

# **Django y React 2024**

Diego Saavedra

Jul 25, 2024

# Table of contents

<b>1 Bienvenido</b>	<b>12</b>
1.1 ¿De qué trata este curso? . . . . .	12
1.2 ¿Para quién es este curso? . . . . .	12
1.3 ¿Cómo contribuir? . . . . .	12
<b>I Unidad 1: Introducción a Python</b>	<b>14</b>
<b>2 Git y GitHub</b>	<b>15</b>
2.1 ¿Qué es Git y GitHub? . . . . .	15
2.2 ¿Quiénes utilizan Git? . . . . .	16
2.3 ¿Cómo se utiliza Git? . . . . .	16
2.4 ¿Para qué sirve Git? . . . . .	17
2.5 ¿Por qué utilizar Git? . . . . .	18
2.6 ¿Dónde puedo utilizar Git? . . . . .	19
2.7 Pasos Básicos . . . . .	19
2.8 Instalación de Visual Studio Code . . . . .	20
2.8.1 Descarga e Instalación de Git . . . . .	21
2.8.2 Configuración . . . . .	22
2.8.3 Creación de un Repositorio “helloWorld” en Python . . . . .	22
2.8.4 Comandos Básicos de Git . . . . .	23
2.8.5 Estados en Git . . . . .	23
<b>3 Tutorial: Moviendo Cambios entre Estados en Git</b>	<b>24</b>
3.1 Introducción . . . . .	24
3.2 Sección 1: Modificar Archivos en el Repositorio . . . . .	24
3.3 Mover Cambios de Local a Staging: . . . . .	24
3.4 Agregar Cambios de Local a Staging: . . . . .	25
3.5 Sección 2: Confirmar Cambios en un Commit . . . . .	25
3.6 Mover Cambios de Staging a Commit: . . . . .	25
3.7 Sección 3: Creación y Fusión de Ramas . . . . .	25
3.8 Crear una Nueva Rama: . . . . .	25
3.9 Implementar Funcionalidades en la Rama: . . . . .	25
3.10 Fusionar Ramas con la Rama Principal: . . . . .	26
3.11 Sección 4: Revertir Cambios en un Archivo . . . . .	26
3.12 Revertir Cambios en un Archivo: . . . . .	26
3.13 Conclusión . . . . .	26
<b>4 Asignación</b>	<b>27</b>

<b>5 GitHub Classroom</b>	<b>28</b>
5.1 ¿Qué es GitHub Classroom? . . . . .	28
5.1.1 Funcionalidades Principales . . . . .	28
5.2 Ejemplo Práctico . . . . .	29
5.2.1 Creación de una Asignación en GitHub Classroom . . . . .	29
5.3 Trabajo de los Estudiantes . . . . .	31
<b>6 Docker</b>	<b>37</b>
<b>7 Conceptos Básicos de Docker</b>	<b>38</b>
7.1 Imagen . . . . .	38
7.2 Contenedor . . . . .	38
7.3 Dockerfile . . . . .	38
7.4 Docker Compose . . . . .	39
<b>8 Uso de Docker</b>	<b>40</b>
8.1 Definir un Dockerfile . . . . .	40
8.2 Construir la Imagen . . . . .	40
8.3 Ejecutar un Contenedor . . . . .	41
8.4 Gestionar Contenedores . . . . .	41
8.5 Docker Compose . . . . .	41
<b>II Unidad 2: Python Básico</b>	<b>43</b>
<b>9 Hola Mundo en Python</b>	<b>44</b>
<b>10 Introducción</b>	<b>45</b>
10.0.1 Paso 1: Instalación de Python . . . . .	45
10.0.2 Paso 2: Instalación de Python en Windows . . . . .	45
10.0.3 Paso 3: Crear nuestro primer “Hola Mundo” en Python . . . . .	48
<b>11 Sintaxis Básica</b>	<b>51</b>
<b>12 Comentarios</b>	<b>52</b>
<b>13 Variables y Tipos de Datos</b>	<b>53</b>
<b>14 Tipos de Datos</b>	<b>54</b>
<b>15 Operadores</b>	<b>56</b>
<b>16 Estructura de Control</b>	<b>60</b>
<b>17 Funciones</b>	<b>65</b>
<b>18 Llamada a Funciones</b>	<b>66</b>
<b>19 Parámetros y Argumentos</b>	<b>67</b>
<b>20 Retorno</b>	<b>68</b>

