Django y React 2024

Diego Saavedra

May 26, 2024

Table of contents

1 Bienvenido **11**
1.1 ¿De qué trata este curso? . . . 11
1.2 ¿Para quién es este curso? . . . 11
1.3 ¿Cómo contribuir? . . . 11

I Unidad 1: Introducción a Python **13**

2 Git y GitHub **14**
2.1 ¿Qué es Git y GitHub? . . . 14
2.2 ¿Quiénes utilizan Git? . . . 15
2.3 ¿Cómo se utiliza Git? . . . 15
2.4 ¿Para qué sirve Git? . . . 16
2.5 ¿Por qué utilizar Git? . . . 17
2.6 ¿Dónde puedo utilizar Git? . . . 18
2.7 Pasos Básicos . . . 18
2.8 Instalación de Visual Studio Code . . . 19
2.8.1 Descarga e Instalación de Git . . . 20
2.8.2 Configuración . . . 21
2.8.3 Creación de un Repositorio “helloWorld” en Python . . . 21
2.8.4 Comandos Básicos de Git . . . 22
2.8.5 Estados en Git . . . 22

3 Tutorial: Moviendo Cambios entre Estados en Git **23**
3.1 Introducción . . . 23
3.2 Sección 1: Modificar Archivos en el Repositorio . . . 23
3.3 Mover Cambios de Local a Staging: . . . 23
3.4 Agregar Cambios de Local a Staging: . . . 24
3.5 Sección 2: Confirmar Cambios en un Commit . . . 24
3.6 Mover Cambios de Staging a Commit: . . . 24
3.7 Sección 3: Creación y Fusión de Ramas . . . 24
3.8 Crear una Nueva Rama: . . . 24
3.9 Implementar Funcionalidades en la Rama: . . . 24
3.10 Fusionar Ramas con la Rama Principal: . . . 25
3.11 Sección 4: Revertir Cambios en un Archivo . . . 25
3.12 Revertir Cambios en un Archivo: . . . 25
3.13 Conclusión . . . 25

4 Asignación **26**

5 GitHub Classroom **27**
5.1 ¿Qué es GitHub Classroom? 27
5.1.1 Funcionalidades Principales 27
5.2 Ejemplo Práctico 28
5.2.1 Creación de una Asignación en GitHub Classroom 28
5.3 Trabajo de los Estudiantes 30

6 Docker **36**

7 Conceptos Básicos de Docker **37**
7.1 Imagen 37
7.2 Contenedor 37
7.3 Dockerfile 37
7.4 Docker Compose 38

8 Uso de Docker **39**
8.1 Definir un Dockerfile 39
8.2 Construir la Imagen 39
8.3 Ejecutar un Contenedor 40
8.4 Gestionar Contenedores 40
8.5 Docker Compose 40

II Unidad 2: Python Básico **42**

9 Hola Mundo en Python **43**

10 Introdución **44**
10.0.1 Paso 1: Instalación de Python 44
10.0.2 Paso 2: Instalación de Python en Windows 44
10.0.3 Paso 3: Crear nuestro primer “Hola Mundo” en Python 47

11 Sintaxis Básica **50**

12 Comentarios **51**

13 Variables y Tipos de Datos **52**

14 Tipos de Datos **53**

15 Operadores **55**

16 Estructura de Control **59**

17 Funciones **64**

18 Llamada a Funciones **65**

19 Parámetros y Argumentos **66**

20 Retorno **67**

21 Ejemplo **68**

22 Asignación **70**
22.1 Objetivo 70
22.2 ¿Qué debes hacer? 70
22.3 Pruebas 70
22.4 Ejecución 71

III Unidad 3: Python Intermedio **72**

23 Listas **73**

24 Tuplas **74**

25 Manipulación de Listas y Tuplas **75**

26 Funciones ingegradas para Listas y Tuplas **76**

27 Listas Anidadas **77**

28 Listas y Tuplas como Argumentos de Funciones **78**

29 Listas y Tuplas como Retorno de Funciones **79**

30 Asignación **80**
30.1 Descripción de la Asignación 80
30.2 Tarea Pendiente: 80
30.3 Cómo Ejecutar el Código 80
30.4 Ejemplo de salida: 80

31 Diccionarios **82**

32 Conjuntos **87**

33 Operaciones con Diccionarios y Conjuntos **91**

34 Asignación **93**
34.1 Descripción 93
34.2 Criterios de Evaluación 93

IV Unidad 4: Python Avanzado **94**

35 Programación Orientada a Objetos **95**
35.1 Asignación 101
35.2 Instrucciones 101
35.3 Contenido del Repositorio 101
35.4 Cómo Ejecutar las Pruebas 101

36 Módulos **102**

37 Paquetes **104**

38 Creación y Uso de Módulos **106**
38.1 Creación de Módulos 106
38.2 Uso de Módulos 106
38.3 Importar Funciones Específicas 106
38.4 Importar con Alias 107
38.5 Importar Todas las Funciones 107

39 Creación y Uso de Paquetes **108**
39.1 Creación de Paquetes 108
39.2 Uso de Paquetes 108
39.3 Importar con Alias 108
39.4 Importar Todas las Funciones 109

40 Asignación Calculadora Pythonica **110**
40.1 Instrucciones 110
40.2 Contenido del Repositorio 110
40.3 Ejercicio 110
40.4 Cómo Ejecutar el Programa 111
40.5 Cómo Ejecutar las Pruebas 111

41 Sistema de Gestión de Inventarios **112**
41.1 Asignación 112

V Unidad 5: Django **113**

42 Introducción a Django **114**
42.1 Conceptos Importantes 114
42.1.1 Entornos Virtuales 115
42.1.2 Modelo Template View (MTV) 116
42.1.3 Formularios 117
42.1.4 Administrador de Django 118
42.1.5 Middleware 118
42.1.6 Autenticación y Autorización 118
42.1.7 Internacionalización 118
42.1.8 Seguridad 119
42.1.9 Testing 119
42.1.10 Despliegue 119

43 Configuración inicial de un proyecto. **120**
43.1 1. Crear un entorno virtual 120
43.2 2. Activar el entorno virtual 120
43.3 3. Instalar Django 121
43.4 4. Crear un proyecto de Django 121
43.5 5. Crear una aplicación de Django 122
43.6 6. Crear una vista 122
43.7 7. Configurar las URL 123
43.8 8. Ejecutar el servidor de desarrollo 123

43.9 9. Crear una migración 124
43.1010. Aplicar una migración 125
43.1112. Crear un superusuario 125
43.1213. Acceder al panel de administración 126

44 Ejercicio **128**

45 Asignación **130**

46 Estructura de archivos y carpetas **131**

47 Creación de una aplicación Django **133**

48 Configuración de la base de datos **134**

49 Crear una vista **135**

50 Crear una plantilla **136**

51 Configurar las rutas **137**

52 Correr el servidor de desarrollo **138**

53 Acceder a la aplicación **139**

54 Acceder a la aplicación **140**

55 Asignación **141**

56 Referencias **143**

57 Modelos **144**

58 Registramos la aplicacion en admin.py **146**

59 Vistas en Django **147**
59.1 Listar productos 147
59.2 Agregar producto 147
59.3 Actualizar producto 148
59.4 Eliminar producto 148
59.5 Buscar producto 149

60 Templates **150**
60.1 Base 150
60.2 Listar 151
60.3 Agregar 151
60.4 Actualizar 152
60.5 Eliminar 152
60.6 Buscar 153

61 URLs **154**
61.1 URLs en la aplicación y el proyecto 154

62 Ejecutar el servidor **156**

63 Django Rest Framework **157**
63.1 Instalación . . . 157
63.2 Actualizar el archivo requirements.txt . . . 157
63.3 Serializers . . . 157
63.4 Views . . . 158
63.5 URLs de la Aplicación . . . 158
63.6 Configuración URLs del Proyecto . . . 158
63.7 Migraciones . . . 159
63.8 Ejecución . . . 159

64 Documentación de la API con drf-yasg **160**

65 Documentación de la API con CoreAPI **162**
65.1 Primero instalamos CoreAPI: . . . 162

66 Testing **163**

67 Corrección de tests en Django Rest Framework **165**
67.1 Introducción . . . 165
67.2 Pasos . . . 165
67.2.1 1. Crear un entorno virtual . . . 165
67.2.2 2. Activar el entorno virtual . . . 166
67.2.3 3. Instalar las dependencias . . . 166
67.2.4 4. Correr los tests . . . 166
67.2.5 5. Corregir los tests . . . 166
67.2.6 6. Correr los tests nuevamente . . . 166
67.2.7 7. Desactivar el entorno virtual . . . 166
67.3 Conclusión . . . 166
67.4 Referencias . . . 167

VI Unidad 6: Frontend **168**

68 Introducción a HTML **169**
68.1 Estructura básica de un documento HTML . . . 169
68.2 Etiquetas HTML . . . 170
68.3 Atributos HTML . . . 170
68.4 Ejemplo de un documento HTML . . . 171
68.5 Recursos adicionales . . . 171
68.6 Ejercicios . . . 171

69 Introducción a CSS **173**
69.1 ¿Qué es CSS? . . . 173
69.2 ¿Cómo funciona CSS? . . . 173
69.3 ¿Cómo se aplica CSS a un documento HTML? . . . 173
69.4 Sintaxis de CSS . . . 173
69.5 Ejemplo de CSS . . . 174
69.6 Comentarios en CSS . . . 174

69.7 Selectores CSS . . . 174
69.8 Propiedades CSS . . . 176
69.9 Unidades CSS . . . 180

70 JavaScript **181**
70.1 ¿Qué es JavaScript? . . . 181
70.2 ¿Dónde puedo aprender JavaScript? . . . 181
70.3 ¿Qué puedo hacer con JavaScript? . . . 181
70.4 ¿Qué es un script de JavaScript? . . . 182
70.5 ¿Dónde puedo ejecutar JavaScript? . . . 182
70.6 ¿Qué es Node.js? . . . 182
70.7 ¿Dónde puedo aprender Node.js? . . . 182
70.8 ¿Qué puedo hacer con Node.js? . . . 182

71 Primeros pasos con JavaScript **183**
71.1 Ecmascript 6 . . . 183
71.2 Motor V8 . . . 183
71.3 Asincronismo . . . 184
71.4 Programación Orientada a Objetos . . . 184
71.5 Programación Funcional . . . 184
71.6 Hola Mundo en JavaScript . . . 184
71.7 Comentarios en JavaScript . . . 185
71.8 Variables en JavaScript . . . 185
71.9 Tipos de datos en JavaScript . . . 186
71.10 Operadores en JavaScript . . . 186
71.11 Condicionales en JavaScript . . . 187
71.12 Bucles en JavaScript . . . 188
71.13 Funciones en JavaScript . . . 188
71.14 Eventos en JavaScript . . . 188
71.15 Objetos en JavaScript . . . 189
71.16 Arrays en JavaScript . . . 189
71.17 Métodos en JavaScript . . . 190
71.18 Clases en JavaScript . . . 190
71.19 Herencia en JavaScript . . . 191
71.20 Promesas en JavaScript . . . 191
71.21 Async/Await en JavaScript . . . 192
71.22 Consumir una api con fetch . . . 193
71.23 Crear una aplicación web con JavaScript . . . 194
71.24 Crear un juego con JavaScript . . . 195
71.25 Conclusiones . . . 196

72 Referencias **197**

73 Nodejs **198**
73.1 ¿Qué es nodejs? . . . 198
73.2 ¿Por qué nodejs? . . . 198
73.3 ¿Cómo instalar nodejs? . . . 198
73.4 ¿Cómo ejecutar un archivo de nodejs? . . . 198
73.5 ¿Qué es npm? . . . 199

73.6 ¿Cómo instalar un paquete de npm? . . . 199
73.7 ¿Cómo crear un proyecto de nodejs? . . . 199
73.8 ¿Cómo instalar express? . . . 199
73.9 ¿Cómo instalar nodemon? . . . 199
73.10 ¿Cómo instalar mongoose? . . . 199
73.11 ¿Cómo instalar dotenv? . . . 200
73.12 ¿Cómo instalar bcrypt? . . . 200
73.13 ¿Cómo instalar jsonwebtoken? . . . 200
73.14 ¿Cómo instalar cors? . . . 200
73.15 ¿Cómo empiezo mis primeros pasos con React? . . . 200
73.16 ¿Qué es vite? . . . 201
73.17 Conclusiones . . . 201
73.18 Referencias . . . 201

74 Introducción a React **202**
74.1 ¿Qué es Vite? . . . 203
74.2 Crear un proyecto con Vite . . . 203
74.3 Entendiendo React . . . 204
74.3.1 src/App.jsx . . . 204
74.3.2 src/main.jsx . . . 204
74.4 Hello World con React . . . 204
74.5 Conclusiones . . . 205
74.6 Ejericios . . . 205

75 Primeros pasos con React **207**
75.1 ¿Qué es React? . . . 207
75.2 Componentes . . . 207
75.3 JSX . . . 207
75.4 Props . . . 207
75.5 State . . . 207
75.6 Ciclo de vida . . . 208
75.7 Hooks . . . 208
75.8 Context . . . 208
75.9 Router . . . 208
75.10 Redux . . . 208
75.11 Práctica . . . 209
75.11.1 Crear la aplicación . . . 209
75.11.2 Crear el componente del contador . . . 209
75.11.3 Crear el componente principal . . . 210
75.11.4 Crear la aplicación principal . . . 210
75.11.5 Ejecutar la aplicación . . . 210
75.11.6 Resultado . . . 211
75.12 Conclusiones . . . 211
75.13 Ejercicios . . . 211
75.13.1 Solución ejercicio 1 . . . 212
75.13.2 Solución ejercicio 2 . . . 212

VII Ejercicios **213**

76 Ejercicios de Git y Github **214**
76.0.1 Ejercicio 1 214
76.0.2 Ejercicio 2 214
76.0.3 Ejercicio 3 214
76.0.4 Ejercicio 4 215
76.0.5 Ejercicio 5 215

77 Ejercicios Python - Nivel 1 **216**
77.1 Ejercicio 1 216
77.2 Ejercicio 2 216
77.3 Ejercicio 3 216
77.4 Ejercicio 4 217
77.5 Ejercicio 5 217

78 Ejercicios Python - Nivel 2 **218**
78.1 Ejercicio 1 218
78.2 Ejercicio 2 218
78.3 Ejercicio 3 218
78.4 Ejercicio 4 219
78.5 Ejercicio 5 219

79 Ejercicios Python - Nivel 3 **220**
79.1 Ejercicio 1: 220
79.2 Ejercicio 2: 220
79.3 Ejercicio 3: 220
79.4 Ejercicio 4: 221
79.5 Ejercicio 5: 221

80 Ejercicios Python - Nivel 4 **222**
80.1 Ejercicio 1: 222
80.2 Ejercicio 2: 222
80.3 Ejercicio 3: 223
80.4 Ejercicio 4: 223
80.5 Ejercicio 5: 224

1 Bienvenido

¡Bienvenido al Curso Completo de Django y React!

En este curso, exploraremos todo, desde los fundamentos hasta las aplicaciones prácticas.

1.1 ¿De qué trata este curso?

Este curso completo me llevará desde los fundamentos básicos de la programación hasta la construcción de aplicaciones prácticas utilizando los frameworks Django y la biblioteca de React.

A través de una combinación de teoría y ejercicios prácticos, me sumergiré en los conceptos esenciales del desarrollo web y avanzaré hacia la creación de proyectos del mundo real.

Desde la configuración del entorno de desarrollo hasta la construcción de una aplicación web de pila completa, este curso me proporcionará una comprensión sólida y experiencia práctica con Django y React.

1.2 ¿Para quién es este curso?

Este curso está diseñado para principiantes y aquellos con poca o ninguna experiencia en programación.

Ya sea que sea un estudiante curioso, un profesional que busca cambiar de carrera o simplemente alguien que quiere aprender desarrollo web, este curso es para usted. Desde adolescentes hasta adultos, todos son bienvenidos a participar y explorar el emocionante mundo del desarrollo web con Django y React.

1.3 ¿Cómo contribuir?

Valoramos su contribución a este curso. Si encuentra algún error, desea sugerir mejoras o agregar contenido adicional, me encantaría saber de usted.

Puede contribuir a través del repositorio en linea, donde puede compartir sus comentarios y sugerencias.

Juntos, podemos mejorar continuamente este recurso educativo para beneficiar a la comunidad de estudiantes y entusiastas de la programación.

Este ebook ha sido creado con el objetivo de proporcionar acceso gratuito y universal al conocimiento.

Estará disponible en línea para cualquier persona, sin importar su ubicación o circunstancias, para acceder y aprender a su propio ritmo.

Puede descargarlo en formato PDF, Epub o verlo en línea en cualquier momento y lugar.

Esperamos que disfrute este emocionante viaje de aprendizaje y descubrimiento en el mundo del desarrollo web con Django y React!

Part I

Unidad 1: Introducción a Python

2 Git y GitHub

Figure 2.1: Git and Github

2.1 ¿Qué es Git y GitHub?

Git y GitHub son herramientas ampliamente utilizadas en el desarrollo de software para el control de versiones y la colaboración en proyectos.

Git es un sistema de control de versiones distribuido que permite realizar un seguimiento de los cambios en el código fuente durante el desarrollo de software. Fue creado por Linus Torvalds en 2005 y se utiliza mediante la línea de comandos o a través de interfaces gráficas de usuario.

GitHub, por otro lado, es una plataforma de alojamiento de repositorios Git en la nube. Proporciona un entorno colaborativo donde los desarrolladores pueden compartir y trabajar en proyectos de software de forma conjunta. Además, ofrece características adicionales como seguimiento de problemas, solicitudes de extracción y despliegue continuo.

En este tutorial, aprenderás los conceptos básicos de Git y GitHub, así como su uso en un proyecto de software real.

2.2 ¿Quiénes utilizan Git?

Figure 2.2: Git

Es ampliamente utilizado por desarrolladores de software en todo el mundo, desde estudiantes hasta grandes empresas tecnológicas. Es una herramienta fundamental para el desarrollo colaborativo y la gestión de proyectos de software.

2.3 ¿Cómo se utiliza Git?

```
commit e072c20b5577c37af7c4fb274b6b53d15dd336ae
Author: Julio Xavier <julioxavierr@live.com>
Date:   Fri Aug 19 16:17:10 2016 -0300

    Commit with error

commit a497c0c03657549e7d4c5ba1b23ffce5faaf46b8
Author: Julio Xavier <julioxavierr@live.com>
Date:   Mon Jan 11 10:51:42 2016 -0200

    Adding common html code in a form

commit 9fa7605ad1837aa44dfb9c711dc8bd60cabc7c5d
Author: Julio Xavier <julioxavierr@live.com>
Date:   Sun Jan 10 22:29:52 2016 -0200

    Pages to show 'details' + Editing Clients
```

Figure 2.3: Git en Terminal

Se utiliza mediante la **línea de comandos** o a través de **interfaces gráficas** de usuario. Proporciona comandos para realizar operaciones como:

1. Inicializar un repositorio,
2. Realizar cambios,
3. Revisar historial,
4. Fusionar ramas,
5. Entre otros.

2.4 ¿Para qué sirve Git?

Figure 2.4: Seguimiento de Cambios con Git

Sirve para realizar un seguimiento de los cambios en el código fuente, coordinar el trabajo entre varios desarrolladores, revertir cambios no deseados y mantener un historial completo de todas las modificaciones realizadas en un proyecto.

2.5 ¿Por qué utilizar Git?

Figure 2.5: Ventajas de Git

Ofrece varias ventajas, como:

- La capacidad de trabajar de forma distribuida
- La gestión eficiente de ramas para desarrollar nuevas funcionalidades
- Corregir errores sin afectar la rama principal
- La posibilidad de colaborar de forma efectiva con otros desarrolladores.

2.6 ¿Dónde puedo utilizar Git?

Figure 2.6: Git en Diferentes Sistemas Operativos

Puede ser utilizado en cualquier sistema operativo, incluyendo Windows, macOS y Linux. Además, es compatible con una amplia variedad de plataformas de alojamiento de repositorios, siendo GitHub una de las más populares.

2.7 Pasos Básicos

> **Tip**
>
> Es recomendable tomar en cuenta una herramienta para la edición de código, como Visual Studio Code, Sublime Text o Atom, para trabajar con Git y GitHub de manera eficiente.

2.8 Instalación de Visual Studio Code

Figure 2.7: Visual Studio Code

Si aún no tienes Visual Studio Code instalado, puedes descargarlo desde https://code.visualstudio.com/download. Es una herramienta gratuita y de código abierto que proporciona una interfaz amigable para trabajar con Git y GitHub.

A continuación se presentan los pasos básicos para utilizar Git y GitHub en un proyecto de software.

2.8.1 Descarga e Instalación de Git

Figure 2.8: Git

1. Visita el sitio web oficial de Git en https://git-scm.com/downloads.
2. Descarga el instalador adecuado para tu sistema operativo y sigue las instrucciones de instalación.

2.8.2 Configuración

Figure 2.9: Configuración de Git

Una vez instalado Git, es necesario configurar tu nombre de usuario y dirección de correo electrónico. Esto se puede hacer mediante los siguientes comandos:

```
git config --global user.name "Tu Nombre"
git config --global user.email "tu@email.com"
```

2.8.3 Creación de un Repositorio "helloWorld" en Python

- Crea una nueva carpeta para tu proyecto y ábrela en Visual Studio Code.
- Crea un archivo Python llamado **hello_world.py** y escribe el siguiente código:

```
def welcome_message():
    name = input("Ingrese su nombre: ")
    print("Bienvenio,", name, "al curso de Django y React!")

if __name__ == "__main__":
    welcome_message()
```

- Guarda el archivo y abre una terminal en Visual Studio Code.
- Inicializa un repositorio Git en la carpeta de tu proyecto con el siguiente comando:

```
git init
```

- Añade el archivo al área de preparación con:

```
git add hello_world.py
```

- Realiza un commit de los cambios con un mensaje descriptivo:

```
git commit -m "Añadir archivo hello_world.py"
```

2.8.4 Comandos Básicos de Git

- **git init:** Inicializa un nuevo repositorio Git.
- **git add :** Añade un archivo al área de preparación.
- **git commit -m "":** Realiza un commit de los cambios con un mensaje descriptivo.
- **git push:** Sube los cambios al repositorio remoto.
- **git pull:** Descarga cambios del repositorio remoto.
- **git branch:** Lista las ramas disponibles.
- **git checkout :** Cambia a una rama específica.
- **git merge :** Fusiona una rama con la rama actual.
- **git reset :** Descarta los cambios en un archivo.
- **git diff:** Muestra las diferencias entre versiones.

2.8.5 Estados en Git

- **Local:** Representa los cambios que realizas en tu repositorio local antes de hacer un commit. Estos cambios están únicamente en tu máquina.
- **Staging:** Indica los cambios que has añadido al área de preparación con el comando `git add`. Estos cambios están listos para ser confirmados en el próximo commit.
- **Commit:** Son los cambios que has confirmado en tu repositorio local con el comando `git commit`. Estos cambios se han guardado de manera permanente en tu repositorio local.
- **Server:** Son los cambios que has subido al repositorio remoto con el comando `git push`. Estos cambios están disponibles para otros colaboradores del proyecto.

---

3 Tutorial: Moviendo Cambios entre Estados en Git

3.1 Introducción

En este tutorial, aprenderemos a utilizar Git para gestionar cambios en nuestro proyecto y moverlos entre diferentes estados. Utilizaremos un ejemplo práctico para comprender mejor estos conceptos.

```
def welcome_message():
    name = input("Ingrese su nombre: ")
    print("Bienvenio,", name, "al curso de Django y React!")

if __name__ == "__main__":
    welcome_message()
```

3.2 Sección 1: Modificar Archivos en el Repositorio

En esta sección, aprenderemos cómo realizar cambios en nuestros archivos y reflejarlos en Git.

3.3 Mover Cambios de Local a Staging:

1. Abre el archivo **hello_world.py** en Visual Studio Code.
2. Modifica el mensaje de bienvenida a "Bienvenido" en lugar de "Bienvenio".
3. Guarda los cambios y abre una terminal en Visual Studio Code.

Hemos corregido un error en nuestro archivo y queremos reflejarlo en Git.

```
def welcome_message():
    name = input("Ingrese su nombre: ")
    print("Bienvenido,", name, "al curso de Django y React!")

if __name__ == "__main__":
    welcome_message()
```

3.4 Agregar Cambios de Local a Staging:

```
git add hello_world.py
```

Hemos añadido los cambios al área de preparación y están listos para ser confirmados en el próximo commit.

3.5 Sección 2: Confirmar Cambios en un Commit

En esta sección, aprenderemos cómo confirmar los cambios en un commit y guardarlos de manera permanente en nuestro repositorio.

