Bootcamp de Desarrollo Web

Clase 16

Introducción a Python

1

¿Qué es el Back End?

- El **back end** es la parte de una aplicación web que no es visible para el usuario final.
- Se compone de servidores, bases de datos y la lógica de la aplicación que hace posible que la aplicación funcione correctamente.
- Es el lugar donde se maneja la lógica de negocio, el procesamiento de datos y la interacción con el servidor.

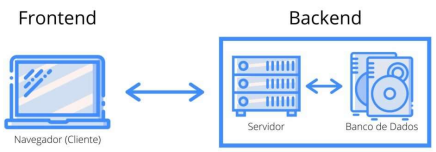
2

Componentes del Back End

- **Servidor:** Es el ordenador o sistema que aloja y ejecuta la aplicación web. Maneja las solicitudes de los usuarios y envía las respuestas adecuadas.
- **Base de Datos:** Almacena y gestiona los datos necesarios para la aplicación. Puede ser SQL (relacional) o NoSQL (no relacional), dependiendo de las necesidades de la aplicación.
- **Lógica de la Aplicación:** Este es el código que gobierna el comportamiento de la aplicación. Maneja la autenticación, la autorización, la validación de datos y otras operaciones esenciales.

Tecnologías Comunes en el Back End

- **Lenguajes de Programación:** Ejemplos incluyen Python, Java, Ruby, PHP y Node.js.
- **Frameworks:** Herramientas como Django, Spring, Ruby on Rails y Express.js que facilitan el desarrollo rápido y eficiente del back end.
- **Bases de Datos:** MySQL, PostgreSQL, MongoDB, entre otras.
- **Servidores Web:** Apache, Nginx, Microsoft IIS.



Importancia del Back End

- El back end es crucial para la **funcionalidad y la seguridad** de una aplicación web.
- Permite el manejo eficiente de datos, garantizando la **escalabilidad y el rendimiento.**
- Es responsable de **proteger la información** confidencial y prevenir vulnerabilidades de seguridad.

5

¿Qué es python?

Python es un conjunto de reglas y sintaxis que permiten a los desarrolladores escribir instrucciones para que las computadoras las ejecuten.

Lo que hace que Python sea especial es su sintaxis clara y concisa, que se asemeja mucho al lenguaje humano, lo que facilita la lectura y comprensión del código incluso para aquellos que son nuevos en la programación.

6

Campo de aplicación

Python se utiliza en una amplia variedad de campos y aplicaciones, desde **desarrollo web** y **análisis de datos** hasta **inteligencia artificial** y **automatización**.

Es conocido por su amplio conjunto de bibliotecas y frameworks que facilitan el desarrollo de una amplia gama de aplicaciones.

Características Clave de Python

- **Es multiparadigma**, ya que soporta la programación imperativa, programación orientada a objetos y funcional.
- **Es multiplataforma**: Se puede encontrar un intérprete de Python para los principales sistemas operativos: Windows, Linux y Mac OS. Además, se puede reutilizar el mismo código en cada una de las plataformas.
- **Es dinámicamente tipado:** Es decir, el tipo de las variables se decide en tiempo de ejecución.

Características Clave de Python

- **Es fuertemente tipado:** No se puede usar una variable en un contexto fuera de su tipo. Si se quisiera, habría que hacer una conversión de tipos.
- **Es interpretado:** El código no se compila a lenguaje máquina.

Primer programa en Python

Normalmente, los programas en Python se escriben en archivos con la extensión **.py**. Estos archivos se pasan al intérprete de Python para que los interprete y ejecute.

Ejemplo:

```
1.  suma = 2 + 3
2.  print(suma)
```

A continuación abre un terminal, sitúate en el directorio en el que creaste el **archivo suma.py** y escribe lo siguiente para ejecutar el código:

```
1.  python3 suma.py
```


Operadores, expresiones y sentencias en Python

Operador

Un operador es un carácter o conjunto de caracteres que actúa sobre una, dos o más variables y/o literales para llevar a cabo una operación con un resultado determinado.

Ejemplos de operadores comunes son los operadores aritméticos **+ (suma), - (resta) o * (producto)**, aunque en Python existen otros operadores.


Expresión

Una expresión es una unidad de código que devuelve un valor y está formada por una combinación de operandos (variables y literales) y operadores.

Los siguientes son ejemplos de expresiones (cada línea es una expresión diferente):

```
1.  5 + 2  # Suma del número 5 y el número 2
2.  a < 10  # Compara si el valor de la variable a es menor que 10
3.  b is None  # Compara si la identidad de la variable b es None
4.  3 * (200 - c) # Resta a 200 el valor de c y lo multiplica por 3
```


Sentencia

Por su parte, una sentencia o declaración es una instrucción que define una acción. Una sentencia puede estar formada por una o varias expresiones, aunque no siempre es así.

En definitiva, las sentencias son las instrucciones que componen nuestro programa y determinan su comportamiento.

Ejemplos de sentencias son la **asignación =** o las instrucciones **if, if ... else ..., for o while** entre otras.


Sentencias de más de una línea

Normalmente, las sentencias ocupan una sola línea. Por ejemplo:

```
1.  a = 2 + 3  # Asigna a la variable <a> el resultado de 2 + 3
```

Sin embargo, aquellas sentencias que son muy largas pueden ocupar más de una línea (la guía de estilo PEP 8, recomienda una longitud de línea máxima de 72 caracteres).