<b>21 Ejemplo</b>	<b>69</b>
<b>22 Asignación</b>	<b>71</b>
22.1 Objetivo . . . . .	71
22.2 ¿Qué debes hacer? . . . . .	71
22.3 Pruebas . . . . .	71
22.4 Ejecución . . . . .	72
<b>III Unidad 3: Python Intermedio</b>	<b>73</b>
<b>23 Listas</b>	<b>74</b>
<b>24 Tuplas</b>	<b>75</b>
<b>25 Manipulación de Listas y Tuplas</b>	<b>76</b>
<b>26 Funciones integradas para Listas y Tuplas</b>	<b>77</b>
<b>27 Listas Anidadas</b>	<b>78</b>
<b>28 Listas y Tuplas como Argumentos de Funciones</b>	<b>79</b>
<b>29 Listas y Tuplas como Retorno de Funciones</b>	<b>80</b>
<b>30 Asignación</b>	<b>81</b>
30.1 Descripción de la Asignación . . . . .	81
30.2 Tarea Pendiente: . . . . .	81
30.3 Cómo Ejecutar el Código . . . . .	81
30.4 Ejemplo de salida: . . . . .	81
<b>31 Diccionarios</b>	<b>83</b>
<b>32 Conjuntos</b>	<b>88</b>
<b>33 Operaciones con Diccionarios y Conjuntos</b>	<b>92</b>
<b>34 Asignación</b>	<b>94</b>
34.1 Descripción . . . . .	94
34.2 Criterios de Evaluación . . . . .	94
<b>IV Unidad 4: Python Avanzado</b>	<b>96</b>
<b>35 Programación Orientada a Objetos</b>	<b>98</b>
35.1 Asignación . . . . .	104
35.2 Instrucciones . . . . .	104
35.3 Contenido del Repositorio . . . . .	104
35.4 Cómo Ejecutar las Pruebas . . . . .	105
<b>36 Módulos</b>	<b>107</b>

<b>37 Paquetes</b>	<b>110</b>
<b>38 Creación y Uso de Módulos</b>	<b>113</b>
38.1 Creación de Módulos . . . . .	113
38.2 Uso de Módulos . . . . .	113
38.3 Importar Funciones Específicas . . . . .	113
38.4 Importar con Alias . . . . .	114
38.5 Importar Todas las Funciones . . . . .	114
<b>39 Creación y Uso de Paquetes</b>	<b>116</b>
39.1 Creación de Paquetes . . . . .	116
39.2 Uso de Paquetes . . . . .	116
39.3 Importar con Alias . . . . .	116
39.4 Importar Todas las Funciones . . . . .	116
<b>40 Asignación Calculadora Pythonica</b>	<b>119</b>
40.1 Instrucciones . . . . .	119
40.2 Contenido del Repositorio . . . . .	119
40.3 Ejercicio . . . . .	119
40.4 Cómo Ejecutar el Programa . . . . .	119
40.5 Cómo Ejecutar las Pruebas . . . . .	120
<b>41 Sistema de Gestión de Inventarios</b>	<b>122</b>
41.1 Asignación . . . . .	122
<b>V Unidad 5: Django</b>	<b>124</b>
<b>42 Introducción a Django</b>	<b>126</b>
42.1 Conceptos Importantes . . . . .	126
42.1.1 Entornos Virtuales . . . . .	127
42.1.2 Modelo Template View (MTV) . . . . .	128
42.1.3 Formularios . . . . .	128
42.1.4 Administrador de Django . . . . .	128
42.1.5 Middleware . . . . .	128
42.1.6 Autenticación y Autorización . . . . .	128
42.1.7 Internacionalización . . . . .	128
42.1.8 Seguridad . . . . .	128
42.1.9 Testing . . . . .	128
42.1.10 Despliegue . . . . .	128
<b>43 Configuración inicial de un proyecto.</b>	<b>128</b>
43.1 1. Crear un entorno virtual . . . . .	128
43.2 2. Activar el entorno virtual . . . . .	128
43.3 3. Instalar Django . . . . .	128
43.4 4. Crear un proyecto de Django . . . . .	128
43.5 5. Crear una aplicación de Django . . . . .	128
43.6 6. Crear una vista . . . . .	128
43.7 7. Configurar las URL . . . . .	128
43.8 8. Ejecutar el servidor de desarrollo . . . . .	128