3.6 Mover Cambios de Staging a Commit:

```
git commit -m "Corregir mensaje de bienvenida"
```

Hemos confirmado los cambios en un commit con un mensaje descriptivo.

3.7 Sección 3: Creación y Fusión de Ramas

En esta sección, aprenderemos cómo crear y fusionar ramas en Git para desarrollar nuevas funcionalidades de forma aislada.

3.8 Crear una Nueva Rama:

```
git branch feature
```

Hemos creado una nueva rama llamada "feature" para desarrollar una nueva funcionalidad.

3.9 Implementar Funcionalidades en la Rama:

1. Abre el archivo **hello_world.py** en Visual Studio Code.
2. Añade una nueva función para mostrar un mensaje de despedida.
3. Guarda los cambios y abre una terminal en Visual Studio Code.
4. Añade los cambios al área de preparación y confírmalos en un commit.
5. Cambia a la rama principal con `git checkout main`.

3.10 Fusionar Ramas con la Rama Principal:

```
git merge feature
```

Hemos fusionado la rama "feature" con la rama principal y añadido la nueva funcionalidad al proyecto.

3.11 Sección 4: Revertir Cambios en un Archivo

En esta sección, aprenderemos cómo revertir cambios en un archivo y deshacerlos en Git.

3.12 Revertir Cambios en un Archivo:

```
git reset hello_world.py
```

Hemos revertido los cambios en el archivo **hello_world.py** y deshecho las modificaciones realizadas.

3.13 Conclusión

En este tutorial, hemos aprendido a gestionar cambios en nuestro proyecto y moverlos entre diferentes estados en Git. Estos conceptos son fundamentales para trabajar de forma eficiente en proyectos de software y colaborar con otros desarrolladores.

4 Asignación

Hello World!

Este proyecto de ejemplo está escrito en Python y se prueba con pytest.

La Asignación

Las pruebas están fallando en este momento porque el método no está devolviendo la cadena correcta. Corrige el código del archivo **hello.py** para que las pruebas sean exitosas, debe devolver la cadena correcta **"Hello World!"**x

El comando de ejecución del test es:

```
pytest test_hello.py
```

¡Mucha suerte!

5 GitHub Classroom

Figure 5.1: Github Classroom

GitHub Classroom es una herramienta poderosa que facilita la gestión de tareas y asignaciones en GitHub, especialmente diseñada para entornos educativos.

5.1 ¿Qué es GitHub Classroom?

Figure 5.2: Github Classroom Windows

GitHub Classroom es una extensión de GitHub que permite a los profesores crear y gestionar asignaciones utilizando repositorios de GitHub. Proporciona una forma organizada y eficiente de distribuir tareas a los estudiantes, recopilar y revisar su trabajo, y proporcionar retroalimentación.

5.1.1 Funcionalidades Principales

Creación de Asignaciones: Los profesores pueden crear tareas y asignaciones directamente desde GitHub Classroom, proporcionando instrucciones detalladas y estableciendo

criterios de evaluación.

Distribución Automatizada: Una vez que se crea una asignación, GitHub Classroom genera automáticamente repositorios privados para cada estudiante o equipo, basándose en una plantilla predefinida.

Seguimiento de Progreso: Los profesores pueden realizar un seguimiento del progreso de los estudiantes y revisar sus contribuciones a través de solicitudes de extracción (pull requests) y comentarios en el código.

Revisión y Retroalimentación: Los estudiantes envían sus trabajos a través de solicitudes de extracción, lo que permite a los profesores revisar y proporcionar retroalimentación específica sobre su código.

5.2 Ejemplo Práctico

5.2.1 Creación de una Asignación en GitHub Classroom

Iniciar Sesión: Ingresa a GitHub Classroom con tu cuenta de GitHub y selecciona la opción para crear una nueva asignación.

Definir la Tarea: Proporciona instrucciones claras y detalladas sobre la tarea, incluyendo cualquier código base o recursos necesarios. Establece los criterios de evaluación para guiar a los estudiantes.

Configurar la Plantilla: Selecciona una plantilla de repositorio existente o crea una nueva plantilla que servirá como base para los repositorios de los estudiantes.

Distribuir la Asignación: Una vez configurada la asignación, comparte el enlace generado con tus estudiantes para que puedan acceder a sus repositorios privados.

5.3 Trabajo de los Estudiantes

Aceptar la Asignación: Los estudiantes reciben el enlace de la asignación y aceptan la tarea, lo que les permite crear un repositorio privado basado en la plantilla proporcionada.

Actualizar el Navegador: Los estudiantes actualizan su navegador para ver el nuevo repositorio creado en su cuenta de GitHub.

Clonar el Repositorio: Los estudiantes clonan el repositorio asignado en su computadora local utilizando el enlace proporcionado.

Utilizar el comando git clone: Aplique el comando git clone para clonar el repositorio en su computadora local.

```
git clone <enlace-del-repositorio>
```

```
Desktop :: pwsh
~\Desktop  git clone https://github.com/Coding-Academy-ec/hello-world-student-pruebas.git
Cloning into 'hello-world-student-pruebas'...
remote: Enumerating objects: 19, done.
remote: Counting objects: 100% (19/19), done.
remote: Compressing objects: 100% (14/14), done.
remote: Total 19 (delta 4), reused 3 (delta 0), pack-reused 0
Receiving objects: 100% (19/19), 4.69 KiB | 1.17 MiB/s, done.
Resolving deltas: 100% (4/4), done.
```

Desarrollar la Tarea: Los estudiantes trabajan en la tarea, realizando los cambios necesarios y realizando commits de manera regular para mantener un historial de su trabajo.

> **Tip**
>
> Puedes probar el test incorporado con el comando `pytest` en la terminal, para verificar que el código cumple con los requerimientos

```
pytest
```

Una vez desarrollado el código de acuerdo a la asignación en local deberían pasar el o los test

Enviar la Solicitud de Extracción: Una vez completada la tarea, los estudiantes envían una solicitud de extracción desde su rama hacia la rama principal del repositorio, solicitando la revisión del profesor.

Una vez realizado el `push` se envía al respositorio principal y se ejecutan los test en Github

> **Tip**
>
> Se recomienda hacer las pruebas en local antes de enviar los cambios al respositorio en Github

Este Action lo que hace es evaluar los cambios realizados

> **Tip**
>
> Se recomienda hacer las pruebas en local antes de enviar los cambios al respositorio en Github

Revisión y Retroalimentación: Los profesores revisan las solicitudes de extracción, proporcionan comentarios sobre el código y evalúan el trabajo de los estudiantes según los criterios establecidos.

Tip

GitHub Classroom ofrece una manera eficiente y organizada de administrar tareas y asignaciones en entornos educativos, fomentando la colaboración, el aprendizaje y la retroalimentación efectiva entre profesores y estudiantes.

6 Docker

Figure 6.1: Docker

Docker es una plataforma de código abierto que permite a los desarrolladores empaquetar, distribuir y ejecutar aplicaciones en contenedores. Los contenedores son entornos ligeros y portátiles que incluyen todo lo necesario para ejecutar una aplicación de forma consistente en cualquier entorno.

7 Conceptos Básicos de Docker

7.1 Imagen

```
docker pull python:3.9-slim
```

Una imagen de Docker es un paquete de software ligero y portátil que incluye todo lo necesario para ejecutar una aplicación, incluidos el código, las bibliotecas y las dependencias. Las imágenes se utilizan como plantillas para crear contenedores.

7.2 Contenedor

```
docker run -d -p 5000:5000 myapp
```

Un contenedor de Docker es una instancia en tiempo de ejecución de una imagen de Docker. Los contenedores son entornos aislados que ejecutan aplicaciones de forma independiente y comparten recursos del sistema operativo subyacente. Cada contenedor está aislado del entorno de host y otros contenedores, lo que garantiza la consistencia y la portabilidad de las aplicaciones.

7.3 Dockerfile

```
# Dockerfile
# Define la imagen base
FROM python:3.9-slim

# Instala las dependencias necesarias
RUN apt-get update && apt-get install -y \
    build-essential \
    libpq-dev \
    libffi-dev \
    && rm -rf /var/lib/apt/lists/*

# Establece el directorio de trabajo
WORKDIR /app
```

```
# Copia los archivos de la aplicación al contenedor
COPY . .

# Instala las dependencias de Python
RUN pip install --no-cache-dir -r requirements.txt

# Establece el comando por defecto para ejecutar la aplicación
CMD ["python", "app.py"]
```

Un Dockerfile es un archivo de texto que contiene instrucciones para construir una imagen de Docker. Especifica qué software se instalará en la imagen y cómo configurar el entorno de ejecución. Los Dockerfiles permiten a los desarrolladores definir de manera reproducible el entorno de ejecución de sus aplicaciones y automatizar el proceso de construcción de imágenes.

7.4 Docker Compose

```
# docker-compose.yml
version: '3'
services:
  web:
    build: .
    ports:
      - "5000:5000"
    volumes:
      - .:/app
    environment:
      FLASK_ENV: development
```

Docker Compose es una herramienta que permite definir y ejecutar aplicaciones Docker multi-contenedor. Permite gestionar la configuración de varios contenedores como un solo servicio, lo que facilita el despliegue y la gestión de aplicaciones complejas que constan de múltiples componentes.

8 Uso de Docker

8.1 Definir un Dockerfile

```
# Dockerfile
# Define la imagen base
FROM python:3.9-slim

# Instala las dependencias necesarias
RUN apt-get update && apt-get install -y \
    build-essential \
    libpq-dev \
    libffi-dev \
    && rm -rf /var/lib/apt/lists/*

# Establece el directorio de trabajo
WORKDIR /app

# Copia los archivos de la aplicación al contenedor
COPY . .

# Instala las dependencias de Python
RUN pip install --no-cache-dir -r requirements.txt

# Establece el comando por defecto para ejecutar la aplicación
CMD ["python", "app.py"]
```

Para utilizar Docker, primero se crea un Dockerfile que especifica cómo construir la imagen de Docker, incluidas las dependencias y la configuración del entorno. El Dockerfile define las capas de la imagen y las instrucciones para configurar el entorno de ejecución de la aplicación.

8.2 Construir la Imagen

```
docker build -t myapp .
```

Una vez que se tiene el Dockerfile, se utiliza el comando docker build para construir la imagen de Docker a partir del Dockerfile. Este comando lee las instrucciones del Dockerfile

y crea una imagen en función de esas instrucciones. La imagen resultante se puede utilizar para crear y ejecutar contenedores.

8.3 Ejecutar un Contenedor

```
docker run -d -p 5000:5000 myapp
```

Después de construir la imagen, se ejecuta un contenedor utilizando el comando docker run, especificando la imagen que se utilizará y cualquier configuración adicional necesaria, como puertos expuestos, variables de entorno y volúmenes montados. El contenedor se ejecuta en un entorno aislado y se puede acceder a través de la red local o de Internet, según la configuración.

8.4 Gestionar Contenedores

```
docker ps
docker stop <container_id>
docker rm <container_id>
```

Docker proporciona varios comandos para gestionar contenedores, como docker ps para ver contenedores en ejecución, docker stop para detener un contenedor y docker rm para eliminar un contenedor. Estos comandos permiten a los usuarios administrar y controlar el ciclo de vida de los contenedores de manera eficiente.

8.5 Docker Compose

```
# docker-compose.yml
version: '3'
services:
  web:
    build: .
    ports:
      - "5000:5000"
    volumes:
      - .:/app
    environment:
      FLASK_ENV: development
```

Para aplicaciones más complejas que requieren múltiples contenedores, se utiliza Docker Compose para definir y gestionar la configuración de los contenedores en un archivo

YAML. Luego, se utiliza el comando docker-compose para gestionar los servicios definidos en el archivo YAML, lo que simplifica el despliegue y la gestión de aplicaciones multi-contenedor.

Part II

Unidad 2: Python Básico

9 Hola Mundo en Python

10 Introdución

En este tutorial aprenderemos los conceptos básicos necesarios para configurar nuestro entorno de desarrollo y escribir código en Python. Comenzaremos con la instalación de Python en Windows y luego veremos cómo escribir y ejecutar nuestro primer programa en Python utilizando Visual Studio Code como nuestro editor de texto.

10.0.1 Paso 1: Instalación de Python

Para poder escribir y ejecutar programas en Python, primero necesitamos instalar Python en nuestra computadora. Python es un lenguaje de programación de alto nivel que es ampliamente utilizado en el desarrollo de aplicaciones web, desarrollo de software, análisis de datos, inteligencia artificial, etc.

> **Note**
>
> Python se puede instalar en Windows, Mac y Linux. En este tutorial, veremos cómo instalar Python en Windows.

10.0.2 Paso 2: Instalación de Python en Windows

1. **Descargar Python**

Para instalar Python en Windows, primero necesitamos descargar el instalador de Python desde el sitio web oficial de Python. Para hacer esto, abra su navegador web y vaya a la página de descargas de Python en el siguiente enlace:

https://www.python.org/downloads/

En la página de descargas, haga clic en el botón de descarga para la última versión de Python. En el momento de escribir este tutorial, la última versión de Python es 3.12.1

Excelente, ahora que hemos descargado el instalador de Python, podemos continuar con la instalación de Python en Windows.

2. **Instalar Python**

Una vez que el instalador de Python se haya descargado, haga doble clic en el archivo de instalación para iniciar el proceso de instalación. Asegúrese de marcar la casilla que dice "Add Python 3.12 to PATH" antes de hacer clic en el botón "Install Now".

Ahora que hemos instalado Python en nuestra computadora, podemos continuar con la configuración de nuestro entorno de desarrollo para escribir y ejecutar programas en Python.

3. **Comprobar que tenemos instalado Python**

Para comprobar que Python se ha instalado correctamente en nuestra computadora, abra una ventana de comandos y escriba el siguiente comando:

```
python --version
```

4. **Impresion de la versión de Python**

Este comando imprimirá la versión de Python que está instalada en su computadora. En mi caso, la versión de Python es 3.12.1.

10.0.3 Paso 3: Crear nuestro primer “Hola Mundo” en Python .

Ahora que hemos instalado Python en nuestra computadora, podemos comenzar a escribir y ejecutar programas en Python. Para hacer esto, necesitamos un editor de texto para escribir nuestro código y un intérprete de Python para ejecutar nuestro código.

En este tutorial, usaremos Visual Studio Code como nuestro editor de texto y el intérprete de Python que instalamos en el paso anterior.

1. **Instalar Visual Studio Code**

Para instalar Visual Studio Code, vaya al sitio web oficial de Visual Studio Code en el siguiente enlace:

https://code.visualstudio.com/

En la página de descargas, haga clic en el botón de descarga para su sistema operativo. En el momento de escribir este tutorial, la última versión de Visual Studio Code es 1.85.2.

Una vez que el instalador de Visual Studio Code se haya descargado, haga doble clic en el archivo de instalación para iniciar el proceso de instalación. Siga las instrucciones en pantalla para completar la instalación.

2. **Crear un nuevo archivo de Python**

Para crear un nuevo archivo de Python en Visual Studio Code, abra Visual Studio Code y haga clic en el botón "New File" en la barra de herramientas. Luego, escriba el siguiente código en el archivo:

```
print("Hola Mundo")
```

3. Este código imprimirá "Hola Mundo" en la consola.

4. Ejecutar el programa

Para ejecutar el programa, haga clic en el botón “Run” en la barra de herramientas. Esto ejecutará el programa y mostrará “Hola Mundo” en la consola.

¡Felicidades!

Acabas de escribir y ejecutar tu primer programa en Python. Ahora que has configurado tu entorno de desarrollo y has escrito tu primer programa en Python, puedes comenzar a explorar el lenguaje de programación Python y aprender a escribir programas más complejos.

11 Sintaxis Básica

Figure 11.1: Python

```
if 5 > 2:
  print("Cinco es mayor que dos")                                    ①
```

① En el ejemplo anterior, la instrucción **print** está indentada, es decir, tiene un espacio al principio de la línea. Esto es necesario para que el código funcione correctamente.

Además de la indentación, en python se utilizan los dos puntos **:** para indicar que se va a escribir un bloque de código.

```
if 5 > 2:                                                            ①
  print("Cinco es mayor que dos")
```

① En este caso, el **:** indica que se va a escribir un bloque de código, el bloque de código que se ejecutará si la condición es verdadera.

12 Comentarios

Los comentarios en python se escriben con el símbolo #.

```
# Este es un comentario
print("Hola Mundo")                                                    ①
```

① En este caso, el comentario está al final de la línea de código.

13 Variables y Tipos de Datos

Tip

Existe la forma de declarar variables de python con su tipo de dato
Ejemplo:

```
x: int = 5                                  ①
y: str = "Hola Mundo"                       ②
```

① En este caso, la variable **x** es de tipo entero.
② En este caso, la variable **y** es de tipo cadena de texto.

En python, las variables se definen de la siguiente manera:

```
x = 5                                       ①
y = "Hola Mundo"                            ②
```

① En este caso, la variable **x** es de tipo entero.
② En este caso, la variable **y** es de tipo cadena de texto.

14 Tipos de Datos

En python, los tipos de datos más comunes son:

- **int**: Entero

Ejemplo:

```
x = 5 ①
```

① La variable **x** es de tipo entero.

- **float**: Flotante

Ejemplo:

```
y = 5.0 ①
```

① La variable **y** es de tipo flotante.

- **str**: Cadena de texto

Ejemplo:

```
z = "Hola Mundo" ①
```

- **bool**: Booleano

Ejemplo:

```
a = True ①
```

① La variable **a** es de tipo booleano.

- **list**: Lista

Ejemplo:

```
b = [1, 2, 3] ①
```

- **tuple**: Tupla

Ejemplo:

```
c = (1, 2, 3)  ①
```

- **dict**: Diccionario

Ejemplo:

```
d = {"nombre": "Juan", "edad": 25}  ①
```

- **set**: Conjunto

Ejemplo:

```
e = {1, 2, 3}  ①
```

① La variable **e** es de tipo conjunto.

- **None**: Nulo

Ejemplo:

```
f = None  ①
```

① La variable **f** es de tipo **None**.

15 Operadores

En python, los operadores más comunes son:

- +: Suma

Ejemplo:

```
x = 5
y = 2
z = x + y                                                    ①
```

① La variable **z** es igual a la suma de **x** y **y**.

- -: Resta

Ejemplo:

```
x = 5
y = 2
a = x - y                                                    ①
```

① La variable **a** es igual a la resta de **x** y **y**.

- **“*“**: Multiplicación

Ejemplo:

```
x = 5
y = 2
b = x * y                                                    ①
```

① La variable **b** es igual a la multiplicación de **x** y **y**.

- /: División

Ejemplo:

```
x = 5
y = 2
c = x / y                                                    ①
```

① La variable **c** es igual a la división de **x** y **y**.

- //: División entera

Ejemplo:

```
x = 5
y = 2
d = x // y                                                    ①
```

① La variable **d** es igual a la división entera de **x** y **y**.

- %: Módulo

Ejemplo:

```
x = 5
y = 2
e = x % y                                                     ①
```

① La variable **e** es igual al módulo de **x** y **y**.

- “”**: Potencia

Ejemplo:

```
x = 5
y = 2
f = x ** y                                                    ①
```

① La variable **f** es igual a la potencia de **x** y **y**.

- ==: Igualdad

Ejemplo:

```
x = 5
y = 2
g = x == y                                                    ①
```

① La variable **g** es igual a la comparación de igualdad entre **x** y **y**.

- !=: Diferente

Ejemplo:

```
x = 5
y = 2
h = x != y                                                    ①
```

① La variable **h** es igual a la comparación de diferencia entre **x** y **y**.

- >: Mayor que

Ejemplo:

```
x = 5
y = 2
i = x > y                                                    ①
```

- <: Menor que

Ejemplo:

```
x = 5
y = 2
j = x < y                                                    ①
```

① La variable **j** es igual a la comparación de menor que entre **x** y **y**.

- >=: Mayor

Ejemplo:

```
x = 5
y = 2
k = x >= y                                                   ①
```

① La variable **k** es igual a la comparación de mayor o igual que entre **x** y **y**.

- <=: Menor

Ejemplo:

```
x = 5
y = 2
l = x <= y                                                   ①
```

① La variable **l** es igual a la comparación de menor o igual que entre **x** y **y**.

- **and**: Y

Ejemplo:

```
x = 5
y = 2
m = x and y                                                  ①
```

- **or**: O

Ejemplo:

```
x = 5
y = 2
n = x or y                                                   ①
```

① La variable **n** es igual a la comparación lógica **or** entre **x** y **y**.

- **not**: Negación

Ejemplo:

```
x = 5
o = not x                                                    ①
```

① La variable **o** es igual a la negación de **x**.

16 Estructura de Control

En python, las estructuras de control más comunes son:

- **if**: Si

Ejemplo:

```
if 5 > 2:                                                    ①
  print("Cinco es mayor que dos")                            ②
```

① En este caso, se evalúa si 5 es mayor que 2.
② Si la condición es verdadera, se imprime el mensaje "Cinco es mayor que dos".

- **elif**: Si no

Ejemplo:

```
x = 5
y = 2
if x > y:                                                    ①
  print("X es mayor que Y")                                  ②
elif x < y:                                                  ③
  print("X es menor que Y")                                  ④
```

① En este caso, se evalúa si **x** es mayor que **y**.
② Si la condición es verdadera, se imprime el mensaje "X es mayor que Y".
③ Si la condición anterior es falsa, se evalúa si **x** es menor que **y**.
④ Si la condición es verdadera, se imprime el mensaje "X es menor que Y".

- **else**: Si no

Ejemplo:

```
x = 5
y = 2
if x > y:                                                    ①
  print("X es mayor que Y")                                  ②
elif x < y:                                                  ③
  print("X es menor que Y")                                  ④
else:                                                        ⑤
  print("X es igual a Y")                                    ⑥
```

① En este caso, se evalúa si **x** es mayor que **y**.

② Si la condición es verdadera, se imprime el mensaje "X es mayor que Y".
③ Si la condición anterior es falsa, se evalúa si **x** es menor que **y**.
④ Si la condición es verdadera, se imprime el mensaje "X es menor que Y".
⑤ Si las condiciones anteriores son falsas, se ejecuta el bloque de código del **else**.
⑥ En este caso, se imprime el mensaje "X es igual a Y".

- **for**: Para

Ejemplo:

```
for x in range(5):                ①
  print(x)                        ②
```

① En este caso, se recorre un rango de 0 a 5.
② En cada iteración, se imprime el valor de **x**.

- **while**: Mientras

Ejemplo:

```
x = 0
while x < 5:                      ①
  print(x)                        ②
  x += 1                          ③
```

① En este caso, se evalúa si **x** es menor que 5.
② En cada iteración, se imprime el valor de **x**.
③ En cada iteración, se incrementa el valor de **x**.

- **break**: Romper

Ejemplo:

```
x = 0
while x < 5:                      ①
  print(x)                        ②
  x += 1                          ③
  if x == 3:                      ④
    break                         ⑤
```

① En este caso, se evalúa si **x** es menor que 5.
② En cada iteración, se imprime el valor de **x**.
③ En cada iteración, se incrementa el valor de **x**.
④ En cada iteración, se evalúa si **x** es igual a 3.
⑤ Si la condición es verdadera, se rompe el ciclo.

- **continue**: Continuar

Ejemplo:

```
x = 0
while x < 5:                ①
  x += 1                    ②
  if x == 3:                ③
    continue                ④
  print(x)                  ⑤
```

① En este caso, se evalúa si **x** es menor que 5.
② En cada iteración, se incrementa el valor de **x**.
③ En cada iteración, se evalúa si **x** es igual a 3.
④ Si la condición es verdadera, se continúa con la siguiente iteración.
⑤ En cada iteración, se imprime el valor de **x**.

- **pass**: Pasar

Ejemplo:

```
x = 0
if x == 0:                  ①
  pass                      ②
```

① En este caso, se evalúa si **x** es igual a 0.
② Si la condición es verdadera, no se hace nada.

- **return**: Retornar

Ejemplo:

```
def suma(x, y):             ①
  return x + y              ②
```

① En este caso, se define una función llamada **suma** que recibe dos parámetros **x** y **y**.
② La función retorna la suma de **x** y **y**.

- **def**: Definir

Ejemplo:

```
def suma(x, y):             ①
  return x + y              ②
```

① En este caso, se define una función llamada **suma** que recibe dos parámetros **x** y **y**.
② La función retorna la suma de **x** y **y**.

- **try**: Intentar

Ejemplo:

```
try:                                                    ①
  print(x)                                              ②
except:                                                 ③
  print("Ocurrió un error")                             ④
```

① En este caso, se intenta ejecutar el bloque de código.
② Si ocurre un error, se ejecuta el bloque de código del **except**.
③ En este caso, se imprime el mensaje "Ocurrió un error".
④ Si no ocurre un error, se continúa con la ejecución del código.

