro de papel.
4. Coloca el filtro en el compartimento designado.
5. Toma el paquete de café molido...

Y así sucesivamente. Si una instrucción es vaga o incorrecta, el robot fallará. Las computadoras son como esos robots: increíblemente rápidas y potentes para seguir instrucciones, pero necesitan que esas instrucciones sean absolutamente precisas y estén escritas en un lenguaje que comprendan.

Existen muchos **lenguajes de programación** (Java, C++, JavaScript, Python, etc.), cada uno con sus fortalezas y debilidades. Nosotros usaremos **Python**, elegido por su sintaxis clara y legible (a menudo se dice que es casi como leer inglés) y su gran versatilidad.

1.3 El Rol del Desarrollador Backend

Piensa en tu aplicación favorita, ya sea una red social, una tienda online o una plataforma de streaming. Lo que ves e interactúas directamente (botones, imágenes, menús) se llama **frontend**. Es la "fachada" o el "salón del restaurante".

El **backend** es todo lo que sucede detrás de esa fachada, la "cocina" y la "trastienda". Las responsabilidades clave de un desarrollador backend incluyen:

- **Lógica del Servidor:** Implementar las reglas del negocio (ej: verificar si un usuario tiene permiso para ver cierto contenido, calcular el total de una compra, procesar un pago).
- **Gestión de Bases de Datos:** Diseñar, crear y administrar cómo se almacena, recupera y actualiza la información (ej: datos de usuarios, productos, publicaciones).
- **APIs (Interfaces de Programación de Aplicaciones):** Crear las "puertas" de comunicación para que el frontend (u otras aplicaciones) puedan solicitar información o realizar acciones en el backend.
- **Autenticación y Autorización:** Asegurarse de que los usuarios sean quienes dicen ser y que solo puedan acceder a los datos y funciones permitidos.
- **Rendimiento y Escalabilidad:** Optimizar el código y la infraestructura para que la aplicación sea rápida y pueda manejar un número creciente de usuarios.

En resumen, el backend es el motor que impulsa la aplicación.

1.4 ¿Por Qué Elegir Python?

Python se ha convertido en una opción extremadamente popular para el desarrollo backend, y por buenas razones:

- **Fácil de Aprender y Leer:** Su sintaxis es limpia y se asemeja al lenguaje humano, lo que facilita la curva de aprendizaje para principiantes y mejora la colaboración en equipos.
- **Gran Comunidad y Ecosistema:** Existe una vasta comunidad global de desarrolladores de Python. Esto significa abundante documentación, tutoriales, foros de ayuda y, crucialmente, una enorme cantidad de **librerías y frameworks** pre-escritos que resuelven problemas comunes y aceleran el desarrollo (como Django, Flask y FastAPI, que exploraremos).
- **Versatilidad:** Python no solo brilla en backend. Se usa extensivamente en ciencia de datos, inteligencia artificial, machine learning, automatización de tareas (scripting), desarrollo de juegos y más. Aprender Python abre muchas puertas.
- **Alta Demanda Laboral:** Dada su popularidad y aplicabilidad, los desarrolladores de Python están muy solicitados en el mercado laboral.

1.5 Preparando Nuestro Entorno de Trabajo

Antes de escribir código, necesitamos instalar las herramientas esenciales: el intérprete de Python y un editor de código.

1.5.1 Instalando Python

El **intérprete de Python** es el programa que lee nuestro código `.py` y lo ejecuta.

1. **Verificar si ya está instalado:** Abre la aplicación de terminal de tu sistema operativo:
    - Windows: Busca "CMD" o "PowerShell" en el menú de inicio.
    - macOS: Ve a `Aplicaciones > Utilidades > Terminal`.
    - Linux: Busca "Terminal" o usa un atajo como `Ctrl+Alt+T`. Escribe `python --version` y presiona Enter. Si no funciona, prueba `python3 --version`. Si obtienes una respuesta como `Python 3.x.x` (donde x son números de versión), ¡ya lo tienes! Si dice `Python 2.x.x` o da un error, necesitas instalar/actualizar.
2. **Descargar Python:** Ve al sitio web oficial: python.org. Ve a la sección "Downloads". Normalmente detectará tu sistema operativo y te ofrecerá el instalador recomendado (asegúrate de que sea para Python 3, idealmente la versión estable más reciente).
3. **Instalar Python:** Ejecuta el instalador descargado.
    - **¡Atención usuarios de Windows!** En la primera pantalla del instalador, asegúrate de marcar la casilla que dice **"Add Python X.X to PATH"** o similar. Esto permite ejecutar Python desde cualquier lugar en la terminal fácilmente.
    - Para la mayoría de los usuarios, la instalación por defecto ("Install Now") es adecuada.
4. **Verificar la Instalación:** Una vez finalizada la instalación, **cierra** cualquier ventana de terminal que tuvieras abierta y **abre una nueva**. Escribe nuevamente `python --version` (o `python3 --version`). Ahora deberías ver la versión 3.x.x que instalaste. Si tienes problemas, consulta la documentación de Python o busca guías específicas para tu sistema operativo.