43.9 9. Crear una migración . . . . .	128
43.1010. Aplicar una migración . . . . .	128
43.1112. Crear un superusuario . . . . .	128
43.1213. Acceder al panel de administración . . . . .	128
<b>44 Ejercicio</b>	<b>128</b>
<b>45 Asignación</b>	<b>128</b>
<b>46 Estructura de archivos y carpetas</b>	<b>128</b>
<b>47 Creación de una aplicación Django</b>	<b>128</b>
<b>48 Configuración de la base de datos</b>	<b>128</b>
<b>49 Crear una vista</b>	<b>128</b>
<b>50 Crear una plantilla</b>	<b>128</b>
<b>51 Configurar las rutas</b>	<b>128</b>
<b>52 Correr el servidor de desarrollo</b>	<b>128</b>
<b>53 Acceder a la aplicación</b>	<b>128</b>
<b>54 Acceder a la aplicación</b>	<b>128</b>
<b>55 Asignación</b>	<b>128</b>
<b>56 Referencias</b>	<b>128</b>
<b>57 Modelos</b>	<b>128</b>
<b>58 Registramos la aplicacion en admin.py</b>	<b>128</b>
<b>59 Vistas en Django</b>	<b>128</b>
59.1 Listar productos . . . . .	128
59.2 Agregar producto . . . . .	128
59.3 Actualizar producto . . . . .	128
59.4 Eliminar producto . . . . .	128
59.5 Buscar producto . . . . .	128
<b>60 Templates</b>	<b>128</b>
60.1 Base . . . . .	128
60.2 Listar . . . . .	128
60.3 Agregar . . . . .	128
60.4 Actualizar . . . . .	128
60.5 Eliminar . . . . .	128
60.6 Buscar . . . . .	128
<b>61 URLs</b>	<b>128</b>
61.1 URLs en la aplicación y el proyecto . . . . .	128

<b>62 Ejecutar el servidor</b>	<b>128</b>
<b>63 Django Rest Framework</b>	<b>128</b>
63.1 Instalación . . . . .	128
63.2 Actualizar el archivo requirements.txt . . . . .	128
63.3 Serializers . . . . .	128
63.4 Views . . . . .	128
63.5 URLs de la Aplicación . . . . .	128
63.6 Configuración URLs del Proyecto . . . . .	128
63.7 Migraciones . . . . .	128
63.8 Ejecución . . . . .	128
<b>64 Documentación de la API con drf-yasg</b>	<b>128</b>
<b>65 Documentación de la API con CoreAPI</b>	<b>128</b>
65.1 Primero instalamos CoreAPI: . . . . .	128
<b>66 Testing</b>	<b>128</b>
<b>67 Corrección de tests en Django Rest Framework</b>	<b>128</b>
67.1 Introducción . . . . .	128
67.2 Pasos . . . . .	128
67.2.1 1. Crear un entorno virtual . . . . .	128
67.2.2 2. Activar el entorno virtual . . . . .	128
67.2.3 3. Instalar las dependencias . . . . .	128
67.2.4 4. Correr los tests . . . . .	128
67.2.5 5. Corregir los tests . . . . .	128
67.2.6 6. Correr los tests nuevamente . . . . .	128
67.2.7 7. Desactivar el entorno virtual . . . . .	128
67.3 Conclusión . . . . .	128
67.4 Referencias . . . . .	128
<b>VI Unidad 6: Frontend</b>	<b>128</b>
<b>68 Introducción a HTML</b>	<b>128</b>
68.1 Estructura básica de un documento HTML . . . . .	128
68.2 Etiquetas HTML . . . . .	128
68.3 Atributos HTML . . . . .	128
68.4 Ejemplo de un documento HTML . . . . .	128
68.5 Recursos adicionales . . . . .	128
68.6 Ejercicios . . . . .	128
<b>69 Introducción a CSS</b>	<b>128</b>
69.1 ¿Qué es CSS? . . . . .	128
69.2 ¿Cómo funciona CSS? . . . . .	128
69.3 ¿Cómo se aplica CSS a un documento HTML? . . . . .	128
69.4 Sintaxis de CSS . . . . .	128
69.5 Ejemplo de CSS . . . . .	128
69.6 Comentarios en CSS . . . . .	128