- **except**: Excepto

Ejemplo:

```
try:                                                    ①
  print(x)                                              ②
except:                                                 ③
  print("Ocurrió un error")                             ④
```

① En este caso, se intenta ejecutar el bloque de código.
② Si ocurre un error, se ejecuta el bloque de código del **except**.
③ En este caso, se imprime el mensaje "Ocurrió un error".
④ Si no ocurre un error, se continúa con la ejecución del código.

- **finally**: Finalmente

Ejemplo:

```
try:                                                    ①
  print(x)                                              ②
except:                                                 ③
  print("Ocurrió un error")                             ④
finally:                                                ⑤
  print("Finalizó la ejecución")                        ⑥
```

① En este caso, se intenta ejecutar el bloque de código.
② Si ocurre un error, se ejecuta el bloque de código del `except`.
③ En este caso, se imprime el mensaje "Ocurrió un error".
④ Si no ocurre un error, se continúa con la ejecución del código.
⑤ Al finalizar la ejecución del bloque de código, se ejecuta el bloque de código del `finally`.
⑥ En este caso, se imprime el mensaje "Finalizó la ejecución".

- **raise**: Lanzar

Ejemplo:

```
if x < 0:                                               ①
  raise Exception("El número no puede ser negativo")    ②
```

① En este caso, se evalúa si `x` es menor que 0.

② Si la condición es verdadera, se lanza una excepción con el mensaje "El número no puede ser negativo".

- **assert**: Afirmar

Ejemplo:

```
assert x > 0, "El número no puede ser negativo"  ①
```

① En este caso, se evalúa si `x` es mayor que 0.

- **import**: Importar

Ejemplo:

```
import math  ①
```

① En este caso, se importa el módulo **math**.

- **from**: Desde

Ejemplo:

```
from math import sqrt  ①
```

① En este caso, se importa la función **sqrt** del módulo **math**.

- **as**: Como

Ejemplo:

```
import math as m  ①
```

① En este caso, se importa el módulo **math** como **m**.

17 Funciones

En python, las funciones se definen de la siguiente manera:

```
def suma(x, y):                                                    ①
  return x + y                                                     ②
```

① En este caso, se define una función llamada **suma** que recibe dos parámetros **x** y **y**.
② La función retorna la suma de **x** y **y**.

18 Llamada a Funciones

En python, las funciones se llaman de la siguiente manera:

```
z = suma(5, 2)                                                    ①
```

① En este caso, se llama a la función **suma** con los argumentos **5** y **2**.

19 Parámetros y Argumentos

En python, los parámetros y argumentos se definen de la siguiente manera:

```
def suma(x, y):                ①
  return x + y                 ②

z = suma(5, 2)                 ③
```

① En este caso, se define una función llamada **suma** que recibe dos parámetros **x** y **y**.
② La función retorna la suma de **x** y **y**.
③ En este caso, se llama a la función **suma** con los argumentos **5** y **2**.

20 Retorno

En python, el retorno se realiza de la siguiente manera:

```
def suma(x, y):                                    ①
  return x + y                                     ②

z = suma(5, 2)                                     ③
```

① En este caso, se define una función llamada **suma** que recibe dos parámetros **x** y **y**.
② La función retorna la suma de **x** y **y**.
③ En este caso, se llama a la función **suma** con los argumentos **5** y **2**.

21 Ejemplo

El programa **Calculadora de propinas** es un ejemplo práctico que permite calcular la propina a dejar en un restaurante.

El funcionamiento del programa es el siguiente:

1. El usuario ingresa el monto total de la cuenta del restaurante.
2. Luego, el usuario selecciona el porcentaje de propina que desea dejar. Por ejemplo, puede elegir un 10%, 15% o 20%.
3. El programa calcula la cantidad de propina a partir del monto total de la cuenta y el porcentaje seleccionado.
4. Finalmente, el programa muestra al usuario el monto total de la cuenta, la cantidad de propina a dejar y el monto total a pagar (suma del monto total de la cuenta y la propina).

Este programa es útil para aquellos que deseen calcular rápidamente la cantidad de propina a dejar en un restaurante, sin tener que hacer los cálculos manualmente. Además, puede ser una buena práctica para familiarizarse con el uso de variables y el control de flujo en la programación.

Ver Código

```
# Ejemplo práctico: Calculadora de propinas

def calcular_propina(total, porcentaje):                                  ①
    propina = total * (porcentaje / 100)                                  ②
    return propina                                                        ③

def calcular_total_con_propina(total, porcentaje):
    propina = calcular_propina(total, porcentaje)
    total_con_propina = total + propina
    return total_con_propina

def main():                                                               ④
    print("Bienvenido a la calculadora de propinas")                      ⑤
    total = float(input("Ingrese el total de la cuenta: "))               ⑥
    porcentaje = float(input("Ingrese el porcentaje de propina que desea dejar: ")) ⑦

    propina = calcular_propina(total, porcentaje)                         ⑧
    total_con_propina = calcular_total_con_propina(total, porcentaje)     ⑨

    print(f"La propina a dejar es: {propina}")                            ⑩
    print(f"El total con propina es: {total_con_propina}")                ⑪
```

```
if __name__ == "__main__":                                    (12)
    main()                                                    (13)
```

(1) En este caso, se define una función llamada **calcular_propina** que recibe dos parámetros **total** y **porcentaje**.
(2) La función calcula la propina a partir del total y el porcentaje.
(3) La función retorna la propina.
(4) En este caso, se define una función llamada **main**.
(5) Se imprime un mensaje de bienvenida.
(6) Se solicita al usuario que ingrese el total de la cuenta.
(7) Se solicita al usuario que ingrese el porcentaje de propina que desea dejar.
(8) Se calcula la propina a partir del total y el porcentaje.
(9) Se calcula el total con propina.
(10) Se imprime la propina a dejar.
(11) Se imprime el total con propina.

22 Asignación

A continuación se sugiere realizar los siguientes ejercicios para practicar la sintaxis y estructura de Python.

Ejercicios Python Básico

22.1 Objetivo

El objetivo de este repositorio es proporcionar una serie de ejercicios de Python básico para que los principiantes en Python puedan practicar y adquirir experiencia en la sintaxis y estructura de Python.

22.2 ¿Qué debes hacer?

Debes Completar cada uno de los ejercicios propuetos a continuación cada uno en su respectivo archivo, el objetivo es adquirir práctica en la sintaxis y estructura de Python. Ejercicios

- **Imprimir Nombre:** Un programa simple que imprime tu nombre en la pantalla.
- **Suma de los Números del 1 al 10:** Un programa que calcula la suma de los números del 1 al 10.
- **Datos Personales:** Un programa que almacena tu edad, nombre y estatura en variables y las imprime en pantalla.
- **Par o Impar:** Un programa que determina si un número ingresado por el usuario es par o impar.
- **Área de un Círculo:** Una función que calcula el área de un círculo dado su radio.
- **Suma de Dos Números:** Una función que recibe dos números como argumentos y devuelve su suma.
- **Área de un Círculo con Parámetro:** Modificación de la función de área de un círculo para recibir el radio como parámetro.
- **Conversión de Temperatura:** Un programa que convierte grados Celsius a Fahrenheit y viceversa.

22.3 Pruebas

Cada ejercicio tiene su archivo de prueba en el que se utilizan las aserciones de pytest para verificar su correcto funcionamiento. Si por ejemplo quiero aplicar el test del primer ejercicio debo completar el primer ejercicio y comentar los demás, luego ejecutar el comando

pytest test_1.py para verificar que el programa funciona correctamente, luego continuar con cada uno de ellos e ir aplicando los test, hasta que al final todos los test pasen y completar la tarea

22.4 Ejecución

Para ejecutar cada programa, simplemente ejecute el archivo **programa.py**. Los archivos de prueba se pueden ejecutar con el comando **pytest**.

Part III

Unidad 3: Python Intermedio

23 Listas

Las listas son un tipo de dato que nos permite almacenar diferentes valores, en una sola variable.

- Las listas **son mutables**, es decir, podemos modificar su contenido.

Ejemplo:

```
mi_lista = [1, 2, 3, 4, 5]
```

Ejercicio:

Crear una lista con los números del 1 al 10, y mostrarla en pantalla.

Solución

```
  mi_lista = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
  print(mi_lista)
```

24 Tuplas

Las tuplas son un tipo de dato que nos permite almacenar diferentes valores, en una sola variable.

- Las tuplas **son inmutables**, es decir, no podemos modificar su contenido.

Ejemplo:

```
mi_tupla = (1, 2, 3, 4, 5)
```

Ejercicio:

Crear una tupla con los números del 1 al 10, y mostrarla en pantalla.

Solución

```
  mi_tupla = (1, 2, 3, 4, 5, 6, 7, 8, 9, 10)
  print(mi_tupla)
```

25 Manipulación de Listas y Tuplas

Para modificar una **lista** se puede realizar diferentes operaciones, como agregar, eliminar, modificar, y acceder a los elementos de la lista o tupla.

Ejemplo:

```
mi_lista = [1, 2, 3, 4, 5]          ①
mi_lista.append(6)                  ②
mi_lista.remove(3)                  ③
mi_lista[0] = 10                    ④
print(mi_lista)                     ⑤
```

① Creamos una lista con los números del 1 al 5.
② Agregamos el número 6 al final de la lista.
③ Eliminamos el número 3 de la lista.
④ Modificamos el primer elemento de la lista por el número 10.
⑤ Mostramos la lista en pantalla.

Ejercicio:

Crear una lista con los números del 1 al 10, y mostrarla en pantalla. Luego, agregar el número 11 al final de la lista, y mostrarla en pantalla. Finalmente, eliminar el número 5 de la lista, y mostrarla en pantalla.

Solución

```
mi_lista = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
print(mi_lista)
mi_lista.append(11)
print(mi_lista)
mi_lista.remove(5)
print(mi_lista)
```

> **Tip**
>
> La caracteristica principal de las tuplas es que son inmutables, por lo que no se pueden modificar.

26 Funciones ingegradas para Listas y Tuplas

Python cuenta con funciones integradas que nos permiten realizar diferentes operaciones con listas y tuplas.

Ejemplo:

```
mi_lista = [1, 2, 3, 4, 5]
mi_tupla = (1, 2, 3, 4, 5)

print(len(mi_lista))                                    ①
print(len(mi_tupla))                                    ②
print(max(mi_lista))                                    ③
print(max(mi_tupla))                                    ④
print(min(mi_lista))                                    ⑤
print(min(mi_tupla))                                    ⑥
print(sum(mi_lista))                                    ⑦
```

① Mostramos la longitud de la lista.
② Mostramos la longitud de la tupla.
③ Mostramos el número mayor de la lista.
④ Mostramos el número mayor de la tupla.
⑤ Mostramos el número menor de la lista.
⑥ Mostramos el número menor de la tupla.
⑦ Mostramos la suma de los elementos de la lista.

Ejercicio:

Crear una lista con los números del 1 al 10, y mostrar la longitud de la lista. Luego, mostrar el número mayor y menor de la lista, y finalmente mostrar la suma de los elementos de la lista.

Solución

```
mi_lista = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
print(len(mi_lista))
print(max(mi_lista))
print(min(mi_lista))
print(sum(mi_lista))
```

27 Listas Anidadas

Las listas anidadas son listas que contienen otras listas.

Ejemplo:

```
mi_lista = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]
print(mi_lista)
```

Ejercicio:

Crear una lista anidada con los números del 1 al 9, y mostrarla en pantalla.

Solución

```
  mi_lista = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]
  print(mi_lista)
```

28 Listas y Tuplas como Argumentos de Funciones

Las listas y tuplas pueden ser utilizadas como argumentos de funciones.

Ejemplo:

```
def mostrar_lista(lista):
    for elemento in lista:
        print(elemento)

mi_lista = [1, 2, 3, 4, 5]
mostrar_lista(mi_lista)
```

Ejercicio:

Crear una función que reciba una lista como argumento, y muestre en pantalla los elementos de la lista.

Solución

```
def mostrar_lista(lista):
    for elemento in lista:
        print(elemento)

mi_lista = [1, 2, 3, 4, 5]
mostrar_lista(mi_lista)
```

29 Listas y Tuplas como Retorno de Funciones

Las listas y tuplas pueden ser utilizadas como retorno de funciones.

Ejemplo:

```
def crear_lista():
    return [1, 2, 3, 4, 5]

mi_lista = crear_lista()
print(mi_lista)
```

Ejercicio:

Crear una función que retorne una lista con los números del 1 al 10, y mostrarla en pantalla.

Solución

```
def crear_lista():
    return [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

mi_lista = crear_lista()
print(mi_lista)
```

30 Asignación

https://classroom.github.com/a/207M40z1

Este repositorio contiene una asignación para el curso de programación en Python. La asignación es sobre la manipulación de listas y tuplas en Python.

30.1 Descripción de la Asignación

El archivo ejercicio.py contiene un script que pide al usuario que ingrese una lista de compras. El usuario debe ingresar los elementos de la lista separados por comas. El script luego imprime la lista de compras.

Además, el script contiene una función llamada convertir_lista_a_tupla() que está destinada a convertir la lista de compras en una tupla. Esta función aún no está completa.

30.2 Tarea Pendiente:

- Completar la función convertir_lista_a_tupla() para que convierta la lista de compras en una tupla.

30.3 Cómo Ejecutar el Código

Para ejecutar el test puedes utilizar el siguiente comando en tu terminal:

```
pytest -s
```

Pytest es una biblioteca que facilita la escritura de pruebas en Python. El parámetro -s se utiliza para mostrar la salida de la prueba en la terminal.

30.4 Ejemplo de salida:

```
$ pytest -s
================================================= test session starts ====================
platform linux -- Python 3.8.10, pytest-6.2.4, pluggy-0.13.1
rootdir: /home/user/Documentos/Python/Asignacion_Lista_Compras
collected 1 item

test_ejercicio.py Lista de Compras: [manzanas, peras, plátanos, uvas]

================================================== 1 passed in 0.01s =====================
```

> **Tip**
>
> Se sugiere que practique la sección **Ejercicios Python - Nivel 3** para reforzar los conocimientos adquiridos.

31 Diccionarios

Los diccionarios son una estructura de datos que nos permite almacenar información en pares clave-valor. La clave es un identificador único que nos permite acceder al valor asociado a ella. Los diccionarios son mutables, es decir, podemos modificar su contenido.

Para crear un diccionario, utilizamos llaves **{}** y separamos cada par clave-valor con dos puntos **:**. Las claves y los valores pueden ser de cualquier tipo de dato.

Ejemplo:

```
mi_diccionario = {
    "nombre": "Juan",
    "edad": 25,
    "ciudad": "Bogotá"
}
```

Ejercicio:

- Crear un diccionario con las siguientes claves y valores:
  - "nombre": "Juan"
  - "edad": 25
  - "ciudad": "Bogotá"

Respuesta

```
mi_diccionario = {
    "nombre": "Juan",
    "edad": 25,
    "ciudad": "Bogotá"
}
```

Para acceder a un valor de un diccionario, utilizamos la clave correspondiente entre corchetes [].

Ejemplo:

```
print(mi_diccionario["nombre"])  # Juan

nombre = mi_diccionario["nombre"]
print(nombre)  # Juan
```

Ejercicio:

- Imprimir el valor de la clave "edad" del diccionario **mi_diccionario**.
- Guardar el valor de la clave "ciudad" del diccionario **mi_diccionario** en una variable llamada **ciudad**.
- Imprimir la variable **ciudad**.
- Imprimir el valor de la clave "nombre" del diccionario **mi_diccionario**.
- Guardar el valor de la clave "edad" del diccionario **mi_diccionario** en una variable llamada **edad**.

Respuesta

```
print(mi_diccionario["edad"])  # 25

ciudad = mi_diccionario["ciudad"]

print(ciudad)  # Bogotá

print(mi_diccionario["nombre"])  # Juan

edad = mi_diccionario["edad"]

print(edad)  # 25
```

Para modificar un valor de un diccionario, utilizamos la clave correspondiente entre corchetes [] y le asignamos el nuevo valor.

Ejemplo:

```
mi_diccionario["edad"] = 30
print(mi_diccionario)  # {'nombre': 'Juan', 'edad': 30, 'ciudad': 'Bogotá'}
```

Ejercicio:

- Modificar el valor de la clave "edad" del diccionario **mi_diccionario** a 30.

Respuesta

```
mi_diccionario["edad"] = 30
print(mi_diccionario)  # {'nombre': 'Juan', 'edad': 30, 'ciudad': 'Bogotá'}
```

Para agregar un nuevo par clave-valor a un diccionario, utilizamos la clave correspondiente entre corchetes [] y le asignamos el nuevo valor.

Ejemplo:

```
mi_diccionario["profesion"] = "Ingeniero"
print(mi_diccionario)  # {'nombre': 'Juan', 'edad': 30, 'ciudad': 'Bogotá', 'profesion':
```

Ejercicio:

- Agregar un nuevo par clave-valor al diccionario **mi_diccionario** con la clave "profesion" y el valor "Ingeniero".
- Imprimir el diccionario **mi_diccionario**.

Respuesta

```
mi_diccionario["profesion"] = "Ingeniero"
print(mi_diccionario)  # {'nombre': 'Juan', 'edad': 30, 'ciudad': 'Bogotá', 'profesion':
```

Respuesta

```
mi_diccionario["profesion"] = "Ingeniero"
print(mi_diccionario)  # {'nombre': 'Juan', 'edad': 30, 'ciudad': 'Bogotá', 'profesion':
```

Para eliminar un par clave-valor de un diccionario, utilizamos la palabra reservada **del** seguida de la clave correspondiente entre corchetes [].

Ejemplo:

```
del mi_diccionario["edad"]
print(mi_diccionario)  # {'nombre': 'Juan', 'ciudad': 'Bogotá', 'profesion': 'Ingeniero'}
```

Ejercicio:

- Eliminar la clave "edad" del diccionario **mi_diccionario**.

Respuesta

```
del mi_diccionario["edad"]
print(mi_diccionario)  # {'nombre': 'Juan', 'ciudad': 'Bogotá', 'profesion': 'Ingeniero'}
```

Para recorrer un diccionario, podemos utilizar un ciclo **for** que recorra las claves del diccionario.

Ejemplo:

```
for clave in mi_diccionario:
    print(clave)

# nombre
# ciudad
# profesion
```

Ejercicio:

- Recorrer el diccionario **mi_diccionario** e imprimir las claves.

Respuesta

```
for clave in mi_diccionario:
    print(clave)

# nombre
# ciudad
# profesion
```

Para recorrer un diccionario y obtener tanto las claves como los valores, podemos utilizar el método **items()**.

Ejemplo:

```
for clave, valor in mi_diccionario.items():
    print(clave, valor)

# nombre Juan
# ciudad Bogotá
# profesion Ingeniero
```

Ejercicio:

- Recorrer el diccionario **mi_diccionario** e imprimir las claves y los valores.
- Imprimir el valor de la clave "nombre" del diccionario **mi_diccionario**.
- Imprimir el valor de la clave "ciudad" del diccionario **mi_diccionario**.
- Imprimir el valor de la clave "profesion" del diccionario **mi_diccionario**.

Respuesta

```
for clave, valor in mi_diccionario.items():
    print(clave, valor)

# nombre Juan
# ciudad Bogotá
# profesion Ingeniero

print(mi_diccionario["nombre"])  # Juan
print(mi_diccionario["ciudad"])  # Bogotá
print(mi_diccionario["profesion"])  # Ingeniero
```

Para verificar si una clave se encuentra en un diccionario, podemos utilizar la palabra reservada **in**.

Ejemplo:

```
if "nombre" in mi_diccionario:
    print("La clave 'nombre' se encuentra en el diccionario")

if "apellido" not in mi_diccionario:
    print("La clave 'apellido' no se encuentra en el diccionario")
```

Ejercicio:

- Verificar si la clave “nombre” se encuentra en el diccionario **mi_diccionario**.
- Verificar si la clave “apellido” no se encuentra en el diccionario **mi_diccionario**.
- Verificar si la clave “ciudad” se encuentra en el diccionario **mi_diccionario**.
- Verificar si la clave “profesion” no se encuentra en el diccionario **mi_diccionario**.
- Verificar si la clave “edad” se encuentra en el diccionario **mi_diccionario**.
- Verificar si la clave “telefono” no se encuentra en el diccionario **mi_diccionario**.

Respuesta

```
if "nombre" in mi_diccionario:
    print("La clave 'nombre' se encuentra en el diccionario")

if "apellido" not in mi_diccionario:
    print("La clave 'apellido' no se encuentra en el diccionario")

if "ciudad" in mi_diccionario:
    print("La clave 'ciudad' se encuentra en el diccionario")

if "profesion" not in mi_diccionario:
    print("La clave 'profesion' no se encuentra en el diccionario")

if "edad" in mi_diccionario:
    print("La clave 'edad' se encuentra en el diccionario")
else:
    print("La clave 'edad' no se encuentra en el diccionario")

if "telefono" not in mi_diccionario:
    print("La clave 'telefono' no se encuentra en el diccionario")
else:
    print("La clave 'telefono' se encuentra en el diccionario")
```

32 Conjuntos

Los conjuntos son una estructura de datos que nos permite almacenar elementos únicos. Los conjuntos son mutables, es decir, podemos modificar su contenido.

Para crear un conjunto, utilizamos llaves **{}** y separamos cada elemento con comas **,**.

Ejemplo:

```
mi_conjunto = {1, 2, 3, 4, 5}
```

Ejercicio:

- Crear un conjunto con los siguientes elementos: 1, 2, 3, 4, 5, 6

Respuesta

```
mi_conjunto = {1, 2, 3, 4, 5, 6}
```

Para agregar un elemento a un conjunto, utilizamos el método **add()**.

Ejemplo:

```
mi_conjunto.add(7)
print(mi_conjunto)  # {1, 2, 3, 4, 5, 6, 7}
```

Ejercicio:

- Agregar el número 8 al conjunto **mi_conjunto**.
- Imprimir el conjunto **mi_conjunto**.
- Agregar el número 9 al conjunto **mi_conjunto**.
- Imprimir el conjunto **mi_conjunto**.
- Agregar el número 10 al conjunto **mi_conjunto**.
- Imprimir el conjunto **mi_conjunto**.
- Agregar el número 11 al conjunto **mi_conjunto**.
- Imprimir el conjunto **mi_conjunto**.
- Agregar el número 12 al conjunto **mi_conjunto**.
- Imprimir el conjunto **mi_conjunto**.

Respuesta

```
mi_conjunto.add(8)
print(mi_conjunto)  # {1, 2, 3, 4, 5, 6, 7, 8}

mi_conjunto.add(9)
print(mi_conjunto)  # {1, 2, 3, 4, 5, 6, 7, 8, 9}

mi_conjunto.add(10)
print(mi_conjunto)  # {1, 2, 3, 4, 5, 6, 7, 8, 9, 10}

mi_conjunto.add(11)
print(mi_conjunto)  # {1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11}

mi_conjunto.add(12)
print(mi_conjunto)  # {1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12}
```

Para eliminar un elemento de un conjunto, utilizamos el método **remove()**.

Ejemplo:

```
mi_conjunto.remove(7)
print(mi_conjunto)  # {1, 2, 3, 4, 5, 6}
```

Ejercicio:

- Eliminar el número 12 del conjunto **mi_conjunto**.
- Imprimir el conjunto **mi_conjunto**.

Respuesta

```
mi_conjunto.remove(12)
print(mi_conjunto)  # {1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11}
```

Para recorrer un conjunto, podemos utilizar un ciclo **for**.

Ejemplo:

```
for elemento in mi_conjunto:
    print(elemento)

# 1
# 2
# 3
# 4
# 5
# 6
# 7
# 8
# 9
```

```
# 10
# 11
```

Ejercicio:

- Recorrer el conjunto mi_conjunto e imprimir los elementos.

Respuesta

```
for elemento in mi_conjunto:
    print(elemento)
```

Para verificar si un elemento se encuentra en un conjunto, podemos utilizar la palabra reservada **in**.

Ejemplo:

```
if 1 in mi_conjunto:
    print("El número 1 se encuentra en el conjunto")

if 7 not in mi_conjunto:
    print("El número 7 no se encuentra en el conjunto")

if 10 in mi_conjunto:
    print("El número 10 se encuentra en el conjunto")

if 15 not in mi_conjunto:
    print("El número 15 no se encuentra en el conjunto")

if 20 in mi_conjunto:
    print("El número 20 se encuentra en el conjunto")

if 25 not in mi_conjunto:
    print("El número 25 no se encuentra en el conjunto")
```

Ejercicio:

- Verificar si el número 30 se encuentra en el conjunto mi_conjunto.
- Verificar si el número 35 no se encuentra en el conjunto mi_conjunto.
- Verificar si el número 40 se encuentra en el conjunto mi_conjunto.