1.5.2 Instalando Visual Studio Code (VS Code)

Necesitamos un lugar donde escribir nuestro código. Un **editor de código** es como un procesador de texto súper vitaminado para programadores. Usaremos **Visual Studio Code (VS Code)**, que es gratuito, muy popular y altamente personalizable.

1. **Descargar VS Code:** Ve a code.visualstudio.com y descarga el instalador para tu sistema operativo (Windows, macOS, Linux).
2. **Instalar VS Code:** Ejecuta el instalador y sigue los pasos (generalmente aceptar términos y pulsar "Siguiente").
3. **Explorar VS Code (Opcional):** Al abrir VS Code, familiarízate brevemente con la interfaz: la barra lateral izquierda con iconos (Explorador, Buscar, Extensiones), el área principal de edición y la posibilidad de abrir una terminal integrada (`Terminal > New Terminal`).
4. **Instalar la Extensión de Python:** Para que VS Code nos ayude específicamente con Python (resaltado de código, sugerencias, etc.), necesitamos una extensión:
    - Haz clic en el icono de Extensiones en la barra lateral izquierda (parece unos bloques encajando).
    - En la barra de búsqueda, escribe `Python`.
    - Busca la extensión llamada **"Python"** publicada por **Microsoft** e instálala haciendo clic en el botón "Install".

1.5.3 Creando el Espacio de Trabajo

Es una buena práctica tener una carpeta dedicada para cada proyecto o curso.

1. Crea una carpeta en tu disco duro donde guardarás todos los archivos de este curso. Por ejemplo, llámala `curso_backend_python`.
2. En VS Code, ve al menú `File > Open Folder...` (o `Archivo > Abrir Carpeta...`) y selecciona la carpeta que acabas de crear. VS Code ahora se centrará en esa carpeta, mostrándola en el panel del Explorador a la izquierda.

¡Listo! Nuestro taller está preparado.

1.6 Nuestro Primer Programa: ¡Hola Mundo!

Es una tradición iniciar en cualquier lenguaje de programación con un programa que simplemente muestra el mensaje "Hola Mundo" en la pantalla.

1. Asegúrate de tener abierta la carpeta del curso en VS Code.
2. En el panel del Explorador (izquierda), haz clic derecho en el espacio vacío debajo del nombre de la carpeta y selecciona "New File" (Nuevo Archivo).
3. Nombra el archivo `clase1_hola.py`. La extensión `.py` es crucial.
4. En el editor que se abre, escribe la siguiente línea de código:

`Código Python:`

```python
print("¡Hola Mundo desde Python!")
```

`print()`: Esta es una **función** incorporada de Python. Las funciones son bloques de código reutilizables que realizan una tarea específica. La función `print()` toma lo que le pases entre paréntesis y lo muestra en la consola o terminal. Como "¡Hola Mundo desde Python!" es texto, lo ponemos entre comillas.

5. **Ejecutar el Código:**
    - Abre la terminal integrada en VS Code (`Terminal > New Terminal`).
    - Verás un prompt esperando comandos. Escribe: `python clase1_hola.py` y presiona Enter. (Si usas `python3` para la verificación, usa `python3` aquí también).
    - ¡Deberías ver el mensaje `¡Hola Mundo desde Python!` impreso justo debajo de tu comando!

¡Felicidades! Has escrito y ejecutado tu primer programa en Python.

1.7 Comentarios: Dejando Notas en el Código

A medida que tus programas crezcan, querrás dejar notas para ti mismo o para otros desarrolladores que lean tu código. Para esto usamos **comentarios**. Python ignora completamente cualquier cosa en una línea que siga al símbolo `#`.

Código Python:

```
# Este es un comentario de una sola línea. Explica qué hace el código siguiente.
print("Este mensaje sí se mostrará.")

# print("Este mensaje no se mostrará porque está comentado.")

variable = 10 # También puedes poner comentarios al final de una línea de código.
```

Usa comentarios para explicar partes complejas de tu código o para dejar recordatorios.