69.7 Selectores CSS . . . . .	128
69.8 Propiedades CSS . . . . .	128
69.9 Unidades CSS . . . . .	128
<b>70 JavaScript</b>	<b>128</b>
70.1 ¿Qué es JavaScript? . . . . .	128
70.2 ¿Dónde puedo aprender JavaScript? . . . . .	128
70.3 ¿Qué puedo hacer con JavaScript? . . . . .	128
70.4 ¿Qué es un script de JavaScript? . . . . .	128
70.5 ¿Dónde puedo ejecutar JavaScript? . . . . .	128
70.6 ¿Qué es Node.js? . . . . .	128
70.7 ¿Dónde puedo aprender Node.js? . . . . .	128
70.8 ¿Qué puedo hacer con Node.js? . . . . .	128
<b>71 Primeros pasos con JavaScript</b>	<b>128</b>
71.1 Ecmascript 6 . . . . .	128
71.2 Motor V8 . . . . .	128
71.3 Asincronismo . . . . .	128
71.4 Programación Orientada a Objetos . . . . .	128
71.5 Programación Funcional . . . . .	128
71.6 Hola Mundo en JavaScript . . . . .	128
71.7 Comentarios en JavaScript . . . . .	128
71.8 Variables en JavaScript . . . . .	128
71.9 Tipos de datos en JavaScript . . . . .	128
71.10Operadores en JavaScript . . . . .	128
71.11Condicionales en JavaScript . . . . .	128
71.12Bucles en JavaScript . . . . .	128
71.13Funciones en JavaScript . . . . .	128
71.14Eventos en JavaScript . . . . .	128
71.15Objetos en JavaScript . . . . .	128
71.16Arrays en JavaScript . . . . .	128
71.17Métodos en JavaScript . . . . .	128
71.18Clases en JavaScript . . . . .	128
71.19Herencia en JavaScript . . . . .	128
71.20Promesas en JavaScript . . . . .	128
71.21Async/Await en JavaScript . . . . .	128
71.22Consumir una api con fetch . . . . .	128
71.23Crear una aplicación web con JavaScript . . . . .	128
71.24Crear un juego con JavaScript . . . . .	128
71.25Conclusiones . . . . .	128
<b>72 Referencias</b>	<b>128</b>
<b>73 Nodejs</b>	<b>128</b>
73.1 Características de Nodejs . . . . .	128
73.2 Instalación de Nodejs . . . . .	128
73.3 Utilizando nodejs . . . . .	128

<b>74 Lógica de la programación con nodejs</b>	<b>128</b>
74.1 GlobalThis . . . . .	128
74.2 Conclusiones . . . . .	128
74.3 Creación de una aplicación web con Nodejs . . . . .	128
74.4 Conclusiones . . . . .	128
<b>75 Alternativas a Nodejs</b>	<b>128</b>
75.1 Deno . . . . .	128
75.2 Creación de una aplicación web con Deno . . . . .	128
75.3 Bun . . . . .	128
75.4 Creación de una aplicación web con Bun . . . . .	128
75.5 Conclusiones . . . . .	128
<b>76 Npm, Yarn y Pnpm</b>	<b>128</b>
76.1 Introducción . . . . .	128
76.2 Npm . . . . .	128
76.3 Instalación de paquetes con Npm . . . . .	128
76.4 Yarn . . . . .	128
76.5 Instalación de paquetes con Yarn . . . . .	128
76.6 Crear un proyecto con Yarn . . . . .	128
76.7 Pnpm . . . . .	128
76.8 Instalación de paquetes con Pnpm . . . . .	128
76.9 Crear un proyecto con Pnpm . . . . .	128
76.10 Conclusiones . . . . .	128
76.11 FNM . . . . .	128
76.12 Instalación de FNM . . . . .	128
76.13 Ejemplo de uso de FNM . . . . .	128
76.14 Conclusiones . . . . .	128
<b>77 Introducción a React</b>	<b>128</b>
77.1 ¿Qué es Vite? . . . . .	128
77.2 Crear un proyecto con Vite . . . . .	128
77.3 Entendiendo React . . . . .	128
77.3.1 src/App.jsx . . . . .	128
77.3.2 src/main.jsx . . . . .	128
77.4 Hello World con React . . . . .	128
77.5 Conclusiones . . . . .	128
77.6 Ejercicios . . . . .	128
<b>78 Primeros pasos con React</b>	<b>128</b>
78.1 ¿Qué es React? . . . . .	128
78.2 Componentes . . . . .	128
78.3 JSX . . . . .	128
78.4 Props . . . . .	128
78.5 State . . . . .	128
78.6 Ciclo de vida . . . . .	128
78.7 Hooks . . . . .	128
78.8 Context . . . . .	128
78.9 Router . . . . .	128