Respuesta

```
if 30 in mi_conjunto:
    print("El número 30 se encuentra en el conjunto")
else:
    print("El número 30 no se encuentra en el conjunto")
```

```
if 35 not in mi_conjunto:
    print("El número 35 no se encuentra en el conjunto")
else:
    print("El número 35 se encuentra en el conjunto")

if 40 in mi_conjunto:
    print("El número 40 se encuentra en el conjunto")
else:
    print("El número 40 no se encuentra en el conjunto")
```

33 Operaciones con Diccionarios y Conjuntos

Para realizar operaciones con diccionarios y conjuntos, podemos utilizar los métodos y funciones que nos proporciona Python.

Ejercicio:

- Crear un diccionario con las siguientes claves y valores:
  - "nombre": "Diego"
  - "edad": 36
  - "ciudad": "Quito"
  - "profesion": "Ingeniero"
  - "telefono": "1234567890"
  - "email": "dsaavedra88@gmail.com"
  - "direccion": "Calle 123 # 45-67"
  - "pais": "Ecuador"
  - Crear un conjunto con los siguientes elementos:
    - 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11

Respuesta

```
mi_diccionario = {
    "nombre": "Diego",
    "edad": 36,
    "ciudad": "Quito",
    "profesion": "Ingeniero",
    "telefono": "1234567890",
    "email": "
    "direccion": "Calle 123 # 45-67",
    "pais": "Ecuador"
}

mi_conjunto = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11}
```

Otra operación que podemos realizar con diccionarios y conjuntos es la unión. Para unir dos diccionarios, utilizamos el método **update()**. Para unir dos conjuntos, utilizamos el método **union()**.

Ejercicio:

- Crear un diccionario con las siguientes claves y valores:

- “apellido”: “Saavedra”
- “genero”: “Masculino”
- “estado_civil”: “Soltero”
- “hijos”: 0
- “mascotas”: 1
- Crear un conjunto con los siguientes elementos:
  - 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22

Respuesta

```
mi_diccionario.update({
    "apellido": "Saavedra",
    "genero": "Masculino",
    "estado_civil": "Soltero",
    "hijos": 0,
    "mascotas": 1
})

mi_conjunto.union({12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22})
```

34 Asignación

https://classroom.github.com/a/93tJXiLB

34.1 Descripción

Esta asignación consiste en corregir y ejecutar un test unitario para un diccionario de frutas. Instrucciones

1. Abre el archivo **ejercicio.py**.
2. Corrige el diccionario frutas para que tenga las siguientes parejas **clave-valor**:

```
"manzana" - "roja"
"banana" - "amarilla"
"pera" - "verde"
"naranja" - "naranja"
```

3. Guarda y cierra el archivo **ejercicio.py**.
4. Ejecuta el test unitario test_ejercicio en tu terminal con el comando:

```
python -m unittest test_ejercicio.py
```

5. Si el **test unitario** se ejecuta sin errores, habrás **completado la asignación**.
6. Si el **test unitario** arroja errores, **corrige** el **diccionario frutas** en **ejercicio.py** y **vuelve** a **ejecutar el test unitario**.
7. Repite los pasos 4 a 6 hasta que el test unitario se ejecute sin errores.

34.2 Criterios de Evaluación

- El diccionario frutas en **ejercicio.py** tiene las parejas clave-valor correctas.
- El test unitario test_frutas en **test_ejercicio.py** se ejecuta sin errores.

:::{.callout-tip} Se sugiere revisar la sección de **Ejercicios Python - Nivel 4** para poder reforzar los conocimientos necesarios para completar esta sección.

Part IV

Unidad 4: Python Avanzado

35 Programación Orientada a Objetos

La programación orientada a objetos (POO) es un paradigma de programación que usa "objetos" para diseñar aplicaciones y programas informáticos.

Un objeto es una entidad que agrupa un estado (atributos) y un comportamiento (métodos). Por ejemplo, un objeto podría representar a una persona con atributos como nombre, edad, género, etc., y comportamientos como caminar, hablar, respirar, etc.

La programación orientada a objetos se basa en varios conceptos fundamentales:

- Clases
- Objetos
- Atributos
- Métodos
- Herencia
- Polimorfismo
- Abstracción
- Encapsulamiento

Estos conceptos se explican a continuación.

- Clases: Una clase es un plano para los objetos. Es un diseño que define un objeto. Una clase puede contener atributos y métodos. Por ejemplo, la clase "Perro" puede tener atributos como raza, color, edad, y métodos como ladrar, comer, dormir, etc.

Ejemlo:

```
class Perro:
    def __init__(self, raza, color, edad):
        self.raza = raza
        self.color = color
        self.edad = edad

    def ladrar(self):
        print("Guau! Guau!")
```

Ejericio:

Crear una clase llamada "Persona" con los atributos "nombre", "edad" y "género". La clase debe tener un método llamado "hablar" que imprime "Hola, mi nombre es [nombre]".

Ver respuesta

```python
class Persona:
    def __init__(self, nombre, edad, genero):
        self.nombre = nombre
        self.edad = edad
        self.genero = genero

    def hablar(self):
        print(f"Hola, mi nombre es {self.nombre}")
```

- Objetos: Un objeto es una instancia de una clase. Cuando se crea un objeto, se reserva memoria para el objeto y se inicializa. Un objeto puede tener atributos y métodos. Por ejemplo, si "Perro" es una clase, entonces "Perro Labrador" es un objeto de la clase "Perro".

Ejemplo:

```python
perro1 = Perro("Labrador", "Dorado", 5)
perro2 = Perro("Bulldog", "Blanco", 3)
```

Ejercicio:

Crear un objeto de la clase "Persona" con nombre "Juan", edad 25 y género "Masculino".

Ver respuesta

```python
juan = Persona("Juan", 25, "Masculino")
```

- Atributos: Los atributos son variables que pertenecen a un objeto. Los atributos definen las características de un objeto. Por ejemplo, "raza", "color" y "edad" son atributos de la clase "Perro".

Ejemplo:

```python
print(perro1.raza)  # Labrador
print(perro2.color)  # Blanco
```

Ejericio:

Imprimir el nombre de la persona "Juan".

Ver respuesta

```python
print(juan.nombre)  # Juan
```

- Métodos: Los métodos son funciones que pertenecen a un objeto. Los métodos definen el comportamiento de un objeto. Por ejemplo, "ladrar" y "comer" son métodos de la clase "Perro".

Ejemplo:

```
perro1.ladrar()  # Guau! Guau!
```

Ejercicio:

Llamar al método “hablar” del objeto “juan”.

Ver respuesta

```
juan.hablar()  # Hola, mi nombre es Juan
```

- Herencia: La herencia es un mecanismo en el que una clase adquiere las propiedades y el comportamiento de otra clase. La clase que hereda se llama clase derivada o subclase, y la clase de la que se hereda se llama clase base o superclase. La herencia permite la reutilización del código y la creación de una jerarquía de clases.

Ejemplo:

```
class Animal:
    def __init__(self, nombre):
        self.nombre = nombre

    def comer(self):
        print("Comiendo...")

class Perro(Animal):
    def ladrar(self):
        print("Guau! Guau!")

perro = Perro("Firulais")
perro.comer()  # Comiendo...
perro.ladrar()  # Guau! Guau!
```

Ejercicio:

Crear una clase “Estudiante” que herede de la clase “Persona”. La clase “Estudiante” debe tener un atributo adicional llamado “carrera” y un método llamado “estudiar” que imprime “Estudiando…”.

Ver respuesta

```
class Estudiante(Persona):
    def __init__(self, nombre, edad, genero, carrera):
        super().__init__(nombre, edad, genero)
        self.carrera = carrera

    def estudiar(self):
        print("Estudiando...")
```

- Polimorfismo: El polimorfismo es la capacidad de un objeto para tomar muchas formas. En Python, el polimorfismo se logra mediante el uso de métodos con el mismo nombre en diferentes clases. El polimorfismo permite que un objeto se comporte de diferentes maneras según el contexto.

Ejemplo:

```
class Animal:
    def hablar(self):
        pass

class Perro(Animal):
    def hablar(self):
        print("Guau! Guau!")

class Gato(Animal):
    def hablar(self):
        print("Miau! Miau!")

animales = [Perro(), Gato()]

for animal in animales:
    animal.hablar()
```

Ejercicio:

Crear una clase "Profesor" con un método "enseñar" que imprime "Enseñando…". Luego, crear una lista de objetos que contenga un objeto de la clase "Estudiante" y un objeto de la clase "Profesor". Llamar al método "hablar" de cada objeto en la lista.

Ver respuesta

```
class Profesor:
    def enseñar(self):
        print("Enseñando...")

profesor = Profesor()
estudiante = Estudiante("Ana", 20, "Femenino", "Ingeniería")

personas = [profesor, estudiante]

for persona in personas:
    persona.hablar()
```

- Abstracción: La abstracción es el proceso de ocultar los detalles de implementación y mostrar solo la funcionalidad al usuario. En Python, la abstracción se logra mediante el uso de clases y métodos. Los usuarios pueden interactuar con los objetos sin conocer los detalles internos de cómo funcionan.

Ejemplo:

```
class Coche:
    def __init__(self, marca, modelo):
        self.marca = marca
        self.modelo = modelo

    def arrancar(self):
        print("Arrancando...")
```

Ejercicio:

Crear una clase "Círculo" con un atributo "radio" y un método "calcular_area" que imprime el área del círculo. Luego, crear un objeto de la clase "Círculo" con radio 5 y llamar al método "calcular_area".

Ver respuesta

```
import math

class Circulo:
    def __init__(self, radio):
        self.radio = radio

    def calcular_area(self):
        area = math.pi * self.radio ** 2
        print(f"El área del círculo es {area}")

circulo = Circulo(5)
circulo.calcular_area()
```

- Encapsulamiento: El encapsulamiento es el proceso de ocultar los detalles de implementación de un objeto y restringir el acceso a ciertos componentes. En Python, el encapsulamiento se logra mediante el uso de métodos y atributos privados. Los métodos y atributos privados no se pueden acceder directamente desde fuera de la clase.

Ejemplo:

```
class Coche:
    def __init__(self, marca, modelo):
        self.marca = marca
        self.modelo = modelo
        self.__velocidad = 0

    def acelerar(self):
        self.__velocidad += 10

    def frenar(self):
```

```
        self.__velocidad -= 10

    def get_velocidad(self):
        return self.__velocidad

coche = Coche("Toyota", "Corolla")
coche.acelerar()
print(coche.get_velocidad())  # 10
coche.frenar()

# Intentar acceder al atributo privado directamente
# print(coche.__velocidad)  # Error
```

Ejercicio:

Modificar la clase "Círculo" para que el atributo "radio" sea privado. Agregar métodos "get_radio" y "set_radio" para obtener y establecer el valor del radio. Luego, crear un objeto de la clase "Círculo" con radio 5, cambiar el radio a 10 y llamar al método "calcular_area".

Ver respuesta

```
import math

class Circulo:
    def __init__(self, radio):
        self.__radio = radio

    def calcular_area(self):
        area = math.pi * self.__radio ** 2
        print(f"El área del círculo es {area}")

    def get_radio(self):
        return self.__radio

    def set_radio(self, radio):
        self.__radio = radio

circulo = Circulo(5)
circulo.set_radio(10)
circulo.calcular_area()
```

La programación orientada a objetos es un concepto fundamental en Python y en muchos otros lenguajes de programación. Al comprender los conceptos de clases, objetos, atributos, métodos, herencia, polimorfismo, abstracción y encapsulamiento, puedes diseñar y desarrollar aplicaciones y programas más eficientes y reutilizables.

35.1 Asignación

https://classroom.github.com/a/LVvqQCln

En esta asignación, aprenderás sobre los conceptos básicos de **Programación Orientada a Objetos (POO)** mediante la implementación de clases en Python.

35.2 Instrucciones

1. Lee cuidadosamente el contenido de este repositorio.
2. Implementa las clases solicitadas en el archivo main.py.
3. Realiza los commits y push necesarios para subir tus cambios a este repositorio.
4. Verifica que tus cambios pasen todas las pruebas.

35.3 Contenido del Repositorio

- **main.py**: Archivo principal donde implementarás tus clases.
- **test_main.py**: Archivo de pruebas unitarias.
- **README.md**: Este archivo con las instrucciones de la asignación.

35.4 Cómo Ejecutar las Pruebas

- Asegúrate de tener Python instalado en tu sistema.
- Abre una terminal y navega hasta la ubicación de este repositorio.
- Ejecuta **python -m unittest** para correr las pruebas.

36 Módulos

Un módulo es un archivo que contiene definiciones y declaraciones de Python. El archivo debe tener una extensión .py. Las definiciones de un módulo pueden ser importadas a otros módulos o al programa principal.

Ejemplo:

```
# modulo.py
def saludar():
    print("Hola, bienvenido a Python")
```

```
# programa.py
import modulo

modulo.saludar()
```

Ejercicio:

Crear un módulo llamado operaciones.py que contenga las siguientes funciones:

- suma(a, b): Retorna la suma de a y b
- resta(a, b): Retorna la resta de a y b
- multiplicacion(a, b): Retorna la multiplicación de a y b
- division(a, b): Retorna la división de a y b

Ver respuesta

operaciones.py

```
def suma(a, b):
    return a + b

def resta(a, b):
    return a - b

def multiplicacion(a, b):
    return a * b

def division(a, b):
    return a / b
```

programa.py

```python
import operaciones

a = 10
b = 5

print(operaciones.suma(a, b))
print(operaciones.resta(a, b))
print(operaciones.multiplicacion(a, b))
print(operaciones.division(a, b))
```

37 Paquetes

Un paquete es un conjunto de módulos organizados en un directorio. Un paquete debe contener un archivo llamado **init**.py. Este archivo puede estar vacío o contener código de inicialización del paquete.

Ejemplo:

```
paquete/
    __init__.py
    modulo1.py
    modulo2.py
```

```python
# modulo1.py
def saludar():
    print("Hola, bienvenido a Python")
```

```python
# modulo2.py
def despedir():
    print("Adiós, hasta luego")
```

```python
# programa.py
from paquete import modulo1, modulo2

modulo1.saludar()
modulo2.despedir()
```

Ejercicio:

Crear un paquete llamado operaciones que contenga los módulos suma.py, resta.py, multiplicacion.py y division.py. Cada módulo debe contener una función que realice la operación correspondiente.

Ver respuesta

operaciones/ **init**.py suma.py resta.py multiplicacion.py division.py

suma.py

```python
def suma(a, b):
    return a + b
```

resta.py

```
def resta(a, b):
    return a - b
```

multiplicacion.py

```
def multiplicacion(a, b):
    return a * b
```

division.py

```
def division(a, b):
    return a / b
```

programa.py

```
from operaciones import suma, resta, multiplicacion, division

a = 10
b = 5

print(suma.suma(a, b))
print(resta.resta(a, b))
print(multiplicacion.multiplicacion(a, b))
print(division.division(a, b))
```

38 Creación y Uso de Módulos

> **Tip**
>
> La diferencia principal entre paquetes y módulos es que los paquetes son directorios que contienen módulos y un archivo **init**.py, mientras que los módulos son archivos individuales que contienen funciones y variables.

38.1 Creación de Módulos

Para crear un módulo, simplemente se crea un archivo con extensión .py y se definen las funciones y variables que se desean exportar.

Ejemplo:

```
# modulo.py
def saludar():
    print("Hola, bienvenido a Python")
```

38.2 Uso de Módulos

Para usar un módulo, se utiliza la palabra reservada import seguida del nombre del módulo.

Ejemplo:

```
# programa.py
import modulo

modulo.saludar()
```

38.3 Importar Funciones Específicas

También es posible importar funciones específicas de un módulo.

Ejemplo:

```
# programa.py
from modulo import saludar

saludar()
```

38.4 Importar con Alias

Es posible importar un módulo o función con un alias.

Ejemplo:

```
# programa.py

import modulo as m

m.saludar()
```

38.5 Importar Todas las Funciones

También es posible importar todas las funciones de un módulo.

Ejemplo:

```
# programa.py

from modulo import *

saludar()
```

39 Creación y Uso de Paquetes

39.1 Creación de Paquetes

Para crear un paquete, se crea un directorio con el nombre del paquete y se agregan los módulos necesarios. Además, se debe crear un archivo **init**.py en el directorio del paquete.

Ejemplo:

```
paquete/
    __init__.py
    modulo1.py
    modulo2.py
```

39.2 Uso de Paquetes

Para usar un paquete, se utiliza la palabra reservada import seguida del nombre del paquete y el nombre del módulo.

Ejemplo:

```python
# programa.py

from paquete import modulo1, modulo2

modulo1.saludar()
modulo2.despedir()
```

39.3 Importar con Alias

Es posible importar un paquete o módulo con un alias.

Ejemplo:

```python
from paquete import modulo1 as m1, modulo2 as m2

m1.saludar()
m2.despedir()
```

39.4 Importar Todas las Funciones

También es posible importar todas las funciones de un módulo.

Ejemplo:

```
# programa.py

from paquete.modulo1 import *

saludar()
```

40 Asignación Calculadora Pythonica

https://classroom.github.com/a/Kyffdibl

En esta asignación, aprenderás sobre la creación y uso de módulos y paquetes en Python.

40.1 Instrucciones

1. Lee cuidadosamente el contenido de este repositorio.
2. Implementa las funciones solicitadas en el archivo **operaciones.py**.
3. Completa el programa principal en el archivo **programa.py**.
4. Realiza los commits y push necesarios para subir tus cambios a este repositorio.
5. Verifica que tus cambios funcionen correctamente.

40.2 Contenido del Repositorio

- **operaciones.py**: Archivo de módulo que contiene las funciones para realizar operaciones matemáticas.
- **programa.py**: Archivo principal donde se utiliza el módulo operaciones.py.
- **test_operaciones.py**: Archivo de pruebas unitarias para verificar las funciones del módulo **operaciones.py**.
- **.gitignore**: Archivo que indica a Git qué archivos y directorios debe ignorar al rastrear los cambios en el repositorio.
- **requirements.txt**: Archivo que especifica las dependencias del proyecto.

40.3 Ejercicio

Crear un módulo llamado operaciones.py que contenga las siguientes funciones:

1. **suma(a, b):** Retorna la **suma** de **a** y **b**.
2. **resta(a, b):** Retorna la **resta** de **a** y **b**.
3. **multiplicacion(a, b):** Retorna la **multiplicación** de **a** y **b**.
4. **division(a, b):** Retorna la **división** de **a** y **b**. Si **b** es **cero**, **retorna** un **mensaje de error**.

40.4 Cómo Ejecutar el Programa

- Asegúrate de tener Python instalado en tu sistema.
- Abre una terminal y navega hasta la ubicación de este repositorio.
- Instala las dependencias ejecutando pip install -r requirements.txt.
- Ejecuta python programa.py para ver los resultados de las operaciones.

40.5 Cómo Ejecutar las Pruebas

- Asegúrate de tener Python instalado en tu sistema.
- Abre una terminal y navega hasta la ubicación de este repositorio.
- Ejecuta **python -m unittest** para correr las pruebas.

41 Sistema de Gestión de Inventarios

41.1 Asignación

https://classroom.github.com/a/S9EkZXRf

Aprenderás a desarrollar un proyecto de utilizando el lenguaje de programación Python.

Un sistema de gestión de inventarios es una herramienta que permite realizar un seguimiento y control de los productos o artículos almacenados en un negocio o empresa.

Aprenderás a utilizar diferentes conceptos y técnicas de programación para implementar las funcionalidades clave de este sistema.

Algunas de las funcionalidades que implementaremos incluyen:

Aprenderás a crear una estructura de datos para almacenar la información de los productos, como su nombre, descripción, precio, cantidad disponible, etc. También aprenderás a agregar nuevos productos al sistema.

Te enseñaré cómo implementar funciones de búsqueda y filtrado para encontrar productos específicos en base a diferentes criterios, como el nombre, la categoría o el precio.

Aprenderás a manejar las actualizaciones de inventario, como la compra o venta de productos. Implementaremos funciones que permitan aumentar o disminuir la cantidad disponible de un producto y mantener un registro de estas transacciones.

Te mostraré cómo generar informes sobre el estado del inventario, como la lista de productos disponibles, los productos más vendidos, los productos con bajo stock, etc. Utilizaremos técnicas de manipulación de datos y generación de informes para presentar esta información de manera clara y concisa.

Part V

Unidad 5: Django

42 Introducción a Django

Figure 42.1: Django Framework

Django es un framework web de alto nivel que fomenta el desarrollo rápido y limpio. Es un framework web de alto nivel que fomenta el desarrollo rápido y limpio. Diseñado por desarrolladores experimentados, Django se encarga de gran parte de la molestia del desarrollo web, por lo que puedes concentrarte en escribir tu aplicación sin necesidad de reinventar la rueda. Es gratuito y de código abierto, tiene una comunidad activa y amigable, y es utilizado por algunas de las mayores empresas del mundo.

42.1 Conceptos Importantes

> **Tip**
>
> Antes de iniciar con Django es necesario conocer el concepto de **Entornos Virtuales**.

42.1.1 Entornos Virtuales

Figure 42.2: Virtual Enviroment

Un entorno virtual es un **entorno de desarrollo aislado** que permite **instalar paquetes de Python sin afectar al sistema global**. Los entornos virtuales son útiles para gestionar las dependencias de un proyecto y para evitar conflictos entre diferentes versiones de los paquetes.

42.1.1.1 Crear un entorno virtual

Para crear un entorno virtual, se puede utilizar la herramienta **venv** de Python.

```
python -m venv env
```

Este comando creará un directorio llamado **env** en el directorio actual con el entorno virtual.

> **Tip**
>
> Tambien se puede utilizar virtualenv para crear entornos virtuales.

42.1.2 Modelo Template View (MTV)

Figure 42.3: Model View Controller

Figure 42.4: Model View Template

Django sigue el patrón de diseño Modelo Vista Template (MVT). Este patrón de diseño separa la lógica de la aplicación en tres componentes principales: Modelo, Vista y Template.

> **Tip**
>
> El archivo URLs.py es el encargado de mapear las URLs de la aplicación a las vistas correspondientes.

- **Modelo**: Es la representación de los datos de la aplicación y las reglas para manipular esos datos. Django utiliza un ORM (Object-Relational Mapping) para interactuar con la base de datos.
- **Vista**: Es la capa de presentación de la aplicación. Se encarga de mostrar los datos al usuario y de interpretar las acciones del usuario.
- **Template**: Es la capa de presentación de la aplicación. Define cómo se muestra la información al usuario. Django utiliza el motor de plantillas Jinja2 para renderizar los templates.

42.1.3 Formularios

Figure 42.5: Django Forms

Los formularios son una parte importante de cualquier aplicación web. Django proporciona una forma sencilla de crear y procesar formularios en las vistas.

42.1.4 Administrador de Django

Figure 42.6: Django Admin

El administrador de Django es una interfaz de administración que permite gestionar los datos de la aplicación de forma sencilla. Django genera automáticamente una interfaz de administración basada en los modelos de la aplicación.

42.1.5 Middleware

El middleware es una capa de procesamiento que se ejecuta antes y después de cada petición HTTP. Django proporciona un conjunto de middlewares que se pueden utilizar para añadir funcionalidades a la aplicación.

42.1.6 Autenticación y Autorización

Django proporciona un sistema de autenticación y autorización que permite gestionar los usuarios y los permisos de la aplicación de forma sencilla.

42.1.7 Internacionalización

Django proporciona soporte para la internacionalización de la aplicación. Permite traducir la aplicación a diferentes idiomas y gestionar las traducciones de forma sencilla.

42.1.8 Seguridad

Django proporciona un conjunto de medidas de seguridad para proteger la aplicación contra ataques comunes, como la inyección de SQL, la falsificación de solicitudes entre sitios (CSRF) y la inyección de código.

42.1.9 Testing

Django proporciona un conjunto de herramientas para realizar pruebas unitarias y de integración en la aplicación. Permite probar la lógica de la aplicación y asegurarse de que funciona correctamente.

42.1.10 Despliegue

Django proporciona un conjunto de herramientas para desplegar la aplicación en un servidor de producción. Permite configurar el entorno de producción y gestionar las actualizaciones de la aplicación de forma sencilla.

43 Configuración inicial de un proyecto.

43.1 1. Crear un entorno virtual

Figure 43.1: Creación de entorno Virtual

```
python3 -m venv env
```

El comando anterior creará un directorio llamado **env** en el directorio actual, que contendrá un entorno virtual de Python.