1.8 Variables: Almacenando Información

Los programas necesitan trabajar con datos. Las **variables** son como etiquetas o nombres que le ponemos a "cajas" en la memoria de la computadora donde guardamos esos datos para poder usarlos y modificarlos más tarde.

Para crear una variable y asignarle un valor, usamos el signo igual (`=`):

Código Python:

```
nombre_del_curso = "Desarrollo Backend con Python"
numero_de_clases = 40
precio = 299.99
curso_empezado = True
```

- `nombre_del_curso`, `numero_de_clases`, `precio`, `curso_empezado` son los nombres de las variables.
- `=` es el operador de asignación.
- `"Desarrollo Backend con Python"`, `40`, `299.99`, `True` son los valores asignados a cada variable.

Reglas y Convenciones para Nombrar Variables:

- Pueden contener letras (a-z, A-Z), números (0-9) y guion bajo (`_`).
- No pueden empezar con un número.

- Distinguen entre mayúsculas y minúsculas (`nombre` es diferente de `Nombre`).
- **Convención (PEP 8):** En Python, se recomienda usar `snake_case` para nombres de variables y funciones (palabras en minúscula separadas por guion bajo), como `nombre_del_curso`.
- Elige nombres descriptivos que indiquen qué guarda la variable.

Una vez creada una variable, puedes usar su nombre para acceder a su valor:

Código Python:

```
print(nombre_del_curso)
print("Este curso tiene", numero_de_clases, "clases.")
```

1.9 Tipos de Datos Fundamentales

Python maneja diferentes tipos de datos. Los más básicos e importantes por ahora son:

- **`int` (Integer - Entero):** Representa números enteros, positivos o negativos, sin parte decimal.

Código Python:

```
edad = 35
temperatura = -5
año = 2025
```

- **`float` (Floating Point - Punto Flotante):** Representa números reales, es decir, números con parte decimal. Se usa el punto (`.`) como separador decimal.

Código Python:

```
altura = 1.78
pi = 3.14159
precio_gasolina = 4.50
```

- **`str` (String - Cadena de Texto):** Representa secuencias de caracteres (letras, números, símbolos). Siempre debe ir entre comillas, ya sean simples (`'`) o dobles (`"`).

Código Python:

```
saludo = "Hola"
ciudad = 'Medellín'
mensaje_error = "Error 404: No encontrado"
```

- `bool` **(Boolean - Booleano):** Representa un valor de verdad. Solo puede tener dos valores posibles: `True` (Verdadero) o `False` (Falso). **Importante:** La primera letra siempre va en mayúscula.

Código Python:

```
es_mayor_de_edad = True
tiene_descuento = False
luces_encendidas = True
```

Puedes averiguar el tipo de dato de una variable usando la función `type()`:

Código Python:

```
print(type(edad))             # Salida: <class 'int'>
print(type(altura))           # Salida: <class 'float'>
print(type(ciudad))           # Salida: <class 'str'>
print(type(es_mayor_de_edad)) # Salida: <class 'bool'>
```

Entender los tipos de datos es crucial porque determina qué operaciones puedes realizar con ellos.

1.10 Operaciones Básicas

Python nos permite realizar diversas operaciones con nuestros datos.

1.10.1 Operadores Aritméticos

Son los símbolos que usamos para realizar cálculos matemáticos:

Operador	Nombre	Ejemplo	Resultado
`+`	Suma	`10 + 5`	`15`

`-`	Resta	`10 - 5`	`5`
`*`	Multiplicación	`10 * 5`	`50`
`/`	División	`10 / 4`	`2.5`
`//`	División Entera	`10 // 4`	`2`
`%`	Módulo (Residuo)	`10 % 4`	`2`
`**`	Exponenciación	`10 ** 2`	`100`

Código Python:

```
base = 5
altura = 10
area = base * altura / 2
print("El área del triángulo es:", area) # Salida: El área del
triángulo es: 25.0

# Ejemplo de módulo: ¿es un número par o impar?
numero = 15
residuo = numero % 2
print(f"El residuo de dividir {numero} entre 2 es: {residuo}") #
Salida: 1 (impar)
```

Python respeta el orden matemático de las operaciones (PEMDAS/BODMAS: Paréntesis, Exponentes, Multiplicación/División, Adición/Sustracción).