Para dividir una sentencia en varias líneas se utiliza el carácter \. Por ejemplo:

```
1.  a = 2 + 3 + 5 + \
2.      7 + 9 + 4 + \
3.      6
```


Sentencias de más de una línea

Además de la separación explícita (la que se realiza con el carácter \), en Python la continuación de línea es implícita siempre y cuando la expresión vaya dentro de los caracteres **(), [] y {}.**

Por ejemplo, podemos inicializar una lista del siguiente modo:

```
1.  a = [1, 2, 7,
2.       3, 8, 4,
3.       9]
```



Bloques de código (Indentación)

Un bloque de código es un grupo de sentencias relacionadas bien delimitadas. A diferencia de otros lenguajes como JAVA o C, en los que se usan los caracteres {} para definir un bloque de código, en Python se usa la indentación o sangrado.

El sangrado o indentación consiste en mover un bloque de texto hacia la derecha insertando espacios o tabuladores al principio de la línea, dejando un margen a la izquierda.


Bloques de código (Indentación)

Un bloque comienza con un nuevo sangrado y acaba con la primera línea cuyo sangrado sea menor.

De nuevo, la guía de estilo de Python recomienda usar los espacios en lugar de las tabulaciones para realizar el sangrado. Yo suelo utilizar 4 espacios.

```
1.  def suma_numeros(numeros):  # Bloque 1
2.      suma = 0                # Bloque 2
3.      for n in numeros:       # Bloque 2
4.          suma += n           # Bloque 3
5.          print(suma)         # Bloque 3
6.      return suma             # Bloque 2
```



Comentarios en Python

Como cualquier otro lenguaje de programación, Python permite escribir comentarios en el código. Los comentarios son útiles para explicar por qué estamos programando algo de un modo concreto o añadir indicaciones. Te aseguro que son de utilidad cuando se retoma un programa o aplicación en el futuro

Los comentarios son ignorados por el intérprete de Python. Solo tienen sentido para los programadores.

Para añadir un comentario a tu código simplemente comienza una línea con el carácter **#**:


Comentarios en Python

Ejemplo:

```
1.  # Esta línea es un comentario
2.
3.  a = 5
4.
5.  # Resultado de multiplicar a por 2
6.  print(a * 2)
```



Comentarios de varias líneas

Para escribir comentarios que ocupan varias líneas, simplemente escribe cada una de las líneas anteponiendo el carácter **#**:

```
1.  # Este comentario ocupa
2.  # 2 líneas
```

También puedes escribir un comentario en varias líneas si lo encierras entre **tres comillas simples ''' o dobles """**

```
1.  a = 2
2.
3.  '''Este comentario
4.  también ocupa 2 líneas'''
5.
6.  print(a)
```


Docstrings

Los docstrings son un tipo de comentarios especiales que se usan para documentar un módulo, función, clase o método. En realidad son la primera sentencia de cada uno de ellos y se encierran entre tres comillas simples o dobles.

Los docstrings son utilizados para generar la documentación de un programa. Además, suelen utilizarlos los entornos de desarrollo para mostrar la documentación al programador de forma fácil e intuitiva.

```
1.  def suma(a, b):
2.      """Esta función devuelve la suma de los parámetros a y b"
3.      return a + b
```



Convenciones de nombres en Python

A la hora de nombrar una variable, una función, un módulo, una clase, etc. en Python, siempre se siguen las siguientes reglas y recomendaciones:

- Un identificador puede ser cualquier combinación de letras (mayúsculas y minúsculas), números y el carácter guión bajo (_).
- Un identificador no puede comenzar por un número.
- A excepción de los nombres de clases, es una convención que todos los identificadores se escriban en minúsculas, separando las palabras con el guión bajo. **Ejemplos**: contador, suma_enteros.

Convenciones de nombres en Python

- Es una convención que los nombres de clases sigan la notación **Camel Case**, es decir, todas las letras en minúscula a excepción del primer carácter de cada palabra, que se escribe en mayúscula. Ejemplos: Coche, VehiculoMotorizado.
- No se pueden usar como identificadores las palabras reservadas.
- Como recomendación, usa identificadores que sean expresivos. Por ejemplo, contador es mejor que simplemente c.
- Python diferencia entre mayúsculas y minúsculas, de manera que variable_1 y Variable_1 son dos identificadores totalmente diferentes.

Palabras reservadas de Python

Python tiene una serie de palabras clave reservadas, por tanto, no pueden usarse como nombres de variables, funciones, etc.

Estas palabras clave se utilizan para definir la sintaxis y estructura del lenguaje Python.

La lista de palabras reservadas es la siguiente:

`and`, `as`, `assert`, `break`, `class`, `continue`, `def`, `del`, `elif`, `else`, `except`, `False`, `finally`, `for`, `from`, `global`, `if`, `import`, `in`, `is`, `lambda`, `None`, `nonlocal`, `not`, `or`, `pass`, `raise`, `return`, `True`, `try`, `yield`, `while` y `with`


Constantes en Python

No obstante, sí que es cierto que el propio Python define una serie de valores constantes en su propio namespace. Los más importantes son:

- **False:** El valor false del tipo bool.
- **True:** El valor true del tipo bool.
- **None:** El valor del tipo NoneType. Generalmente None se utiliza para representar la ausencia de valor de una variable.


Constantes en Python

Terminamos esta introducción a Python señalando que, a diferencia de otros lenguajes, en Python no existen las constantes.

Entendemos como constante una variable que una vez asignado un valor, este no se puede modificar. Es decir, que a la variable no se le puede asignar ningún otro valor una vez asignado el primero.

Se puede simular este comportamiento, siempre desde el punto de vista del programador y atendiendo a convenciones propias, pero no podemos cambiar la naturaleza mutable de las variables.

Bootcamp de Desarrollo Web

Clase 16

Introducción a Python