43.2 2. Activar el entorno virtual

Figure 43.2: Activación de entorno Virtual

```
source env/bin/activate
```

El comando anterior activará el entorno virtual en sistemas Unix. En Windows, el comando es:

```
env\Scripts\activate
```

Este comando tambien se puede dividir en 2 partes:

```
cd env/Scripts/
activate
```

Para desactivar el entorno virtual, simplemente ejecute:

```
deactivate
```

43.3 3. Instalar Django

Figure 43.3: Instalación de Django

```
pip install django==4.2
```

El comando anterior instalará la última versión de Django en el entorno virtual.

43.4 4. Crear un proyecto de Django

Figure 43.4: Creación de un Proyecto en Django

```
django-admin startproject helloWorld .
```

El comando anterior creará un nuevo directorio llamado **helloWorld** en el directorio actual, que contendrá un proyecto de Django.

43.5 5. Crear una aplicación de Django

Figure 43.5: Creación de una App en Django

```
python manage.py startapp hello
```

El comando anterior creará un nuevo directorio llamado **hello** en el directorio actual, que contendrá una aplicación de Django.

> **Tip**
>
> Recerda que puedes abrir el editor de código Visual Studio Code con el comando **code .**

43.6 6. Crear una vista

Figure 43.6: Vistas en Django

```
# hello/views.py

from django.http import HttpResponse

def index(request):
    return HttpResponse("Hello, World!")
```

43.7 7. Configurar las URL

Figure 43.7: URLs de la App en Django

```
# helloWorld/urls.py

from django.contrib import admin
from django.urls import include, path

urlpatterns = [
    path("", include("hello.urls")),
    path("admin/", admin.site.urls),
]
```

43.8 8. Ejecutar el servidor de desarrollo

Figure 43.8: Servidor de Desarrollo en Django

```
python manage.py runserver
```

El comando anterior ejecutará el servidor de desarrollo de Django. Para acceder al servidor, abra un navegador web y vaya a la dirección **http://0.0.0.0:8000/**.

Figure 43.9: Visualizar el servidor corriendo desde el navegador

> **Tip**
>
> Para detener el servidor de desarrollo, presione **Ctrl + C** en la terminal.

43.9 9. Crear una migración

Figure 43.10: Preparación de las Migraciones en Django

```
python manage.py makemigrations
```

El comando anterior creará una migración para los cambios en los modelos de la base de datos.

43.10 10. Aplicar una migración

```
statick@lenovo [v] 3.12.3 ~/workspaces/course_of_django python manage.py migrate
Operations to perform:
  Apply all migrations: admin, auth, contenttypes, sessions
Running migrations:
  Applying contenttypes.0001_initial... OK
  Applying auth.0001_initial... OK
  Applying admin.0001_initial... OK
  Applying admin.0002_logentry_remove_auto_add... OK
  Applying admin.0003_logentry_add_action_flag_choices... OK
  Applying contenttypes.0002_remove_content_type_name... OK
  Applying auth.0002_alter_permission_name_max_length... OK
  Applying auth.0003_alter_user_email_max_length... OK
  Applying auth.0004_alter_user_username_opts... OK
  Applying auth.0005_alter_user_last_login_null... OK
  Applying auth.0006_require_contenttypes_0002... OK
  Applying auth.0007_alter_validators_add_error_messages... OK
  Applying auth.0008_alter_user_username_max_length... OK
  Applying auth.0009_alter_user_last_name_max_length... OK
  Applying auth.0010_alter_group_name_max_length... OK
  Applying auth.0011_update_proxy_permissions... OK
  Applying auth.0012_alter_user_first_name_max_length... OK
  Applying sessions.0001_initial... OK
statick@lenovo [v] 3.12.3 ~/workspaces/course_of_django
```

Figure 43.11: Preparación de las Migraciones en Django

```
python manage.py migrate
```

El comando anterior aplicará la migración a la base de datos.

43.11 12. Crear un superusuario

```
statick@lenovo [v] 3.12.3 ~/workspaces/course_of_django python manage.py createsuperuser
Username (leave blank to use 'statick'):
Email address:
Password:
Password (again):
This password is too short. It must contain at least 8 characters.
This password is too common.
This password is entirely numeric.
Bypass password validation and create user anyway? [y/N]: y
Superuser created successfully.
statick@lenovo [v] 3.12.3 ~/workspaces/course_of_django
```

Figure 43.12: Creación de un Superusuario en Django

```
python manage.py createsuperuser
```

El comando anterior creará un superusuario para acceder al panel de administración de Django.

43.12 13. Acceder al panel de administración

Figure 43.13: Login Admin en Django

Para acceder al panel de administración de Django, abra un navegador web y vaya a la dirección **http://127.0.0.1:8000/admin/**. Inicie sesión con el superusuario creado en el paso anterior.

Figure 43.14: Admin en Django

44 Ejercicio

Crear un proyecto de Django llamado **helloWorld** con una aplicación llamada **hello** que muestre un mensaje de bienvenida en la página de inicio.

Ver solución

```
python3 -m venv env                                    ①
source env/bin/activate                                ②
pip install django==4.2                                ③
django-admin startproject helloWorld .                 ④
python manage.py startapp hello                        ⑤
```

① Crear un entorno virtual.
② Activar el entorno virtual.
③ Instalar Django.
④ Crear un proyecto de Django.
⑤ Crear una aplicación de Django.

```
# hello/views.py

from django.http import HttpResponse                   ①

def index(request):                                    ②
    return HttpResponse("Hello, World!")               ③
```

① Importar la clase **HttpResponse** de **django.http**.
② Crear una vista llamada **index**.
③ Devolver un mensaje de bienvenida.

```
# helloWorld/urls.py

from django.urls import path                           ①
from hello import views                                ②

urlpatterns = [                                        ③
    path("", views.index),                             ④
]
```

① Importar la función **path** de **django.urls**.
② Importar el módulo **views** de la aplicación **hello**.
③ Crear una lista de rutas.

④ Asociar la ruta raíz con la vista **index**.

```
# helloWorld/urls.py

from django.contrib import admin                                    ①
from django.urls import include, path                               ②

urlpatterns = [                                                     ③
    path("", include("hello.urls")),                                ④
    path("admin/", admin.site.urls),                                ⑤
]
```

① Importar el módulo **admin** de **django.contrib**.
② Importar la función **include** y la clase **path** de **django.urls**.
③ Crear una lista de rutas.
④ Incluir las rutas de la aplicación **hello**.
⑤ Incluir las rutas del panel de administración.

```
python manage.py runserver                                          ①
```

① Ejecutar el servidor de desarrollo.

45 Asignación

Desarrolla una aplicación web que muestre una lista de productos en la página de inicio. Cada producto debe tener un nombre, una descripción y un precio. Además, la aplicación debe tener un panel de administración donde se puedan agregar, editar y eliminar productos.

46 Estructura de archivos y carpetas

Django tiene una estructura de archivos y carpetas que se debe seguir para que el proyecto funcione correctamente. A continuación se muestra la estructura de archivos y carpetas de un proyecto Django:

> **Tip**
>
> Recuerda crear el entorno virtual y activarlo antes de ejecutar el comando.
>
> ```
> python -m venv venv
> source venv/bin/activate
> ```
>
> Creamos un directorio con el siguiente comando:
>
> ```
> mkdir myproject
> cd myproject
> ```
>
> Instalamos Django con el siguiente comando:
>
> ```
> pip install django==4.2.0
> ```
>
> Creamos el proyecto con el siguiente comando:
>
> ```
> django-admin startproject myproject .
> ```

```
manage.py # <1>
myproject # <2>
    asgi.py # <3>
    __init__.py # <4>
    settings.py # <5>
    urls.py # <6>
    wsgi.py # <7>
```

1.- Archivo de gestión del proyecto.

2.- Carpeta del proyecto.

3.- Archivo de configuración de ASGI.

4.- Archivo de inicialización del proyecto.

5.- Archivo de configuración del proyecto.

6.- Archivo de configuración de las rutas del proyecto.

7.- Archivo de configuración de WSGI.

47 Creación de una aplicación Django

Para crear una aplicación Django se debe ejecutar el siguiente comando:

```
python manage.py startapp myapp ①
```

① Nombre de la aplicación.

48 Configuración de la base de datos

Para configurar la base de datos se debe modificar el archivo `settings.py` del proyecto. A continuación se muestra un ejemplo de configuración de la base de datos:

```
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.sqlite3',                    ①
        'NAME': BASE_DIR / 'db.sqlite3',                           ②
    }
}
```

① Motor de base de datos.
② Ruta del archivo de la base de datos.

Ejemplo

En este ejemplo crearemos una aplicación que muestre un mensaje en la página principal. Para ello, se deben seguir los siguientes pasos:

1. Crear una vista.
2. Crear una plantilla.
3. Configurar las rutas.

49 Crear una vista

Para crear una vista se debe modificar el archivo **views.py** de la aplicación. A continuación se muestra un ejemplo de vista:

```
from django.http import HttpResponse

def index(request):
    return HttpResponse("Hello, world!")
```

50 Crear una plantilla

Para crear una plantilla se debe crear una carpeta llamada **templates** en la carpeta de la aplicación. A continuación se muestra un ejemplo de plantilla:

```
<!DOCTYPE html>
<html>
<head>
    <title>MyApp</title>
</head>
<body>
    <h1>Hello, world!</h1>
</body>
</html>
```

Para que Django pueda encontrar la plantilla, se debe configurar la ruta de la plantilla en el archivo **settings.py** del proyecto. A continuación se muestra un ejemplo de configuración de la ruta de la plantilla:

```
TEMPLATES = [
    {
        'BACKEND': 'django.template.backends.django.DjangoTemplates',
        'DIRS': [BASE_DIR / 'templates'],                                ①
        'APP_DIRS': True,
        'OPTIONS': {
            'context_processors': [
                'django.template.context_processors.debug',
                'django.template.context_processors.request',
                'django.contrib.auth.context_processors.auth',
                'django.contrib.messages.context_processors.messages',
            ],
        },
    },
]
```

① Ruta de la plantilla.

51 Configurar las rutas

Para configurar las rutas se debe modificar el archivo **urls.py** de la aplicación. A continuación se muestra un ejemplo de configuración de las rutas:

```
from django.urls import path
from . import views

urlpatterns = [
    path('', views.index, name='index'),
]
```

52 Correr el servidor de desarrollo

Para correr el servidor de desarrollo se debe ejecutar el siguiente comando:

```
python manage.py runserver
```

53 Acceder a la aplicación

Para acceder a la aplicación se debe abrir un navegador web y escribir la siguiente URL:

http://127.0.0.1:8000/

Posiblemente sea necesario preparar las migraciones y aplicarlas a la base de datos:

```
python manage.py makemigrations ①
python manage.py migrate ②
```

① Prepara las migraciones.
② Aplica las migraciones a la base de datos.

54 Acceder a la aplicación

Para acceder a la aplicación se debe abrir un navegador web y escribir la siguiente URL:

http://127.0.0.0.1:8000/

Muy bien hecho! Has creado tu primera aplicación Django. Ahora puedes seguir explorando la documentación oficial de Django para aprender más sobre el framework.

55 Asignación

Seguir cada uno de los pasos de esta sección para crear una aplicación Django que muestre un mensaje en la página principal. La aplicación debe tener los siguientes archivos y carpetas:

```
manage.py # <1>
myproject # <2>
    asgi.py # <3>
    __init__.py # <4>
    settings.py # <5>
    urls.py # <6>
    wsgi.py # <7>
myapp # <8>
    __init__.py # <9>
    admin.py # <10>
    apps.py # <11>
    migrations # <12>
        __init__.py # <13>
    models.py # <14>
    tests.py # <15>
    views.py # <16>
    templates # <17>
        index.html # <18>
```

1.- Archivo de gestión del proyecto.

2.- Carpeta del proyecto.

3.- Archivo de configuración de ASGI.

4.- Archivo de inicialización del proyecto.

5.- Archivo de configuración del proyecto.

6.- Archivo de configuración de las rutas del proyecto.

7.- Archivo de configuración de WSGI.

8.- Carpeta de la aplicación.

9.- Archivo de inicialización de la aplicación.

10.- Archivo de configuración del administrador de Django.

11.- Archivo de configuración de la aplicación.

12.- Carpeta de migraciones de la aplicación.

13.- Archivo de inicialización de las migraciones.

14.- Archivo de configuración de los modelos de la aplicación.

15.- Archivo de pruebas de la aplicación.

16.- Archivo de configuración de las vistas de la aplicación.

17.- Carpeta de plantillas de la aplicación.

18.- Archivo de la plantilla de la aplicación.

Recuerda que para que Django pueda encontrar la plantilla, se debe configurar la ruta de la plantilla en el archivo **settings.py** del proyecto. A continuación se muestra un ejemplo de configuración de la ruta de la plantilla:

```
TEMPLATES = [
    {
        'BACKEND': 'django.template.backends.django.DjangoTemplates',
        'DIRS': [BASE_DIR / 'templates'],                                ①
        'APP_DIRS': True,
        'OPTIONS': {
            'context_processors': [
                'django.template.context_processors.debug',
                'django.template.context_processors.request',
                'django.contrib.auth.context_processors.auth',
                'django.contrib.messages.context_processors.messages',
            ],
        },
    },
]
```

① Ruta de la plantilla.

56 Referencias

- Django

57 Modelos

Para entender este tema crearemos un sistema de gestión de inventario de productos. Para ello, crearemos una clase **Producto** que representará un producto en el inventario. Cada producto tendrá un nombre, un precio y una cantidad en inventario.

Empezaremos creando un entorno virtual e instalando Django.

```
python -m venv env
source env/bin/activate
mkdir inventario
cd inventario
pip install django==4.2.0
```

Ahora crearemos un proyecto de Django llamado **inventario**

```
django-admin startproject inventario .
```

Luego crearemos una aplicación llamada **productos**

```
python manage.py startapp productos
```

Info: En Django, un proyecto es un conjunto de aplicaciones web y un proyecto puede contener múltiples aplicaciones.

Para que la app productos funcione, debemos registrarla en el archivo **settings.py** del proyecto **inventario**.

```
INSTALLED_APPS = [
    ...
    "productos",
]
```

Ahora crearemos la clase **Producto** en el archivo **models.py** de la aplicación **productos**.

```
from django.db import models

class Producto(models.Model):
    nombre = models.CharField(max_length=100)
    precio = models.DecimalField(max_digits=10, decimal_places=2)
    cantidad = models.IntegerField()
```

```
    def __str__(self):
        return self.nombre
```

Tip

Tip: La función **__str__** es una función especial que se llama cuando se convierte un objeto a una cadena de texto.

58 Registramos la aplicacion en admin.py

Para que Django reconozca la clase **Producto**, debemos registrarla en el archivo **admin.py** de la aplicación **productos**.

```
from django.contrib import admin
from .models import Producto                                    ①

admin.site.register(Producto)                                   ②
```

① Importamos la clase **Producto**.
② Registramos la clase **Producto** en el panel de administración de Django.

59 Vistas en Django

Ahora crearemos las vistas de nuestro sistema en Django. Para ello, crearemos una función para cada vista que renderizará una plantilla HTML.

> Tip
>
> En Django, una vista es una función que recibe una petición HTTP y devuelve una respuesta HTTP.

> Tip
>
> **¿Qué es un CRUD?**
> Un CRUD es un acrónimo que significa **Crear**, **Leer**, **Actualizar** y **Eliminar**. Es un conjunto de operaciones básicas que se pueden realizar en una base de datos o en un sistema de gestión de datos.

59.1 Listar productos

Para listar los productos en inventario, crearemos una función **listar_productos** que renderizará la plantilla **listar.html** con la lista de productos.

```
from pyexpat.errors import messages
from django.shortcuts import render, redirect, get_object_or_404
from .models import Producto
from django.urls import reverse

productos = []

def listar_productos(request):
    productos = Producto.objects.all()
    return render(request, 'listar.html', {'productos': productos})
```

59.2 Agregar producto

Para agregar un producto al inventario, crearemos una función **agregar_producto** que recibe los datos del producto a agregar y lo agrega a la lista de productos.

```
def agregar_producto(request):
    if request.method == "POST":
        nombre = request.POST.get("nombre")
        precio = request.POST.get("precio")
        cantidad = request.POST.get("cantidad")
        Producto.objects.create(nombre=nombre, precio=precio, cantidad=cantidad)
        return redirect('productos:listar_productos')
    return render(request, "agregar.html")
```

59.3 Actualizar producto

Para actualizar un producto en el inventario, crearemos una función **actualizar_producto** que recibe los datos del producto a actualizar y actualiza el precio y la cantidad del producto.

```
def actualizar_producto(request, id):
    producto = get_object_or_404(Producto, pk=id)
    if request.method == 'POST':
        nombre = request.POST.get('nombre')
        precio = request.POST.get('precio')
        cantidad = request.POST.get('cantidad')

        # Actualiza los campos del producto
        producto.nombre = nombre
        producto.precio = precio
        producto.cantidad = cantidad
        producto.save()

        return redirect('productos:listar_productos')
    else:
        return render(request, 'actualizar.html', {'producto': producto})
```

59.4 Eliminar producto

Para eliminar un producto del inventario, crearemos una función **eliminar_producto** que recibe el nombre del producto a eliminar y lo elimina de la lista de productos.

```
def eliminar_producto(request):
    if request.method == "POST":
        nombre = request.POST.get("nombre")
        try:
            producto = Producto.objects.get(nombre=nombre)
            producto.delete()
        except Producto.DoesNotExist:
```

```
            pass

        return redirect('productos:listar_productos')
    return render(request, "eliminar.html")
```

59.5 Buscar producto

Para buscar un producto en el inventario, crearemos una función **buscar_producto** que recibe el nombre del producto a buscar y renderiza la plantilla **buscar.html** con el producto encontrado.

```
def buscar_producto(request):
    if request.method == "POST":
        nombre = request.POST.get("nombre")
        try:
            producto = Producto.objects.get(nombre=nombre)
            return render(request, "buscar.html", {"producto": producto})
        except Producto.DoesNotExist:
            return render(request, "buscar.html", {"producto": None})
    return render(request, "buscar.html")
```

60 Templates

Django utiliza un sistema de herencia de plantillas llamado Jinja2. Adicional a ello uilizaremos un framework de CSS llamado Bootstrap.

> Tip
>
> **Jinja2** es un motor de plantillas para Python que se utiliza en Django para renderizar plantillas HTML.

> Tip
>
> **Bootstrap** es un framework de CSS que se utiliza para crear sitios web y aplicaciones web responsivos y móviles.

60.1 Base

Crearemos un archivo **base.html** que contendrá la estructura base de todas las páginas de nuestro sistema.

```
<!DOCTYPE html>
<html lang="es">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>
        {% block title %}
        Inventario
        {% endblock %}
    </title>
    <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.3/dist/css/bootstrap.min.css" 
</head>
<body>
    <div class="container">
        {% block content %}
        {% endblock %}
    </div>
<script src="https://cdn.jsdelivr.net/npm/bootstrap@5.3.3/dist/js/bootstrap.bundle.min.js
</body>
</html>
```

60.2 Listar

Crearemos un archivo **listar.html** que contendrá la lista de productos en inventario.

```
{% extends "base.html" %}

{% block title %} Listar Productos {% endblock %}

{% block content%}
<h1>Listar Productos</h1>
<ul class="list-group">
    {% for producto in productos %}
    <li class="list-group-item">
        {{ producto.nombre }} - {{ producto.precio }} - {{ producto.cantidad }}
    </li>
    {% endfor %}
</ul>
{% endblock%}
```

60.3 Agregar

Crearemos un archivo **agregar.html** que contendrá un formulario para agregar un producto al inventario.

```
{% extends "base.html" %}

{% block title %}Agregar producto{% endblock %}

{% block content %}
<h1>Agregar producto</h1>

<form action="{% url 'productos:agregar_producto' %}" method="post">
    {% csrf_token %}
    <div class="mb-3">
        <label for="nombre" class="form-label">Nombre</label>
        <input type="text" class="form-control" id="nombre" name="nombre">
    </div>
    <div class="mb-3">
        <label for="precio" class="form-label">Precio</label>
        <input type="number" class="form-control" id="precio" name="precio">
    </div>
    <div class="mb-3">
        <label for="cantidad" class="form-label">Cantidad</label>
        <input type="number" class="form-control" id="cantidad" name="cantidad">
    </div>
    <button type="submit" class="btn btn-primary">Agregar</button>
```

```
</form>
{% endblock %}
```

60.4 Actualizar

Crearemos un archivo **actualizar.html** que contendrá un formulario para actualizar un producto en el inventario.

```
{% extends "base.html" %}

{% block title %}Actualizar producto{% endblock %}

{% block content %}
<h1>Actualizar producto</h1>
<form action="{% url 'productos:actualizar_producto' producto.id %}" method="post">
    {% csrf_token %}
    <input type="hidden" name="nombre" value="{{ producto.nombre }}">
    <input type="hidden" name="id" value="{{ producto.id }}"> {# Agregamos un campo ocult
    <div class="mb-3">
        <label for="precio" class="form-label">Precio</label>
        <input type="number" class="form-control" id="precio" name="precio" value="{{ pro
    </div>
    <div class="mb-3">
        <label for="cantidad" class="form-label">Cantidad</label>
        <input type="number" class="form-control" id="cantidad" name="cantidad" value="{{
    </div>
    <button type="submit" class="btn btn-primary">Actualizar</button>
</form>
{% endblock %}
```

60.5 Eliminar

Crearemos un archivo **eliminar.html** que contendrá un formulario para eliminar un producto del inventario.

```
{% extends "base.html" %}

{% block title %}Eliminar producto{% endblock %}

{% block content %}
<h1>Eliminar producto</h1>
<form action="{% url 'productos:eliminar_producto' %}" method="post">
    {% csrf_token %}  <!-- Agrega el token CSRF aquí -->
    <div class="mb-3">
        <label for="nombre" class="form-label">Nombre</label>
```

```
        <input type="text" class="form-control" id="nombre" name="nombre">
    </div>
    <button type="submit" class="btn btn-primary">Eliminar</button>
</form>
{% endblock %}
```

60.6 Buscar

Crearemos un archivo **buscar.html** que contendrá un formulario para buscar un producto en el inventario.

```
{% extends "base.html" %}

{% block title %}Buscar producto{% endblock %}

{% block content %}
<h1>Buscar producto</h1>
<form action="{% url 'productos:buscar_producto' %}" method="post">
    {% csrf_token %}
    <div class="mb-3">
        <label for="nombre" class="form-label">Nombre</label>
        <input type="text" class="form-control" id="nombre" name="nombre">
    </div>
    <button type="submit" class="btn btn-primary">Buscar</button>
</form>

{% if producto %}
<div class="mt-3">
    <h2>Información del producto:</h2>
    <p>Nombre: {{ producto.nombre }}</p>
    <p>Precio: {{ producto.precio }}</p>
    <p>Cantidad: {{ producto.cantidad }}</p>
</div>
{% endif %}
{% endblock %}
```

61 URLs

Para que nuestro sistema funcione, necesitamos definir las URLs que se utilizarán para acceder a las diferentes vistas.

61.1 URLs en la aplicación y el proyecto

En el archivo **urls.py** de la aplicación **productos** definiremos las URLs de las vistas de nuestro sistema.

```
from django.urls import path
from . import views

app_name = 'productos'

urlpatterns = [
    path('', views.listar_productos, name='listar_productos'),
    path('agregar/', views.agregar_producto, name='agregar_producto'),
    path('actualizar/<int:id>/', views.actualizar_producto, name='actualizar_producto'),
    path('eliminar/', views.eliminar_producto, name='eliminar_producto'),
    path('buscar/', views.buscar_producto, name='buscar_producto'),
]
```

En el archivo **urls.py** del proyecto **inventario** incluiremos las URLs de la aplicación **productos**.

```
from django.contrib import admin
from django.urls import path, include

urlpatterns = [
    path('admin/', admin.site.urls),
    path('', include('productos.urls')),
]
```

Tip

Tip: Para acceder al panel de administración de Django, debemos crear un superusuario con el siguiente comando.

```
python manage.py createsuperuser
```

Tip

Si realizamos modificaciones en el modelo de datos, debemos aplicar las migraciones con el siguiente comando.

```
python manage.py makemigrations ①
python manage.py migrate ②
```

① Crea las migraciones.
② Aplica las migraciones.

62 Ejecutar el servidor

Para ejecutar el servidor de desarrollo de Django, utilizaremos el siguiente comando.

```
python manage.py runserver
```

Con esto hemos creado un sistema de gestión de inventario de productos con Django. Ahora podemos listar, agregar, actualizar, eliminar y buscar productos en el inventario y hemos utilizado el sistema de plantillas de Django para crear las vistas de nuestro sistema. Adicional a ello tambien hemos utilizado el framework de CSS Bootstrap para darle estilo a nuestro sistema.

63 Django Rest Framework

Django Rest Framework es una librería que nos permite crear APIs REST de una forma sencilla y rápida.