1.10.2 Operadores de Asignación

Además del `=` básico, existen operadores que combinan una operación aritmética con la asignación, sirviendo como atajos:

Operador	Ejemplo	Equivalente a
`+=`	`x += 1`	`x = x + 1`

`-=`	`y -= 5`	`y = y - 5`
`*=`	`z *= 2`	`z = z * 2`
`/=`	`w /= 3`	`w = w / 3`
`//=`	`a //= 2`	`a = a // 2`
`%=`	`b %= 3`	`b = b % 3`
`**=`	`c **= 2`	`c = c ** 2`

Código Python:

```
contador = 0
contador += 5  # contador ahora es 5
contador *= 2  # contador ahora es 10
print("Valor final del contador:", contador)
```

1.11 Interactuando con el Usuario

Los programas suelen necesitar recibir datos del usuario y mostrarle resultados.

1.11.1 Salida de Datos: La Función `print()`

Ya la hemos usado bastante. `print()` muestra los valores que le pasemos entre paréntesis en la consola. Puede recibir múltiples argumentos separados por comas, y por defecto los separa con un espacio:

Código Python:

```
nombre = "Carlos"
edad = 42
print("Usuario:", nombre, "- Edad:", edad)
# Salida: Usuario: Carlos - Edad: 42
```

1.11.2 Entrada de Datos: La Función `input()`

Para pedirle datos al usuario, usamos `input()`. Opcionalmente, podemos pasarle un mensaje (string) entre paréntesis que se mostrará al usuario para indicarle qué debe ingresar.

Código Python:

```
ciudad_natal = input("¿En qué ciudad naciste? ")
print("¡Qué bien!", ciudad_natal, "es una ciudad bonita.")
```

Al ejecutar esto, el programa se detendrá después de mostrar la pregunta, esperará a que escribas algo y presiones Enter, y lo que escribas se guardará en la variable `ciudad_natal`.

¡Advertencia Importante! La función `input()` **siempre** devuelve el dato ingresado por el usuario como una **cadena de texto (`str`)**, sin importar si el usuario escribió números.

Observa este ejemplo y el error que produce:

Código Python:

```
edad_texto = input("Introduce tu edad: ")
print("El tipo de dato de edad_texto es:", type(edad_texto)) #
Muestra <class 'str'>

# Intentamos sumar 1 a la edad (¡esto fallará!)
# edad_siguiente = edad_texto + 1
# print("El año que viene tendrás:", edad_siguiente)
# >>> TypeError: can only concatenate str (not "int") to str
```

El error `TypeError` nos dice que no podemos sumar (`+`) directamente una cadena de texto (`str`) con un número entero (`int`). Necesitamos convertir la entrada del usuario.

1.12 La Necesidad de Convertir Tipos (Casting)

Para poder realizar operaciones matemáticas con datos que hemos recibido como texto (por ejemplo, desde `input()`), debemos **convertir su tipo**. Este proceso se llama *casting*. Las funciones principales para esto son:

- **`int(valor)`**: Intenta convertir `valor` a un número entero. Si `valor` no parece un número entero válido (ej: "hola", "3.14"), dará un error (`ValueError`).
- **`float(valor)`**: Intenta convertir `valor` a un número de punto flotante (decimal). Puede convertir enteros ("10") y decimales ("3.14"). Si no parece un número válido, dará `ValueError`.
- **`str(valor)`**: Convierte `valor` (sea número, booleano, etc.) a una cadena de texto.

Corrijamos el ejemplo anterior usando `int()`:

Código Python:

```
edad_texto = input("Introduce tu edad: ")
print("La edad ingresada como texto es:", edad_texto)

# Convertimos el texto a entero ANTES de operar
edad_numero = int(edad_texto)
print("El tipo de dato de edad_numero es:", type(edad_numero)) #
Muestra <class 'int'>

edad_siguiente = edad_numero + 1
print("El año que viene tendrás:", edad_siguiente)
```

¡Ahora funciona correctamente! Siempre recuerda convertir la entrada del usuario al tipo numérico adecuado (`int` o `float`) si necesitas hacer cálculos con ella.

1.13 Formateo de Cadenas con f-strings

Hemos visto que podemos usar `+` para concatenar strings o pasar múltiples argumentos a `print()`. Sin embargo, una forma mucho más moderna, legible y eficiente de incluir variables dentro de cadenas de texto es usando **f-strings** (formatted string literals).

La sintaxis es simple: pones una letra `f` justo antes de las comillas de apertura de la cadena, y luego, dentro de la cadena, pones los nombres de las variables (o incluso expresiones) entre llaves `{}`.