En el proyecto que hemos iniciado vamos a utilizar Django Rest Framework para crear una API REST que nos permita gestionar los datos de nuestra aplicación.

63.1 Instalación

Para instalar Django Rest Framework, ejecutamos el siguiente comando:

```
pip install djangorestframework
```

Una vez instalado, añadimos **'rest_framework'** a la lista de aplicaciones instaladas en el archivo settings.py:

```
INSTALLED_APPS = [
    ...
    'rest_framework',
]
```

63.2 Actualizar el archivo requirements.txt

Es necesario eliminar el archivo requirements.txt y volver a crearlo con el siguiente comando:

```
pip freeze > requirements.txt
```

63.3 Serializers

Los serializadores nos permiten convertir los datos de nuestro modelo en un formato que pueda ser fácilmente consumido por una API REST.

Para crear un serializador, creamos un archivo serializers.py en la carpeta de nuestra aplicación y añadimos el siguiente código:

```
from rest_framework import serializers
from .models import Producto

class ProductoSerializer(serializers.ModelSerializer):
    class Meta:
        model = Producto
        fields = '__all__'
```

63.4 Views

Las vistas en Django Rest Framework son similares a las vistas en Django. En lugar de devolver una respuesta HTML, devuelven una respuesta JSON que puede ser consumida por una API REST.

Para crear una vista, creamos un archivo views.py en la carpeta de nuestra aplicación y añadimos el siguiente código:

```
from rest_framework import viewsets
from .serializers import ProductoSerializer
from .models import Producto

class ProductoViewSet(viewsets.ModelViewSet):
    queryset = Producto.objects.all()
    serializer_class = ProductoSerializer
```

63.5 URLs de la Aplicación

Para conectar nuestras vistas con las URLs de nuestra aplicación, creamos un archivo urls.py en la carpeta de nuestra aplicación y añadimos el siguiente código:

```
from django.urls import path
from .views import ProductoViewSet

urlpatterns = [
    path('api/productos/', ProductoViewSet.as_view({'get': 'list', 'post': 'create'}), na
    path('api/productos/<int:pk>/', ProductoViewSet.as_view({'get': 'retrieve', 'put': 'u
]
```

63.6 Configuración URLs del Proyecto

Para configurar nuestra API REST, añadimos las URLs de nuestra aplicación a las URLs del proyecto en el archivo urls.py de la carpeta del proyecto:

```
from django.contrib import admin
from django.urls import path, include
from rest_framework import routers
from productos.views import ProductoViewSet

# Creamos un enrutador para las vistas de Django REST Framework
router = routers.DefaultRouter()
router.register(r'productos', ProductoViewSet)

urlpatterns = [
    path('admin/', admin.site.urls),
    path('', include('productos.urls')),
    path('api/', include(router.urls)),
]
```

63.7 Migraciones

Antes de utilizar nuestra API REST, debemos aplicar las migraciones necesarias para crear las tablas en la base de datos:

```
python manage.py makemigrations
python manage.py migrate
```

63.8 Ejecución

Una vez configurada nuestra API REST, podemos ejecutar el servidor de desarrollo de Django y acceder a la API a través de un navegador o una herramienta como Postman:

```
python manage.py runserver
```

En este caso, la API estará disponible en la ruta 'http://127.0.0.1:8000/api/productos/'.

64 Documentación de la API con drf-yasg

Primero instalamos el paquete de documentación de la API:

```
pip install drf-yasg
```

Luego, añadimos 'drf_yasg' a la lista de aplicaciones instaladas en el archivo settings.py:

```
INSTALLED_APPS = [
    ...
    'drf_yasg',
]
```

Añadimos las URLs de la documentación de la API a las URLs del proyecto en el archivo urls.py de la carpeta del proyecto:

```
from django.urls import path, include
from rest_framework import routers
from productos.views import ProductoViewSet
from rest_framework.permissions import AllowAny
from drf_yasg.views import get_schema_view
from drf_yasg import openapi

router = routers.DefaultRouter()
router.register(r'productos', ProductoViewSet)

schema_view = get_schema_view(
   openapi.Info(
      title="API de Productos",
      default_version='v1',
      description="Documentación de la API de Productos",
      terms_of_service="https://www.google.com/policies/terms/",
      contact=openapi.Contact(email="contact@example.com"),
      license=openapi.License(name="BSD License"),
   ),
   public=True,
   permission_classes=(AllowAny,),
)

urlpatterns = [
    path('admin/', admin.site.urls),
    path('', include('productos.urls')),
```

```
    path('api/', include(router.urls)),
    path('swagger/', schema_view.with_ui('swagger', cache_timeout=0), name='schema-swagge
    path('redoc/', schema_view.with_ui('redoc', cache_timeout=0), name='schema-redoc'),
]
```

Una vez configurada la documentación de la API, podemos ejecutar el servidor de desarrollo de Django y acceder a la documentación de la API a través de un navegador:

```
python manage.py runserver
```

65 Documentación de la API con CoreAPI

CoreAPI es una biblioteca que facilita la creación y el consumo de APIs.

65.1 Primero instalamos CoreAPI:

```
pip install coreapi
```

Para generar la documentación automáticamente, agregamos la configuración en settings.py:

```
REST_FRAMEWORK = {
    'DEFAULT_SCHEMA_CLASS': 'rest_framework.schemas.coreapi.AutoSchema'
}
```

Luego, añadimos la vista de esquema en el archivo urls.py de la carpeta del proyecto:

```
from rest_framework.schemas import get_schema_view

schema_view = get_schema_view(title='API de Productos')

urlpatterns += [
    path('docs/', schema_view, name='api-docs'),
]
```

Ahora podemos ejecutar el servidor de desarrollo y acceder a la documentación generada por CoreAPI en la ruta '/docs/':

```
python manage.py runserver
```

Con estos pasos, hemos integrado Django Rest Framework, documentado nuestra API con drf-yasg y CoreAPI.

66 Testing

Vamos a empezar la parte de testing para ello vamos a crear en la ruta raiz el archivo pytest.ini

```
[pytest]

DJANGO_SETTINGS_MODULE = inventario.test_settings

python_files = tests.py test_*.py *_tests.py
```

Ahora vamos a crear el archivo test_settings.py en la carpeta inventario

```
from .settings import *

# Configuraciones adicionales para el entorno de prueba
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.sqlite3',
        'NAME': ':memory:',
    }
}
```

Ahora vamos a crear el archivo test_productos.py en la carpeta inventario

```
import pytest
from productos.models import Producto

@pytest.mark.django_db
def test_crear_producto():
    producto = Producto.objects.create(
        nombre="Producto de prueba",
        precio=10.00,
        cantidad=5
    )
    assert producto.nombre == "Producto de prueba"
    assert producto.precio == 10.00
    assert producto.cantidad == 5

@pytest.mark.django_db
def test_str_producto():
    producto = Producto.objects.create(
```

```
        nombre="Producto de prueba",
        precio=10.00,
        cantidad=5
    )
    assert str(producto) == "Producto de prueba"
```

Ahora vamos a ejecutar el comando pytest

```
pytest
```

Si todo esta correcto deberiamos ver algo como esto

```
========================================================= test session starts ===============
platform linux -- Python 3.12.3, pytest-8.2.0, pluggy-1.5.0
django: version: 4.2, settings: inventario.test_settings (from ini)
rootdir: /home/statick/workspaces/Curso_django_and_react/inventario_django
configfile: pytest.ini
plugins: django-4.8.0
collected 2 items

productos/tests/test_productos.py ..

========================================================= warnings summary =================
```

Con esto podemos ver que los test han pasado correctamente.

Podemos ver que pytest.ini tiene la configuracion de DJANGO_SETTINGS_MODULE = inventario.test_settings, esto es para que pytest pueda encontrar las configuraciones de django.

67 Corrección de tests en Django Rest Framework

67.1 Introducción

En este documento se describen los pasos necesarios para corregir los tests de Django Rest Framework.

Actualmente tenemos el siguiente error en los tests:

```
=================================================== test session starts ===================
platform linux -- Python 3.12.3, pytest-8.2.0, pluggy-1.5.0
rootdir: /home/statick/workspaces/Curso_django_and_react/inventario_django
collected 0 items / 1 error

========================================================== ERRORS ==========================
_____________________________________ ERROR collecting productos/test_views.py _______
ImportError while importing test module '/home/statick/workspaces/Curso_django_and_react/
Hint: make sure your test modules/packages have valid Python names.
Traceback:
/usr/lib64/python3.12/importlib/__init__.py:90: in import_module
    return _bootstrap._gcd_import(name[level:], package, level)
productos/test_views.py:1: in <module>
    from django.urls import reverse
E   ModuleNotFoundError: No module named 'django'
================================================= short test summary info =================
ERROR productos/test_views.py
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!! Interrupted: 1 error during collection !!!!!!!!
==================================================== 1 error in 0.07s ====================
```

67.2 Pasos

67.2.1 1. Crear un entorno virtual

```
python3 -m venv venv
```

67.2.2 2. Activar el entorno virtual

```
source venv/bin/activate
```

67.2.3 3. Instalar las dependencias

```
pip install -r requirements.txt
```

67.2.4 4. Correr los tests

```
pytest
```

67.2.5 5. Corregir los tests

Para corregir los tests, se debe modificar el archivo `productos/test_views.py` y corregir el error de importación.

67.2.6 6. Correr los tests nuevamente

```
pytest
```

67.2.7 7. Desactivar el entorno virtual

```
deactivate
```

67.3 Conclusión

Una vez corregidos los tests, se debe hacer un pull request al repositorio original.

67.4 Referencias

- Django Rest Framework
- Django
- pytest
- Virtualenv
- Python
- Git
- GitHub
- Markdown
- Pip

Part VI

Unidad 6: Frontend

68 Introducción a HTML

HTML es un lenguaje de marcado que se utiliza para el desarrollo de páginas de internet. Es un estándar que sirve de referencia del software que conecta con la elaboración de páginas web en sus diferentes versiones, define una estructura básica y un código (denominado código HTML) para la definición de contenido de una página web, como texto, imágenes, videos, juegos, entre otros. Es un estándar a cargo del W3C, organización dedicada a la estandarización de casi todas las tecnologías ligadas a la web, sobre todo en lo referente a su escritura e interpretación. Se considera el lenguaje más importante en el desarrollo de páginas web.

HTML se escribe en forma de "etiquetas", rodeadas por corchetes angulares (<,>). HTML también puede describir, hasta un cierto punto, la apariencia de un documento, y puede incluir un script (por ejemplo JavaScript), el cual puede afectar el comportamiento de navegadores web y otros procesadores de HTML.

HTML se ha desarrollado en diferentes versiones, llamadas "versiones de HTML". Sin embargo, su núcleo es siempre el mismo: definir la estructura de una página web. Las versiones de HTML son:

- HTML 2
- HTML 3
- HTML 4
- HTML 5

HTML 5 es la última versión de HTML, y es la que se utiliza actualmente.

68.1 Estructura básica de un documento HTML

Un documento HTML tiene una estructura básica que debe seguir. Esta estructura es la siguiente:

```
<!DOCTYPE html>
<html>
    <head>
        <title>Título de la página</title>
    </head>
    <body>
        <!-- Contenido de la página -->
    </body>
</html>
```

A continuación se describen las partes de un documento HTML:

- **!DOCTYPE html**: Define la versión de HTML que se está utilizando. En este caso, HTML 5.
- **html**: Es el elemento raíz de un documento HTML.
- **head**: Contiene información sobre el documento, como su título.
- **title**: Define el título de la página, que se muestra en la pestaña del navegador.
- **body**: Contiene el contenido de la página, como texto, imágenes, videos, etc.

68.2 Etiquetas HTML

Las etiquetas HTML son los bloques de construcción de un documento HTML. Las etiquetas se utilizan para definir la estructura y el contenido de la página. Cada etiqueta tiene un propósito específico y puede contener otros elementos HTML.

A continuación se muestran algunas etiquetas HTML comunes:

- **
  a
  **: Define encabezados de diferentes tamaños.
- **
  **: Define un párrafo de texto.
- : Define un enlace a otra página.
- : Define una imagen.
- **
  **: Define una lista desordenada.
- **
  **: Define una lista ordenada.
- **
  **: Define un elemento de lista.
- **
  **: Define una tabla.
- **
  **: Define una fila en una tabla.
- **
  **: Define una celda en una tabla.
- **
  **: Define una sección genérica de una página.
- : Define una sección en línea de una página.

68.3 Atributos HTML

Los atributos HTML proporcionan información adicional sobre los elementos HTML. Los atributos se especifican en la etiqueta de inicio y se utilizan para modificar el comportamiento o la apariencia del elemento.

A continuación se muestran algunos atributos HTML comunes:

- **href**: Especifica la URL de destino de un enlace.

- **src**: Especifica la URL de origen de una imagen.
- **alt**: Especifica un texto alternativo para una imagen.
- **width**: Especifica el ancho de un elemento.
- **height**: Especifica la altura de un elemento.
- **style**: Especifica estilos CSS para un elemento.

68.4 Ejemplo de un documento HTML

A continuación se muestra un ejemplo de un documento HTML básico que contiene un encabezado, un párrafo de texto y una imagen:

```html
<!DOCTYPE html>
<html>
    <head>
        <title>Ejemplo de HTML</title>
    </head>
    <body>
        <h1>Este es un encabezado</h1>
        <p>Este es un párrafo de texto.</p>
        <img src="imagen.jpg" alt="Una imagen">
    </body>
</html>
```

En este ejemplo, se define un encabezado de nivel 1 con el texto "Este es un encabezado", un párrafo de texto con el texto "Este es un párrafo de texto" y una imagen con la URL "imagen.jpg" y el texto alternativo "Una imagen".

68.5 Recursos adicionales

- W3Schools HTML Tutorial
- Mozilla Developer Network HTML Guide
- HTML Reference
- HTML Cheat Sheet

68.6 Ejercicios

1. Crea un documento HTML que contenga un encabezado de nivel 1 con el texto "Hola, mundo!" y un párrafo de texto con el texto "Este es un ejemplo de un documento HTML básico".

2. Crea un documento HTML que contenga una lista desordenada con tres elementos de lista. Los elementos de lista deben contener los siguientes textos: "Elemento 1", "Elemento 2" y "Elemento 3".

3. Crea un documento HTML que contenga una tabla con tres filas y tres columnas. En cada celda de la tabla, coloca un número del 1 al 9.
4. Crea un documento HTML que contenga un enlace a la página de Google. El enlace debe tener el texto “Ir a Google” y debe abrirse en una nueva pestaña del navegador.
5. Crea un documento HTML que contenga una imagen de tu elección. La imagen debe tener un ancho de 200 píxeles y un alto de 200 píxeles.

69 Introducción a CSS

69.1 ¿Qué es CSS?

CSS es un lenguaje que describe el estilo de un documento HTML. CSS describe cómo se mostrarán los elementos HTML en la pantalla, en papel o en otros medios. CSS ahorra mucho trabajo. Puede controlar el diseño de varias páginas web a la vez.

69.2 ¿Cómo funciona CSS?

Cuando un navegador lee un documento HTML, construye un modelo de objeto de documento (DOM) basado en el contenido del documento. El navegador luego aplica CSS al DOM para determinar cómo se deben mostrar los elementos HTML en la pantalla.

69.3 ¿Cómo se aplica CSS a un documento HTML?

Hay tres formas de aplicar CSS a un documento HTML:

- **CSS externo**: El CSS externo se define en un archivo .css separado y se aplica a un documento HTML con la etiqueta `<link>`.
- **CSS interno**: El CSS interno se define en la sección `<style>` de un documento HTML y se aplica a ese documento HTML.
- **CSS en línea**: El CSS en línea se define en el atributo `style` de un elemento HTML.

69.4 Sintaxis de CSS

Un archivo CSS consta de un conjunto de reglas CSS. Cada regla CSS consta de un selector y un bloque de declaración.

```
selector {
  propiedad: valor;
}
```

- **Selector**: Un selector es un nombre que identifica un elemento HTML al que se aplicará un estilo.
- **Propiedad**: Una propiedad es un atributo de un elemento HTML que se puede cambiar con CSS.

- **Valor**: Un valor es el valor que se asigna a una propiedad.

69.5 Ejemplo de CSS

```
h1 {
  color: blue;
  font-size: 24px;
}

p {
  color: red;
  font-size: 16px;
}
```

En este ejemplo, el selector `h1` selecciona todos los elementos `<h1>` en el documento HTML y les aplica un color azul y un tamaño de fuente de 24 píxeles. El selector `p` selecciona todos los elementos `<p>` en el documento HTML y les aplica un color rojo y un tamaño de fuente de 16 píxeles.

69.6 Comentarios en CSS

Los comentarios en CSS se pueden agregar entre `/*` y `*/`.

```
/* Este es un comentario en CSS */
```

69.7 Selectores CSS

Los selectores CSS se utilizan para seleccionar los elementos HTML a los que se aplicará un estilo. Hay varios tipos de selectores CSS:

- **Selector de tipo**: Selecciona todos los elementos de un tipo específico.

```
h1 {
  color: blue;
}
```

- **Selector de clase**: Selecciona todos los elementos que tienen un atributo `class` específico.

```
.blue-text {
  color: blue;
}
```

- **Selector de ID**: Selecciona un elemento que tiene un atributo `id` específico.

```
#blue-text {
  color: blue;
}
```

- **Selector de atributo**: Selecciona elementos con un atributo específico.

```
[title] {
  color: blue;
}
```

- **Selector de combinación**: Selecciona elementos que coinciden con varios selectores.

```
h1, p {
  color: blue;
}
```

- **Selector descendente**: Selecciona elementos que son descendientes de otro elemento.

```
div p {
  color: blue;
}
```

- **Selector de hijo**: Selecciona elementos que son hijos directos de otro elemento.

```
div > p {
  color: blue;
}
```

- **Selector de hermano adyacente**: Selecciona elementos que son hermanos adyacentes de otro elemento.

```
p + p {
  color: blue;
}
```

- **Selector de hermano general**: Selecciona elementos que son hermanos de otro elemento.

```
p ~ p {
  color: blue;
}
```

69.8 Propiedades CSS

Las propiedades CSS se utilizan para cambiar el aspecto de los elementos HTML. Hay muchas propiedades CSS que se pueden utilizar para cambiar el aspecto de los elementos HTML. Algunas de las propiedades CSS más comunes son:

- **color**: Establece el color del texto.

```
h1 {
  color: blue;
}
```

- **font-size**: Establece el tamaño de la fuente.

```
h1 {
  font-size: 24px;
}
```

- **font-family**: Establece la fuente del texto.

```
h1 {
  font-family: Arial, sans-serif;
}
```

- **text-align**: Establece la alineación del texto.

```
h1 {
  text-align: center;
}
```

- **background-color**: Establece el color de fondo.

```
body {
  background-color: lightgray;
}
```

- **border**: Establece el borde de un elemento.

```
div {
  border: 1px solid black;
}
```

- **margin**: Establece el margen de un elemento.

```
div {
  margin: 10px;
}
```

- **padding**: Establece el relleno de un elemento.

```
div {
  padding: 10px;
}
```

- **width**: Establece el ancho de un elemento.

```
div {
  width: 100px;
}
```

- **height**: Establece la altura de un elemento.

```
div {
  height: 100px;
}
```

- **display**: Establece cómo se mostrará un elemento.

```
div {
  display: block;
}

span {
  display: inline;
}
```

- **position**: Establece la posición de un elemento.

```
div {
  position: relative;
}
```

- **top**: Establece la posición superior de un elemento.

```
div {
  top: 10px;
}
```

- **right**: Establece la posición derecha de un elemento.

```
div {
  right: 10px;
}
```

- **bottom**: Establece la posición inferior de un elemento.

```
div {
  bottom: 10px;
}
```

- **left**: Establece la posición izquierda de un elemento.

```
div {
  left: 10px;
}
```

- **float**: Establece si un elemento flotará a la izquierda, a la derecha o no flotará.

```
img {
  float: left;
}
```

- **clear**: Establece si un elemento debe estar al lado de los elementos flotantes o no.

```
div {
  clear: both;
}
```

- **z-index**: Establece la pila de elementos.

```
div {
  z-index: 1;
}
```

- **opacity**: Establece la opacidad de un elemento.

```
div {
  opacity: 0.5;
}
```

- **visibility**: Establece si un elemento es visible o no.

```
div {
  visibility: hidden;
}
```

- **overflow**: Establece cómo se manejará el desbordamiento de un elemento.

```
div {
  overflow: hidden;
}
```

- **cursor**: Establece el cursor del ratón cuando se coloca sobre un elemento.

```
a {
  cursor: pointer;
}
```

- **list-style-type**: Establece el tipo de marcador de una lista.

```
ul {
  list-style-type: square;
}
```

- **text-decoration**: Establece la decoración del texto.

```
a {
  text-decoration: none;
}
```

- **text-transform**: Establece cómo se transformará el texto.

```
h1 {
  text-transform: uppercase;
}
```

- **text-shadow**: Establece la sombra del texto.

```
h1 {
  text-shadow: 2px 2px 5px red;
}
```

- **box-shadow**: Establece la sombra de un cuadro.

```
div {
  box-shadow: 2px 2px 5px gray;
}
```

- **border-radius**: Establece el radio de los bordes de un elemento.

```
div {
  border-radius: 10px;
}
```

- **transition

```
div {
  transition: width 2s, height 2s;
}
```

- **animation

```
div {
  animation: example 5s infinite;
}
```

69.9 Unidades CSS

Las unidades CSS se utilizan para especificar longitudes y tamaños. Hay varias unidades CSS que se pueden utilizar para especificar longitudes y tamaños. Algunas de las unidades CSS más comunes son:

- **px**: Píxeles.

```
h1 {
  font-size: 24px;
}
```

- **em**: Relativo al tamaño de la fuente del elemento.

```
h1 {
  font-size: 2em;
}
```

- **rem**: Relativo al tamaño de la fuente del elemento raíz.

```
h1 {
  font-size: 2rem;
}
```

- **%**: Porcentaje del tamaño del contenedor.

```
div {
  width: 50%;
}

img {
  width: 100%;
}
```

- **vw**: Porcentaje del ancho de la ventana gráfica.

```
div {
  width: 50vw;
}

img {
  width: 100vw;
}
```

70 JavaScript

Para empezar a aprender JavaScript es necesario tener conocimientos básicos de HTML y CSS. JavaScript es un lenguaje de programación que se utiliza para crear páginas web interactivas. Es un lenguaje de programación del lado del cliente, lo que significa que se ejecuta en el navegador del usuario, en lugar de en el servidor web.

JavaScript es un lenguaje de programación de alto nivel, interpretado y orientado a objetos. Es un lenguaje de programación que se utiliza para crear páginas web interactivas. JavaScript es un lenguaje de programación del lado del cliente, lo que significa que se ejecuta en el navegador del usuario, en lugar de en el servidor web.

70.1 ¿Qué es JavaScript?

JavaScript es un lenguaje de programación que se utiliza para crear páginas web interactivas. Es un lenguaje de programación del lado del cliente, lo que significa que se ejecuta en el navegador del usuario, en lugar de en el servidor web.

70.2 ¿Dónde puedo aprender JavaScript?

Hay muchos recursos en línea que puedes utilizar para aprender JavaScript. Algunos de los recursos más populares incluyen:

- Codecademy
- MDN Web Docs
- W3Schools

70.3 ¿Qué puedo hacer con JavaScript?

JavaScript se utiliza para crear páginas web interactivas. Algunas de las cosas que puedes hacer con JavaScript incluyen:

- Validar formularios
- Crear animaciones
- Crear juegos
- Crear aplicaciones web

70.4 ¿Qué es un script de JavaScript?

Un script de JavaScript es un archivo de texto que contiene código JavaScript. Los scripts de JavaScript se pueden incrustar en una página web utilizando la etiqueta **script**. Los scripts de JavaScript se utilizan para agregar interactividad a una página web.

70.5 ¿Dónde puedo ejecutar JavaScript?

JavaScript se puede ejecutar en cualquier navegador web moderno. Algunos de los navegadores web más populares que admiten JavaScript incluyen:

- Google Chrome
- Mozilla Firefox
- Microsoft Edge
- Safari

Sin embargo tambien es posible ejecutar JavaScript en el servidor utilizando Node.js.

70.6 ¿Qué es Node.js?

Node.js es un entorno de ejecución de JavaScript del lado del servidor que permite a los desarrolladores crear aplicaciones web utilizando JavaScript. Node.js se basa en el motor de JavaScript V8 de Google Chrome y proporciona una forma de ejecutar JavaScript en el servidor.

70.7 ¿Dónde puedo aprender Node.js?

Hay muchos recursos en línea que puedes utilizar para aprender Node.js. Algunos de los recursos más populares incluyen:

- Node.js
- W3Schools
- MDN Web Docs

70.8 ¿Qué puedo hacer con Node.js?

Node.js se utiliza para crear aplicaciones web del lado del servidor. Algunas de las cosas que puedes hacer con Node.js incluyen:

- Crear servidores web
- Crear aplicaciones web en tiempo real
- Crear aplicaciones de línea de comandos
- Crear aplicaciones de red

71 Primeros pasos con JavaScript

Para empezar a aprender JavaScript, necesitas tener un editor de texto y un navegador web. Puedes utilizar cualquier editor de texto para escribir código JavaScript, en este caso vamos a utilizar Visual Studio Code.

Se sugiere utilizar la siguiente lista de plugins para Visual Studio Code:

- JavaScript (ES6) code snippets
- ESLint
- Prettier - Code formatter

Antes de comenzar la parte práctica de JavaScript, es importante tener en cuenta algunos temas como: Ecmascript 6, Motor V8, Asincronismo, Programación Orientada a Objetos, Programación Funcional, en JavaScript.

71.1 Ecmascript 6

ECMAScript 6 (también conocido como ES6 o ECMAScript 2015) es la sexta versión de ECMAScript, el estándar subyacente de JavaScript. ECMAScript 6 fue lanzado en junio de 2015 y trajo muchas nuevas características y mejoras al lenguaje JavaScript.

Algunas de las características más importantes de ECMAScript 6 incluyen:

- **let y const**: Declaración de variables con let y const.
- **Arrow functions**: Funciones de flecha.
- **Template literals**: Literales de plantilla.
- **Destructuring**: Desestructuración de objetos y arreglos.
- **Spread operator**: Operador de propagación.
- **Classes**: Clases.
- **Promises**: Promesas.
- **Async/Await**: Async/Await.

71.2 Motor V8

V8 es el motor de JavaScript de código abierto de Google que se utiliza en Google Chrome y en Node.js. V8 es un motor de JavaScript de alto rendimiento que compila el código JavaScript en código de máquina nativo antes de ejecutarlo.

V8 utiliza una técnica de compilación just-in-time (JIT) para compilar el código JavaScript en tiempo de ejecución y optimizar su rendimiento. V8 es uno de los motores de JavaScript más rápidos y eficientes disponibles en la actualidad.

71.3 Asincronismo

El asincronismo en JavaScript se utiliza para manejar operaciones asincrónicas, como la lectura de archivos, las solicitudes de red y las interacciones del usuario. JavaScript es un lenguaje de programación basado en eventos, lo que significa que puede ejecutar múltiples tareas simultáneamente sin bloquear el hilo principal.

Para manejar operaciones asincrónicas en JavaScript, puedes utilizar callbacks, promesas o async/await. Los callbacks son funciones que se pasan como argumentos a otras funciones y se ejecutan cuando se completa una operación asincrónica. Las promesas son objetos que representan el resultado de una operación asincrónica y se pueden encadenar para manejar múltiples operaciones asincrónicas. Async/await es una forma más moderna de manejar operaciones asincrónicas en JavaScript y se basa en promesas.

71.4 Programación Orientada a Objetos

La programación orientada a objetos (POO) en JavaScript se utiliza para crear objetos y clases que encapsulan datos y funciones relacionadas. JavaScript es un lenguaje de programación orientado a objetos, lo que significa que puedes crear objetos y clases para representar entidades del mundo real.

Para crear una clase en JavaScript, utiliza la palabra clave **class** seguida del nombre de la clase y define las propiedades y métodos de la clase. Para crear un objeto en JavaScript, utiliza la palabra clave **new** seguida del nombre de la clase y los parámetros de la clase.

71.5 Programación Funcional

La programación funcional en JavaScript se utiliza para crear funciones que se pueden pasar como argumentos a otras funciones y se pueden utilizar para realizar operaciones en datos. JavaScript es un lenguaje de programación funcional, lo que significa que puedes crear funciones de primera clase y funciones de orden superior.

Para crear una función en JavaScript, utiliza la palabra clave **function** seguida del nombre de la función y los parámetros de la función. Para crear una función de orden superior en JavaScript, utiliza funciones de flecha y funciones de retorno.

Con estos temas cubiertos podemos comenzar a escribir código en JavaScript.

71.6 Hola Mundo en JavaScript

Para crear un “Hola Mundo” en JavaScript, sigue estos pasos:

1. Abre Visual Studio Code.
2. Crea un nuevo archivo y guárdalo con la extensión **.html**.
3. Escribe el siguiente código en el archivo:

```html
<!DOCTYPE html>
<html>
<head>
    <title>Hola Mundo en JavaScript</title>
</head>
<body>
    <h1 id="mensaje"></h1>
    <script href='app.js'></script>
</body>
</html>
```

4. Crea un nuevo archivo y guárdalo con la extensión **.js**.

```javascript
document.getElementById("mensaje").innerHTML = "Hola Mundo!";
```

5. Guarda el archivo y ábrelo en un navegador web.
6. Deberías ver un mensaje que dice "Hola Mundo!" en la página.

71.7 Comentarios en JavaScript

Los comentarios en JavaScript se utilizan para explicar el código y hacerlo más legible. Los comentarios en JavaScript comienzan con `//` para comentarios de una sola línea.

```javascript
// Este es un comentario de una sola línea
```

También puedes utilizar comentarios de varias líneas utilizando `/* … */`

```javascript
/*
Este es un comentario de varias líneas
*/
```

71.8 Variables en JavaScript

Las variables en JavaScript se utilizan para almacenar datos. Para declarar una variable en JavaScript, utiliza la palabra clave **var**, **let** o **const** seguida del nombre de la variable.

```javascript
var nombre = "Juan";
let edad = 30;
const PI = 3.1416;
```

> **Tip**
>
> Se recomienda utilizar **let** en lugar de **var** para declarar variables en JavaScript.

71.9 Tipos de datos en JavaScript

JavaScript es un lenguaje de programación de tipado dinámico, lo que significa que no es necesario especificar el tipo de datos de una variable al declararla. JavaScript tiene varios tipos de datos, incluidos:

- **String**: Cadena de texto.
- **Number**: Número.
- **Boolean**: Valor booleano (verdadero o falso).
- **Array**: Arreglo de elementos.
- **Object**: Objeto.
- **Function**: Función.
- **Null**: Valor nulo.
- **Undefined**: Valor indefinido.

```
let nombre = "Juan"; // String
let edad = 30; // Number
let esMayor = true; // Boolean
let colores = ["Rojo", "Verde", "Azul"]; // Array
let persona = {nombre: "Juan", edad: 30}; // Object
let sumar = function(a, b) { return a + b; }; // Function
let nulo = null; // Null
let indefinido = undefined; // Undefined
```

Existen otras opciones de tipos de datos de tipo null y undefined, pero estos son los más comunes.

71.10 Operadores en JavaScript

Los operadores en JavaScript se utilizan para realizar operaciones en variables y valores. Algunos de los operadores más comunes en JavaScript incluyen:

- **Operadores aritméticos**: +, -, *, /, %.
- **Operadores de asignación**: =, +=, -=, *=, /=.
- **Operadores de comparación**: ==, !=, ===, !==, >, <, >=, <=.
- **Operadores lógicos**: &&, ||, !.
- **Operadores de incremento y decremento**: ++, –.
- **Operadores de concatenación**: +.

```
let a = 10;
let b = 5;

let suma = a + b; // 15
let resta = a - b; // 5
let multiplicacion = a * b; // 50
let division = a / b; // 2
```

```
let igual = a == b; // false
let mayor = a > b; // true
let menor = a < b; // false

let and = (a > 0) && (b > 0); // true
let or = (a > 0) || (b > 0); // true
let not = !(a > 0); // false

let incremento = a++; // 10
let decremento = b--; // 5

let cadena = "Hola " + "Mundo!"; // Hola Mundo!
```

71.11 Condicionales en JavaScript

Los condicionales en JavaScript se utilizan para tomar decisiones basadas en ciertas condiciones. Algunos de los condicionales más comunes en JavaScript incluyen:

- **if**: Se utiliza para ejecutar un bloque de código si una condición es verdadera.
- **else**: Se utiliza para ejecutar un bloque de código si una condición es falsa.
- **else if**: Se utiliza para ejecutar un bloque de código si la primera condición es falsa y la segunda condición es verdadera.
- **switch**: Se utiliza para ejecutar diferentes bloques de código según diferentes casos.

```
let edad = 18;

if (edad >= 18) {
    console.log("Eres mayor de edad");
} else {
    console.log("Eres menor de edad");
}

let dia = "Lunes";

switch (dia) {
    case "Lunes":
        console.log("Hoy es Lunes");
        break;
    case "Martes":
        console.log("Hoy es Martes");
        break;
    default:
        console.log("Hoy es otro día");
}
```

71.12 Bucles en JavaScript

Los bucles en JavaScript se utilizan para repetir una serie de instrucciones varias veces. Algunos de los bucles más comunes en JavaScript incluyen:

- **for**: Se utiliza para repetir un bloque de código un número específico de veces.
- **while**: Se utiliza para repetir un bloque de código mientras una condición sea verdadera.
- **do...while**: Se utiliza para repetir un bloque de código al menos una vez y luego repetirlo mientras una condición sea verdadera.

```
for (let i = 0; i < 5; i++) {
    console.log(i);
}

let j = 0;
while (j < 5) {
    console.log(j);
    j++;
}

let k = 0;
do {
    console.log(k);
    k++;
} while (k < 5);
```

71.13 Funciones en JavaScript

Las funciones en JavaScript se utilizan para encapsular un bloque de código y reutilizarlo en diferentes partes de un programa. Para declarar una función en JavaScript, utiliza la palabra clave **function** seguida del nombre de la función y los parámetros de la función.

```
function saludar(nombre) {
    console.log("Hola " + nombre);
}

saludar("Juan");
```

71.14 Eventos en JavaScript

Los eventos en JavaScript se utilizan para manejar la interacción del usuario con una página web. Algunos de los eventos más comunes en JavaScript incluyen:

- **click**: Se activa cuando se hace clic en un elemento.

- **mouseover**: Se activa cuando el puntero del ratón se mueve sobre un elemento.
- **keydown**: Se activa cuando se presiona una tecla.

```html
<!DOCTYPE html>
<html>
<head>
    <title>Eventos en JavaScript</title>
</head>
<body>
    <button id="boton">Haz clic aquí</button>
    <script href="script.js"></script>
</body>
</html>
```

```javascript
document.getElementById("boton").addEventListener("click", function() {
    alert("Haz hecho clic en el botón");
});
```

En los archivos anteriore se muestra un ejemplo de cómo manejar eventos en JavaScript.

71.15 Objetos en JavaScript

Los objetos en JavaScript se utilizan para almacenar colecciones de datos y funciones. Para crear un objeto en JavaScript, utiliza llaves **{}** y define las propiedades y métodos del objeto.

```javascript
let persona = {
    nombre: "Juan",
    edad: 30,
    saludar: function() {
        console.log("Hola, mi nombre es " + this.nombre);
    }
};

console.log(persona.nombre); // Juan
console.log(persona.edad); // 30
persona.saludar(); // Hola, mi nombre es Juan
```

71.16 Arrays en JavaScript

Los arrays en JavaScript se utilizan para almacenar una colección de elementos. Para crear un array en JavaScript, utiliza corchetes [] y separa los elementos con comas.

```
let colores = ["Rojo", "Verde", "Azul"];

console.log(colores[0]); // Rojo
console.log(colores[1]); // Verde
console.log(colores[2]); // Azul
```

71.17 Métodos en JavaScript

Los métodos en JavaScript son funciones que se pueden llamar en un objeto. Algunos de los métodos más comunes en JavaScript incluyen:

- **push()**: Agrega un elemento al final de un array.
- **pop()**: Elimina el último elemento de un array.
- **shift()**: Elimina el primer elemento de un array.
- **unshift()**: Agrega un elemento al principio de un array.
- **splice()**: Agrega o elimina elementos de un array en una posición específica.

```
let colores = ["Rojo", "Verde", "Azul"];

colores.push("Amarillo");
console.log(colores); // ["Rojo", "Verde", "Azul", "Amarillo"]

colores.pop();
console.log(colores); // ["Rojo", "Verde", "Azul"]

colores.shift();
console.log(colores); // ["Verde", "Azul"]

colores.unshift("Rojo");
console.log(colores); // ["Rojo", "Verde", "Azul"]

colores.splice(1, 0, "Amarillo");
console.log(colores); // ["Rojo", "Amarillo", "Verde", "Azul"]
```

71.18 Clases en JavaScript

Las clases en JavaScript se utilizan para crear objetos basados en un modelo. Para crear una clase en JavaScript, utiliza la palabra clave **class** seguida del nombre de la clase y define las propiedades y métodos de la clase.

```
class Persona {
    constructor(nombre, edad) {
        this.nombre = nombre;
        this.edad = edad;
    }
```

```
    saludar() {
        console.log("Hola, mi nombre es " + this.nombre);
    }
}

let juan = new Persona("Juan", 30);
console.log(juan.nombre); // Juan
console.log(juan.edad); // 30
juan.saludar(); // Hola, mi nombre es Juan
```

71.19 Herencia en JavaScript

La herencia en JavaScript se utiliza para crear una clase basada en otra clase. Para heredar una clase en JavaScript, utiliza la palabra clave **extends** seguida del nombre de la clase padre.

```
class Empleado extends Persona {
    constructor(nombre, edad, salario) {
        super(nombre, edad);
        this.salario = salario;
    }

    trabajar() {
        console.log("Estoy trabajando");
    }
}

let pedro = new Empleado("Pedro", 25, 20000);
console.log(pedro.nombre); // Pedro
console.log(pedro.edad); // 25
console.log(pedro.salario); // 20000
pedro.saludar(); // Hola, mi nombre es Pedro
pedro.trabajar(); // Estoy trabajando
```

71.20 Promesas en JavaScript

Las promesas en JavaScript se utilizan para manejar operaciones asincrónicas. Una promesa puede estar en uno de los siguientes estados:

- **Pendiente**: La operación aún no se ha completado.
- **Cumplida**: La operación se ha completado con éxito.
- **Rechazada**: La operación ha fallado.

```
let promesa = new Promise(function(resolve, reject) {
    setTimeout(function() {
        let exito = true;

        if (exito) {
            resolve("La operación se ha completado con éxito");
        } else {
            reject("La operación ha fallado");
        }
    }, 2000);
});

promesa.then(function(resultado) {
    console.log(resultado);
}).catch(function(error) {
    console.log(error);
});
```

71.21 Async/Await en JavaScript

Async/Await en JavaScript se utiliza para manejar operaciones asincrónicas de forma más sencilla. La palabra clave **async** se utiliza para declarar una función asincrónica, mientras que la palabra clave **await** se utiliza para esperar a que una promesa se resuelva.

```
async function operacion() {
    let promesa = new Promise(function(resolve, reject) {
        setTimeout(function() {
            let exito = true;

            if (exito) {
                resolve("La operación se ha completado con éxito");
            } else {
                reject("La operación ha fallado");
            }
        }, 2000);
    });

    let resultado = await promesa;
    console.log(resultado);
}

operacion();
```

71.22 Consumir una api con fetch

En esta sección vamos a consumir una api de pokemon con fetch, para ello vamos a seguir los siguientes pasos:

1. Abre Visual Studio Code.
2. Crea un nuevo archivo y guárdalo con la extensión **.html**.
3. Escribe el siguiente código en el archivo:

```html
<!DOCTYPE html>
<html>
<head>
    <title>Consumir una api con fetch</title>
</head>
<body>
    <ul id="pokemones"></ul>
    <script href="script.js"></script>
</body>
</html>
```

4. Crea un nuevo archivo y guárdalo con la extensión **.js**.

```javascript
fetch("https://pokeapi.co/api/v2/pokemon")
    .then(response => response.json())
    .then(data => {
        let pokemones = data.results;

        pokemones.forEach(pokemon => {
            let li = document.createElement("li");
            li.textContent = pokemon.name;
            document.getElementById("pokemones").appendChild(li);
        });
    })
    .catch(error => console.log(error));
```

5. Guarda el archivo y ábrelo en un navegador web.
6. Deberías ver una lista de nombres de pokemones en la página.

En los archivos anteriores se muestra un ejemplo de cómo consumir una api con fetch en JavaScript.

71.23 Crear una aplicación web con JavaScript

En esta sección vamos a crear una aplicación web con JavaScript, para ello vamos a seguir los siguientes pasos:

1. Abre Visual Studio Code.
2. Crea un nuevo archivo y guárdalo con la extensión **.html**.
3. Escribe el siguiente código en el archivo:

```html
<!DOCTYPE html>
<html>
<head>
    <title>Crear una aplicación web con JavaScript</title>
</head>
<body>
    <h1>Calculadora</h1>
    <input type="text" id="numero1">
    <input type="text" id="numero2">
    <button onclick="sumar()">Sumar</button>
    <button onclick="restar()">Restar</button>
    <button onclick="multiplicar()">Multiplicar</button>
    <button onclick="dividir()">Dividir</button>
    <h2 id="resultado"></h2>
    <script href="script.js"></script>
</body>
</html>
```

4. Crea un nuevo archivo y guárdalo con la extensión **.js**.

```javascript
function sumar() {
    let numero1 = parseInt(document.getElementById("numero1").value);
    let numero2 = parseInt(document.getElementById("numero2").value);
    let resultado = numero1 + numero2;
    document.getElementById("resultado").textContent = resultado;
}

function restar() {
    let numero1 = parseInt(document.getElementById("numero1").value);
    let numero2 = parseInt(document.getElementById("numero2").value);
    let resultado = numero1 - numero2;
    document.getElementById("resultado").textContent = resultado;
}

function multiplicar() {
    let numero1 = parseInt(document.getElementById("numero1").value);
    let numero2 = parseInt(document.getElementById("numero2").value);
    let resultado = numero1 * numero2;
```

```
    document.getElementById("resultado").textContent = resultado;
}

function dividir() {
    let numero1 = parseInt(document.getElementById("numero1").value);
    let numero2 = parseInt(document.getElementById("numero2").value);
    let resultado = numero1 / numero2;
    document.getElementById("resultado").textContent = resultado;
}
```

5. Guarda el archivo y ábrelo en un navegador web.
6. Deberías ver una calculadora en la página.

En los archivos anteriores se muestra un ejemplo de cómo crear una aplicación web con JavaScript.

71.24 Crear un juego con JavaScript

En esta sección vamos a crear un juego con JavaScript, para ello vamos a seguir los siguientes pasos:

1. Abre Visual Studio Code.
2. Crea un nuevo archivo y guárdalo con la extensión **.html**.
3. Escribe el siguiente código en el archivo:

```
<!DOCTYPE html>
<html>
<head>
    <title>Crear un juego con JavaScript</title>
</head>
<body>
    <h1>Adivina el número</h1>
    <input type="text" id="numero">
    <button onclick="adivinar()">Adivinar</button>
    <h2 id="resultado"></h2>
    <script href="script.js"></script>
</body>
</html>
```

4. Crea un nuevo archivo y guárdalo con la extensión **.js**.

```
let numeroAleatorio = Math.floor(Math.random() * 100) + 1;
let intentos = 0;

function adivinar() {
    let numero = parseInt(document.getElementById("numero").value);
    intentos++;

    if (numero === numeroAleatorio) {
        document.getElementById("resultado").textContent = "¡Felicidades! Has adivinado e
    } else if (numero < numeroAleatorio) {
        document.getElementById("resultado").textContent = "El número es mayor";
    } else {
        document.getElementById("resultado").textContent = "El número es menor";
    }
}
```

5. Guarda el archivo y ábrelo en un navegador web.
6. Deberías ver un juego de adivinar el número en la página.

En los archivos anteriores se muestra un ejemplo de cómo crear un juego con JavaScript.

71.25 Conclusiones

JavaScript es un lenguaje de programación poderoso y versátil que se utiliza para crear páginas web interactivas. Con JavaScript, puedes agregar interactividad a una página web, crear aplicaciones web y juegos, y mucho más. JavaScript es un lenguaje de programación del lado del cliente, lo que significa que se ejecuta en el navegador del usuario, en lugar de en el servidor web. JavaScript es un lenguaje de programación de alto nivel, interpretado y orientado a objetos. JavaScript es un lenguaje de programación basado en eventos, lo que significa que puedes ejecutar múltiples tareas simultáneamente sin bloquear el hilo principal. JavaScript es un lenguaje de programación basado en eventos, lo que significa que puedes ejecutar múltiples tareas simultáneamente sin bloquear el hilo principal. JavaScript es un lenguaje de programación basado en eventos, lo que significa que puedes ejecutar múltiples tareas simultáneamente sin bloquear el hilo principal.

72 Referencias

- Codecademy
- MDN Web Docs
- W3Schools
- Node.js
- W3Schools Node.js
- MDN Web Docs Node.js

73 Nodejs

73.1 ¿Qué es nodejs?

Node.js es un entorno de ejecución para JavaScript construido con el motor de JavaScript V8 de Chrome. Node.js usa un modelo de operaciones E/S sin bloqueo y orientado a eventos, que lo hace liviano y eficiente. El ecosistema de paquetes de Node.js, npm, es el ecosistema mas grande de librerías de código abierto en el mundo.

73.2 ¿Por qué nodejs?

Node.js es un entorno de ejecución para JavaScript construido con el motor de JavaScript V8 de Chrome. Node.js usa un modelo de operaciones E/S sin bloqueo y orientado a eventos, que lo hace liviano y eficiente. El ecosistema de paquetes de Node.js, npm, es el ecosistema mas grande de librerías de código abierto en el mundo.

73.3 ¿Cómo instalar nodejs?

Para instalar nodejs en windows debemos descargar el instalador desde la página oficial de nodejs nodejs.org, una vez descargado el instalador lo ejecutamos y seguimos los pasos del instalador.

Para instalar nodejs en linux debemos abrir una terminal y ejecutar los siguientes comandos:

```
sudo apt-get update
sudo apt-get install nodejs
sudo apt-get install npm
```

73.4 ¿Cómo ejecutar un archivo de nodejs?

Para ejecutar un archivo de nodejs debemos abrir una terminal y ejecutar el siguiente comando:

```
node archivo.js
```

73.5 ¿Qué es npm?

npm es el sistema de gestión de paquetes por defecto para Node.js, un entorno de ejecución para JavaScript. npm es un acrónimo de Node Package Manager.

73.6 ¿Cómo instalar un paquete de npm?

Para instalar un paquete de npm debemos abrir una terminal y ejecutar el siguiente comando:

```
npm install nombre-del-paquete
```

73.7 ¿Cómo crear un proyecto de nodejs?

Para crear un proyecto de nodejs debemos abrir una terminal y ejecutar los siguientes comandos:

```
mkdir nombre-del-proyecto
cd nombre-del-proyecto
npm init -y
```

73.8 ¿Cómo instalar express?

Para instalar express debemos abrir una terminal y ejecutar el siguiente comando:

```
npm install express
```

73.9 ¿Cómo instalar nodemon?

Para instalar nodemon debemos abrir una terminal y ejecutar el siguiente comando:

```
npm install nodemon
```

73.10 ¿Cómo instalar mongoose?

Para instalar mongoose debemos abrir una terminal y ejecutar el siguiente comando:

```
npm install mongoose
```

73.11 ¿Cómo instalar dotenv?

Para instalar dotenv debemos abrir una terminal y ejecutar el siguiente comando:

```
npm install dotenv
```

73.12 ¿Cómo instalar bcrypt?

Para instalar bcrypt debemos abrir una terminal y ejecutar el siguiente comando:

```
npm install bcrypt
```

73.13 ¿Cómo instalar jsonwebtoken?

Para instalar jsonwebtoken debemos abrir una terminal y ejecutar el siguiente comando:

```
npm install jsonwebtoken
```

73.14 ¿Cómo instalar cors?

Para instalar cors debemos abrir una terminal y ejecutar el siguiente comando:

```
npm install cors
```

Ahora que ya sabes como instalar los paquetes necesarios para crear un proyecto de nodejs, puedes empezar a crear tu proyecto.

Con Nodejs podemos trabajar en proyectos de frontend con librerías populares como React, Angular, Vue, etc. y también podemos trabajar en proyectos de backend con librerías populares como Express, Mongoose, etc.

73.15 ¿Cómo empiezo mis primeros pasos con React?

Para trabajar con React lo podemos hacer de la forma tradicional que ya no es tan recomandada o utilziar vite que es una herramienta que nos permite trabajar con React de una forma más rápida y sencilla.

73.16 ¿Qué es vite?

Vite es un nuevo compilador de frontend que permite trabajar con React de una forma más rápida y sencilla.

Esto lo veremos en el siguiente capítulo.