Código Python:

```
nombre = "Laura"
edad = 28
profesion = "desarrolladora"

# Forma tradicional (concatenación, requiere convertir edad a str)
# print("Hola, soy " + nombre + ", tengo " + str(edad) + " años y soy
" + profesion + ".")

# Forma con f-string (más limpia y directa)
mensaje = f"Hola, soy {nombre}, tengo {edad} años y soy {profesion}."
print(mensaje)
# Salida: Hola, soy Laura, tengo 28 años y soy desarrolladora.
```

```
# ¡Incluso puedes poner expresiones dentro de las llaves!
print(f"En 5 años, {nombre} tendrá {edad + 5} años.")
# Salida: En 5 años, Laura tendrá 33 años.
```

Las f-strings son la forma recomendada de formatear cadenas en Python moderno.

1.14 ¡A Practicar!

La teoría es importante, pero la programación se aprende haciendo. Vamos a resolver un par de ejercicios para afianzar lo visto. Crea un archivo llamado `clase1_practica.py`.

Ejercicio 1: Saludo Personalizado

- Pide al usuario su nombre.
- Pide al usuario su año de nacimiento.
- Calcula la edad aproximada del usuario (Año actual - Año de nacimiento).
- Muestra un mensaje de saludo que incluya el nombre y la edad calculada.

Código Python:

```
# Solución Ejercicio 1
print("--- Ejercicio 1: Saludo Personalizado ---")

# Pedimos los datos al usuario
nombre_usuario = input("¿Cuál es tu nombre? ")
año_nacimiento_str = input("¿En qué año naciste? ")

# Convertimos el año de nacimiento a entero
año_nacimiento = int(año_nacimiento_str)

# Obtenemos el año actual (requiere importar un módulo, ¡un
adelanto!)
import datetime
año_actual = datetime.datetime.now().year
# O simplemente podemos definirlo manualmente para este ejercicio:
# año_actual = 2025 # ¡Asegúrate de usar el año correcto!

# Calculamos la edad
edad_aproximada = año_actual - año_nacimiento

# Mostramos el saludo usando f-string
```

```
print(f"¡Hola {nombre_usuario}! Si naciste en {año_nacimiento},
tienes o cumplirás {edad_aproximada} años en {año_actual}.")
print("-" * 20)
```

Nota: Para obtener el año actual automáticamente usamos el módulo `datetime`. Lo veremos más adelante, pero es útil saber que existe. Si no, puedes poner el año actual directamente.

Ejercicio 2: Calculadora Simple de Área de Rectángulo

- Pide al usuario la base de un rectángulo.
- Pide al usuario la altura de un rectángulo.
- Calcula el área (base * altura).
- Calcula el perímetro (2 * base + 2 * altura).
- Muestra el área y el perímetro calculados.

Código Python:

```
# Solución Ejercicio 2
print("--- Ejercicio 2: Calculadora de Rectángulo ---")

# Pedimos los datos (usaremos float para permitir decimales)
base_str = input("Introduce la longitud de la base del rectángulo: ")
altura_str = input("Introduce la altura del rectángulo: ")

# Convertimos a float
base = float(base_str)
altura = float(altura_str)

# Calculamos área y perímetro
area = base * altura
perimetro = 2 * base + 2 * altura
# Alternativa para perímetro: perimetro = 2 * (base + altura)

# Mostramos los resultados
print(f"Un rectángulo con base {base} y altura {altura}:")
print(f"- Tiene un área de: {area}")
print(f"- Tiene un perímetro de: {perimetro}")
print("-" * 20)
```

¡Intenta ejecutar estos ejercicios y modifícalos para experimentar!

1.15 Resumen del Capítulo y Próximos Pasos

¡Felicidades por completar este primer capítulo! Hemos cubierto los fundamentos esenciales:

- Entendimos qué es programar y el rol del backend.
- Instalamos Python y VS Code.
- Escribimos y ejecutamos nuestro primer script.
- Aprendimos sobre variables y los tipos de datos básicos (`int`, `float`, `str`, `bool`).
- Utilizamos operadores aritméticos y de asignación.
- Interactuamos con el usuario mediante `input()` y `print()`.
- Comprendimos la importancia de la conversión de tipos (`int()`, `float()`).
- Descubrimos la utilidad de las f-strings para formatear texto.

En el **Capítulo 2**, daremos un paso crucial: aprenderemos a controlar el flujo de nuestros programas. Introduciremos las **estructuras de control condicionales (`if`, `elif`, `else`)**, que permiten a nuestros programas tomar decisiones y ejecutar diferentes bloques de código según se cumplan o no ciertas condiciones.

¡Sigue practicando! Intenta crear pequeños scripts por tu cuenta. Por ejemplo, un conversor de grados Celsius a Fahrenheit (Fórmula: `F = (C * 9/5) + 32`). ¡Nos vemos en el próximo capítulo!