73.17 Conclusiones

Nodejs es un entorno de ejecución para JavaScript construido con el motor de JavaScript V8 de Chrome. Node.js usa un modelo de operaciones E/S sin bloqueo y orientado a eventos, que lo hace liviano y eficiente. El ecosistema de paquetes de Node.js, npm, es el ecosistema mas grande de librerías de código abierto en el mundo.

73.18 Referencias

nodejs.org

npm

express

nodemon

mongoose

dotenv

bcrypt

jsonwebtoken

cors

vite

react

angular

vue

74 Introducción a React

Figure 74.1: React

Antes de conocer React, es importante entender que es una librería de JavaScript que se utiliza para construir interfaces de usuario. React fue desarrollado por Facebook y lanzado en 2013. React es una librería de código abierto y se utiliza para construir interfaces de usuario en aplicaciones web. React es una de las librerías de JavaScript más populares y se utiliza en aplicaciones web de todo tipo.

La forma clásica de instalar React es a través de npm, el gestor de paquetes de Node.js. Para instalar React, primero debes instalar Node.js en tu sistema. Luego, puedes instalar React utilizando el siguiente comando:

```
npm install react
```

Sin embargo en la actualidad no se recomienda hacerlo de esta forma, es mejor utilizar Vite, un entorno de desarrollo que permite trabajar con React de forma más sencilla.

74.1 ¿Qué es Vite?

Figure 74.2: Vite + React

Vite es un entorno de desarrollo que permite trabajar con React de forma más sencilla. Vite es un entorno de desarrollo que se basa en la tecnología de ES Modules, que es una forma de importar y exportar módulos en JavaScript. Vite utiliza ES Modules para cargar los módulos de JavaScript de forma rápida y eficiente. Vite también utiliza un servidor de desarrollo que permite trabajar con React de forma más sencilla.

La documentación oficial de Vite se encuentra en el siguiente enlace: https://vitejs.dev/

74.2 Crear un proyecto con Vite

Para crear un proyecto con Vite, puedes utilizar el siguiente comando:

```
npm create vite@latest
```

Este comando crea un proyecto con Vite y te guía a través de la configuración del proyecto. Puedes elegir el nombre del proyecto, el tipo de proyecto (React, Vue, Vanilla), y otras opciones de configuración.

```
npm install
```

74.3 Entendiendo React

74.3.1 src/App.jsx

Antes de correr un servidor de pruebas podemos entender la estructura de React, en el archivo App.jsx ubicado en la ruta **src/App.jsx** se encuentra el código principal de la aplicación, en este archivo se importa React y ReactDOM, se crea un componente de React llamado **App** y se renderiza en el DOM utilizando **ReactDOM.render**.

74.3.2 src/main.jsx

En el archivo **src/main.jsx** se importa el componente **App** y se renderiza en el elemento con el id **root** en el DOM.

Por ahora no serán necesarios los archivo **App.css** y **index.css**. Asi que puedes eliminarlos. Para correr la aplicación puedes utilizar el siguiente comando:

```
npm run dev
```

74.4 Hello World con React

Para crear un Hello World con React, puedes utilizar el siguiente código en el archivo **src/App.jsx**:

```
function App() {
  return (<h1>Hello World in React</h1>)
}

export default App
```

Este código crea un componente de React llamado **App** que renderiza el texto "Hello World" en la pantalla.

Para crear el hello world tambien vamos a modificar el archivo **src/main.jsx** para que renderice el componente **App** en el elemento con el id **root** en el DOM.

```
function App() {
  return (<h1>Hello World in React</h1>)
}

export default App
```

Este código crea un componente de React llamado **App** que renderiza el texto "Hello World" en la pantalla.

Para ejecutar la aplicación, puedes utilizar el siguiente comando:

```
npm run dev
```

Este comando inicia el servidor de desarrollo de Vite y abre la aplicación en tu navegador. Verás el texto “Hello World” en la pantalla.

Figure 74.3: Django Framework

74.5 Conclusiones

En este capítulo aprendimos a crear un proyecto con Vite y React, entendimos la estructura de un proyecto de React y creamos un Hello World con React. En el próximo capítulo aprenderemos a crear componentes en React.

74.6 Ejericios

1. Crea un nuevo proyecto con Vite y React.
2. Crea un componente de React que renderice un texto en la pantalla.

Ver respuesta

Solución ejercicio 1:

```
npm create vite@latest
```

Solución ejercicio 2:

```
function App() {
  return (<h1>Este es un componente de React</h1>)
}

export default App
```

75 Primeros pasos con React

Antes de continuar con la parte práctica es necesario repasar algunos conceptos básicos de React.

75.1 ¿Qué es React?

React es una librería de JavaScript para construir interfaces de usuario. Fue desarrollada por Facebook y lanzada en 2013. React es una de las librerías más populares para construir interfaces de usuario en la actualidad.

75.2 Componentes

En React, todo es un componente. Un componente es una pieza de la interfaz de usuario que puede ser reutilizada en diferentes partes de la aplicación. Los componentes pueden ser simples o complejos, y pueden contener otros componentes.

75.3 JSX

JSX es una extensión de JavaScript que permite escribir código HTML dentro de JavaScript. JSX es una de las características más importantes de React, ya que permite escribir componentes de una forma más sencilla y legible.

75.4 Props

Las props son los argumentos que se pasan a un componente. Las props son inmutables, lo que significa que no pueden ser modificadas por el componente que las recibe.

75.5 State

El state es un objeto que contiene los datos de un componente. El state es mutable, lo que significa que puede ser modificado por el componente que lo contiene.

75.6 Ciclo de vida

Los componentes de React tienen un ciclo de vida que consta de diferentes fases. Algunas de las fases más importantes son:

- **componentDidMount**: Se ejecuta después de que el componente ha sido montado en el DOM.
- **componentDidUpdate**: Se ejecuta después de que el componente ha sido actualizado.
- **componentWillUnmount**: Se ejecuta antes de que el componente sea desmontado del DOM.

75.7 Hooks

Los hooks son una característica de React que permite añadir estado y otras características a los componentes funcionales. Algunos de los hooks más comunes son:

- **useState**: Permite añadir estado a los componentes funcionales.
- **useEffect**: Permite añadir efectos secundarios a los componentes funcionales.
- **useContext**: Permite acceder al contexto de un componente.

75.8 Context

El contexto es una característica de React que permite pasar datos a través del árbol de componentes sin tener que pasar props manualmente en cada nivel.

75.9 Router

El router es una característica de React que permite gestionar la navegación entre diferentes páginas de la aplicación.

75.10 Redux

Redux es una librería de JavaScript para gestionar el estado de la aplicación de una forma predecible y escalable. Redux se basa en tres principios fundamentales: un único origen de verdad, solo lectura y cambios mediante acciones.

75.11 Práctica

Ahora que hemos repasado los conceptos básicos de React, vamos a crear una aplicación sencilla para poner en práctica lo aprendido. En esta aplicación vamos a crear un contador que permita incrementar y decrementar un número.

Para crear la aplicación vamos a utilizar como aprendimos en la sección anterior Vite + React.

75.11.1 Crear la aplicación

Para crear la aplicación vamos a utilizar el siguiente comando:

```
npm create vite@latest
```

Luego seleccionamos la opción **react** y el nombre de la aplicación.

75.11.2 Crear el componente del contador

Para organizar de mejor forma el código vamos a crear el directorio **components** y dentro de este directorio vamos a crear el archivo **Counter.js** con el siguiente contenido:

```
import React, { useState } from 'react';

const Counter = () => {
  const [count, setCount] = useState(0);

  const increment = () => {
    setCount(count + 1);
  };

  const decrement = () => {
    setCount(count - 1);
  };

  return (
    <div>
      <h1>{count}</h1>
      <button onClick={increment}>Increment</button>
      <button onClick={decrement}>Decrement</button>
    </div>
  );
};

export default Counter;
```

75.11.3 Crear el componente principal

Para organizar de mejor forma el código vamos a crear el directorio **pages** y dentro de este directorio vamos a crear el archivo **Home.js** con el siguiente contenido:

```
import React from 'react';
import Counter from '../components/Counter';

const Home = () => {
  return (
    <div>
      <h1>Counter App</h1>
      <Counter />
    </div>
  );
};

export default Home;
```

> Tip
>
> Podemos utilizar la opción de exportación por defecto agregando **export default** al componente cuando lo creamos o podemos exportar el componente al final del archivo utilizando **export**.

75.11.4 Crear la aplicación principal

Modificamos el archivo **src/App.js** con el siguiente contenido:

```
import React from 'react';
import Home from './pages/Home';

const App = () => {
  return (
    <div>
      <Home />
    </div>
  );
};

export default App;
```

75.11.5 Ejecutar la aplicación

Para ejecutar la aplicación vamos a utilizar el siguiente comando:

```
npm run dev
```

75.11.6 Resultado

Si todo ha salido bien, deberíamos ver la aplicación en el navegador con el contador funcionando correctamente.

Figure 75.1: App Counter

75.12 Conclusiones

En este capítulo aprendimos los conceptos básicos de React y cómo crear una aplicación sencilla utilizando Vite + React. En el próximo capítulo vamos a aprender cómo trabajar con componentes en React y cómo organizar el código de una aplicación de forma eficiente.

75.13 Ejercicios

1. Modificar el contador para que no pueda ser menor que cero.
2. Crear un nuevo componente que muestre un mensaje si el contador es mayor que cero.

Ver solución

75.13.1 Solución ejercicio 1

Para modificar el contador para que no pueda ser menor que cero, podemos modificar la función **decrement** de la siguiente forma:

```
const decrement = () => {
  if (count > 0) {
    setCount(count - 1);
  }
};
```

75.13.2 Solución ejercicio 2

Para crear un nuevo componente que muestre un mensaje si el contador es mayor que cero, podemos crear un nuevo componente llamado **Message.js** con el siguiente contenido:

```
import React from 'react';

const Message = ({ count }) => {
  return count > 0 ? <h2>El contador es mayor que cero</h2> : null;
};

export default Message;
```

Luego podemos importar y utilizar este componente en el componente **Home**:

```
import React from 'react';
import Counter from '../components/Counter';
import Message from '../components/Message';

const Home = () => {
  return (
    <div>
      <h1>Counter App</h1>
      <Counter />
      <Message />
    </div>
  );
};

export default Home;
```

Part VII

Ejercicios

76 Ejercicios de Git y Github

76.0.1 Ejercicio 1

1. Crear un repositorio en Github
2. Clonar el repositorio en tu computadora
3. Crear un archivo de texto con tu nombre y subirlo al repositorio
4. Hacer un commit con el mensaje "Agrego mi nombre"
5. Hacer un push al repositorio

Respuesta:

```
git clone [url del repositorio]
cd [nombre del repositorio]
echo "Mi nombre es: [Tu nombre]" > nombre.txt
git add nombre.txt
git commit -m "Agrego mi nombre"
git push origin master
```

76.0.2 Ejercicio 2

1. Crear un repositorio en Github
2. Clonar el repositorio en tu computadora
3. Crear un archivo de python que imprima tu nombre
4. Hacer un commit con el mensaje "Agrego archivo de python"
5. Hacer un push al repositorio

Respuesta:

```
git clone [url del repositorio]
cd [nombre del repositorio]
echo "print('Mi nombre es: [Tu nombre]')" > nombre.py
git add nombre.py
git commit -m "Agrego archivo de python"
git push origin master
```

76.0.3 Ejercicio 3

1. Crear un repositorio en Github
2. Clonar el repositorio en tu computadora
3. Crear un archivo de python que imprima un saludo de bienvenida

4. Hacer un commit con el mensaje “Agrego saludo de bienvenida”
5. Hacer un push al repositorio

Respuesta:

```
git clone [url del repositorio]
cd [nombre del repositorio]
echo "print('Hola, bienvenido')" > saludo.py
git add saludo.py
git commit -m "Agrego saludo de bienvenida"
git push origin master
```

76.0.4 Ejercicio 4

1. Crear un repositorio en Github
2. Clonar el repositorio en tu computadora
3. Crear un archivo de python que imprima un saludo de despedida
4. Hacer un commit con el mensaje “Agrego saludo de despedida”
5. Hacer un push al repositorio

Respuesta:

```
git clone [url del repositorio]
cd [nombre del repositorio]
echo "print('Adios, hasta luego')" > despedida.py
git add despedida.py
git commit -m "Agrego saludo de despedida"
git push origin master
```

76.0.5 Ejercicio 5

1. Crear un repositorio en Github
2. Clonar el repositorio en tu computadora
3. Crear un archivo de python que imprima un saludo de bienvenida y un saludo de despedida
4. Hacer un commit con el mensaje “Agrego saludo de bienvenida y despedida”
5. Hacer un push al repositorio

Respuesta:

```
git clone [url del repositorio]
cd [nombre del repositorio]
echo "print('Hola, bienvenido')" > saludo.py
echo "print('Adios, hasta luego')" > despedida.py
git add saludo.py despedida.py
git commit -m "Agrego saludo de bienvenida y despedida"
git push origin master
```

77 Ejercicios Python - Nivel 1

77.1 Ejercicio 1

- Crear un programa que muestre por pantalla la cadena “Hola Mundo!”.

Solución

```
print("Hola Mundo!")
```

77.2 Ejercicio 2

- Crear un programa que muestre por pantalla tu nombre.

Solución

```
print("Tu nombre")
```

77.3 Ejercicio 3

- Crear un programa que pida al usuario que introduzca su nombre y muestre por pantalla la cadena “Hola”, seguido del nombre y un signo de exclamación.

Solución

```
nombre = input("Introduce tu nombre: ")
print("Hola", nombre, "!")
```

Otra forma de hacerlo:

```
nombre = input("Introduce tu nombre: ")
print(f"Hola {nombre}!")
```

77.4 Ejercicio 4

- Crear un programa que pregunte al usuario por el número de horas trabajadas y el coste por hora. Después debe mostrar por pantalla la paga que le corresponde.

Solución

```
horas = float(input("Introduce tus horas de trabajo: "))
coste = float(input("Introduce lo que cobras por hora: "))
paga = horas * coste
print("Tu paga es de", paga)
```

77.5 Ejercicio 5

- Crear un programa que pida al usuario una cantidad de dolares, una tasa de interés y un número de años. Mostrar por pantalla en cuanto se habrá convertido el capital inicial transcurridos esos años si cada año se aplica la tasa de interés introducida.
- Formula del interés compuesto: Cn = C * (1 + x/100) ^ n

Solución

```
cantidad = float(input("¿Cantidad a invertir? "))
interes = float(input("¿Interés porcentual anual? "))
años = int(input("¿Años?"))
print("Capital final: ", round(cantidad * (interes / 100 + 1) ** años, 2))
```

78 Ejercicios Python - Nivel 2

78.1 Ejercicio 1

- Crear una función que reciba una lista de números y devuelva su media aritmética.
- Ejemplo: media_aritmetica([1, 2, 3, 4, 5]) -> 3.0

Posible solución

```
def media_aritmetica(lista):
    return sum(lista) / len(lista)
```

78.2 Ejercicio 2

- Crear una función que reciba una lista de números y devuelva su mediana.
- Ejemplo: mediana([1, 2, 3, 4, 5]) -> 3.0

Posible solución

```
def mediana(lista):
    lista_ordenada = sorted(lista)
    n = len(lista_ordenada)
    if n % 2 == 0:
        return (lista_ordenada[n // 2 - 1] + lista_ordenada[n // 2]) / 2
    else:
        return lista_ordenada[n // 2]
```

78.3 Ejercicio 3

- Crear una función que reciba una lista de números y devuelva su moda.
- Si hay más de una moda, devolver una lista con todas las modas.
- Si no hay moda, devolver una lista vacía.
- La moda es el número que más veces se repite en una lista.
- Si todos los números se repiten el mismo número de veces, no hay moda.
- Ejemplo: moda([1, 2, 3, 2, 3, 4]) -> [2, 3]

Posible solución

```
def moda(lista):
    frecuencias = {}
    for numero in lista:
        if numero in frecuencias:
            frecuencias[numero] += 1
        else:
            frecuencias[numero] = 1
    max_frecuencia = max(frecuencias.values())
    modas = [numero for numero, frecuencia in frecuencias.items() if frecuencia == max_fr
    return modas if len(modas) > 1 else modas[0] if modas else []
```

78.4 Ejercicio 4

- Crear una función que reciba una lista de números y devuelva su desviación típica.
- La desviación típica es la raíz cuadrada de la varianza.
- La varianza es la media de los cuadrados de las diferencias entre cada número y la media aritmética.
- Ejemplo: desviacion_tipica([1, 2, 3, 4, 5]) -> 1.4142135623730951

Posible solución

```
def desviacion_tipica(lista):
    media = sum(lista) / len(lista)
    varianza = sum((numero - media) ** 2 for numero in lista) / len(lista)
    return varianza ** 0.5
```

78.5 Ejercicio 5

- Crear una función que reciba una lista de números y devuelva su coeficiente de variación.
- El coeficiente de variación es la desviación típica dividida por la media aritmética.
- Ejemplo: coeficiente_variacion([1, 2, 3, 4, 5]) -> 0.4472135954999579

Posible solución

```
def coeficiente_variacion(lista):
    media = sum(lista) / len(lista)
    desviacion_tipica = sum((numero - media) ** 2 for numero in lista) / len(lista) ** 0.
    return desviacion_tipica / media
```

79 Ejercicios Python - Nivel 3

79.1 Ejercicio 1:

- Crear una lista vacía y agregar elementos a la misma hasta que el usuario ingrese "fin".

Posible solución

```
lista = []
while True:
    elemento = input("Ingrese un elemento: ")
    if elemento == "fin":
        break
    lista.append(elemento)
print(lista)
```

79.2 Ejercicio 2:

- Crear una lista con los números del 1 al 10 y mostrar los números pares.

Posible solución

```
lista = list(range(1, 11))
for numero in lista:
    if numero % 2 == 0:
        print(numero)
```

79.3 Ejercicio 3:

- Crear una lista con los números del 1 al 10 y mostrar los números impares.

Posible solución

```
lista = list(range(1, 11))
for numero in lista:
    if numero % 2 != 0:
        print(numero)
```

79.4 Ejercicio 4:

- Crear una lista de nombres de estudiantes y mostrar aquellos cuyos nombres comiencen con la letra "A".

Posible solución

```
nombres = ["Ana", "Juan", "Pedro", "Andrea", "Lucía", "Antonio"]
for nombre in nombres:
    if nombre[0].lower() == "a":
        print(nombre)
```

79.5 Ejercicio 5:

- Crear una lista de números y mostrar aquellos que sean mayores a 100.

Posible solución

```
numeros = [10, 20, 150, 200, 300, 400, 500]
for numero in numeros:
    if numero > 100:
        print(numero)
```

80 Ejercicios Python - Nivel 4

80.1 Ejercicio 1:

- Crear un conjunto vacío y agregar elementos al mismo hasta que el usuario ingrese "fin".

Posible solución

```
conjunto = set()
while True:
    elemento = input("Ingrese un elemento o 'fin' para terminar: ")
    if elemento == "fin":
        break
    conjunto.add(elemento)
print(conjunto)
```

80.2 Ejercicio 2:

- Crear un diccionario vacío y agregar elementos al mismo hasta que el usuario ingrese "fin".
- Cada elemento debe ser una tupla con dos elementos, el primero será la clave y el segundo el valor.
- Si el usuario ingresa una clave que ya existe, se debe mostrar un mensaje de error y no agregar el elemento.

Posible solución

```
diccionario = dict()
while True:
    clave = input("Ingrese una clave o 'fin' para terminar: ")
    if clave == "fin":
        break
    if clave in diccionario:
        print("La clave ya existe")
        continue
    valor = input("Ingrese un valor: ")
    diccionario[clave] = valor
print(diccionario)
```

80.3 Ejercicio 3:

- Crear un diccionario con los nombres de los meses como claves y la cantidad de días que tienen como valor.
- Mostrar los meses que tienen 31 días.
- Mostrar los meses que tienen 30 días.
- Mostrar los meses que tienen 28 días.

Posible solución

```
meses = {
    "enero": 31,
    "febrero": 28,
    "marzo": 31,
    "abril": 30,
    "mayo": 31,
    "junio": 30,
    "julio": 31,
    "agosto": 31,
    "septiembre": 30,
    "octubre": 31,
    "noviembre": 30,
    "diciembre": 31
}

meses_31 = [mes for mes, dias in meses.items() if dias == 31]
meses_30 = [mes for mes, dias in meses.items() if dias == 30]
meses_28 = [mes for mes, dias in meses.items() if dias == 28]

print("Meses con 31 días:", meses_31)
print("Meses con 30 días:", meses_30)
print("Meses con 28 días:", meses_28)
```

80.4 Ejercicio 4:

- Crear un diccionario con los nombres de los meses como claves y la cantidad de paises de sur america mostrando mediante el pais la capital.
- Mostrar la capital de Argentina.
- Mostrar la capital de Brasil.
- Mostrar la capital de Ecuador.

Posible solución

```
paises = {
    "Argentina": "Buenos Aires",
    "Bolivia": "La Paz",
    "Brasil": "Brasilia",
```

```
    "Chile": "Santiago",
    "Colombia": "Bogotá",
    "Ecuador": "Quito",
    "Guyana": "Georgetown",
    "Paraguay": "Asunción",
    "Perú": "Lima",
    "Surinam": "Paramaribo",
    "Uruguay": "Montevideo",
    "Venezuela": "Caracas"
}

print("La capital de Argentina es", paises["Argentina"])
print("La capital de Brasil es", paises["Brasil"])
print("La capital de Ecuador es", paises["Ecuador"])
```

80.5 Ejercicio 5:

- Crear un diccinario con los nombres de los presidentes de Ecuador y la fecha en la que asumieron el cargo.
- Mostrar la fecha en la que asumió el presidente Eloy Alfaro.
- Mostrar la fecha en la que asumió el presidente Garcia Moreno.

Posible solución

```
presidentes = {

    "Gustavo Noboa": "23 de noviembre de 2023",
    "Guillermo Lasso": "24 de mayo de 2021",
    "Lenín Moreno": "24 de mayo de 2017",
    "Rafael Correa": "15 de enero de 2007",
    "Jamil Mahuad": "10 de agosto de 1998",
    "Abdalá Bucaram": "10 de agosto de 1996",
    "Sixto Durán Ballén": "10 de agosto de 1992",
    "Rodrigo Borja": "10 de agosto de 1988",
    "León Febres Cordero": "10 de agosto de 1984",
    "Osvaldo Hurtado": "10 de agosto de 1981",
    "Jaime Roldós": "10 de agosto de 1979",
    "Guillermo Rodríguez": "24 de mayo de 1972",
    "José María Velasco Ibarra": "1 de septiembre de 1968",
    "Otto Arosemena": "16 de febrero de 1966",
    "Carlos Julio Arosemena": "1 de septiembre de 1961",
    "Camilo Ponce Enríquez": "1 de septiembre de 1956",
    "José María Velasco Ibarra": "1 de septiembre de 1952",
    "Galalza Castro": "1 de septiembre de 1947",
    "Carlos Arroyo del Río": "1 de septiembre de 1940",
    "Andrés Córdova": "1 de septiembre de 1938",
```

```
    "Alberto Enríquez Gallo": "1 de septiembre de 1937",
    "Federico Páez": "1 de septiembre de 1935",
    "José María Velasco Ibarra": "1 de septiembre de 1934",
    "Abelardo Montalvo": "1 de septiembre de 1933",
    "Neptalí Bonifaz": "1 de septiembre de 1931",
    "Isidro Ayora": "1 de septiembre de 1926",
    "Gonzalo Córdova": "1 de septiembre de 1924",
    "José Luis Tamayo": "1 de septiembre de 1920",
    "Leónidas Plaza": "1 de septiembre de 1912",
    "Emilio Estrada": "1 de septiembre de 1911",
    "Carlos Freile Zaldumbide": "1 de septiembre de 1907",
    "Eloy Alfaro": "1 de septiembre de 1906",
    "Leónidas Plaza": "1 de septiembre de 1901",
    "Eloy Alfaro": "1 de septiembre de 1897",
    "Antonio Flores Jijón": "1 de septiembre de 1888",
    "José Plácido Caamaño": "1 de septiembre de 1883",
    "Pedro José de Arteta": "1 de septiembre de 1882",
    "Francisco Xavier León": "1 de septiembre de 1878",
    "Antonio Borrero": "1 de septiembre de 1875",
    "Gabriel García Moreno": "1 de septiembre de 1861",
    "Francisco Robles": "1 de septiembre de 1856",
    "Diego Noboa": "1 de septiembre de 1850",
    "José Joaquín de Olmedo": "1 de septiembre de 1845",
    "Juan José Flores": "1 de septiembre de 1830",
}

print("Eloy Alfaro asumió el", presidentes["Eloy Alfaro"])
print("García Moreno asumió el", presidentes["Gabriel García Moreno"])
```