78.10Redux . . . . .	128
78.11Práctica . . . . .	128
78.11.1 Crear la aplicación . . . . .	128
78.11.2 Crear el componente del contador . . . . .	128
78.11.3 Crear el componente principal . . . . .	128
78.11.4 Crear la aplicación principal . . . . .	128
78.11.5 Ejecutar la aplicación . . . . .	128
78.11.6 Resultado . . . . .	128
78.12Conclusiones . . . . .	128
78.13Ejercicios . . . . .	128
78.13.1 Solución ejercicio 1 . . . . .	128
78.13.2 Solución ejercicio 2 . . . . .	128
<b>79 Axios con React</b>	<b>128</b>
79.1 Crear la aplicación . . . . .	128
79.2 Crear el componente . . . . .	128
79.3 Importar el componente . . . . .	128
79.4 Ejecutar la aplicación . . . . .	128
79.5 Conclusión . . . . .	128
<b>80 Fetch con React para obtener el listado de productos</b>	<b>128</b>
80.1 Crear la aplicación . . . . .	128
80.2 Crear el componente . . . . .	128
80.3 Importar el componente . . . . .	128
80.4 Ejecutar la aplicación . . . . .	128
80.5 Conclusiones . . . . .	128
<b>81 Crud Básico con React y Axios</b>	<b>128</b>
81.1 Crear el proyecto . . . . .	128
81.2 Crear la API . . . . .	128
81.3 Crear el proyecto de React . . . . .	128
81.4 Crear la lista de tareas . . . . .	128
81.5 Mostrar la lista de tareas . . . . .	128
81.6 Probar la aplicación . . . . .	128
81.7 Conclusiones . . . . .	128
<b>VII Ejercicios</b>	<b>128</b>
<b>82 Ejercicios de Git y Github</b>	<b>128</b>
82.0.1 Ejercicio 1 . . . . .	128
82.0.2 Ejercicio 2 . . . . .	128
82.0.3 Ejercicio 3 . . . . .	128
82.0.4 Ejercicio 4 . . . . .	128
82.0.5 Ejercicio 5 . . . . .	128
<b>83 Ejercicios Python - Nivel 1</b>	<b>128</b>
83.1 Ejercicio 1 . . . . .	128
83.2 Ejercicio 2 . . . . .	128

83.3 Ejercicio 3 . . . . .	128
83.4 Ejercicio 4 . . . . .	128
83.5 Ejercicio 5 . . . . .	128
<b>84 Ejercicios Python - Nivel 2</b>	<b>128</b>
84.1 Ejercicio 1 . . . . .	128
84.2 Ejercicio 2 . . . . .	128
84.3 Ejercicio 3 . . . . .	128
84.4 Ejercicio 4 . . . . .	128
84.5 Ejercicio 5 . . . . .	128
<b>85 Ejercicios Python - Nivel 3</b>	<b>128</b>
85.1 Ejercicio 1: . . . . .	128
85.2 Ejercicio 2: . . . . .	128
85.3 Ejercicio 3: . . . . .	128
85.4 Ejercicio 4: . . . . .	128
85.5 Ejercicio 5: . . . . .	128
<b>86 Ejercicios Python - Nivel 4</b>	<b>128</b>
86.1 Ejercicio 1: . . . . .	128
86.2 Ejercicio 2: . . . . .	128
86.3 Ejercicio 3: . . . . .	128
86.4 Ejercicio 4: . . . . .	128
86.5 Ejercicio 5: . . . . .	128

# 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 línea, 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

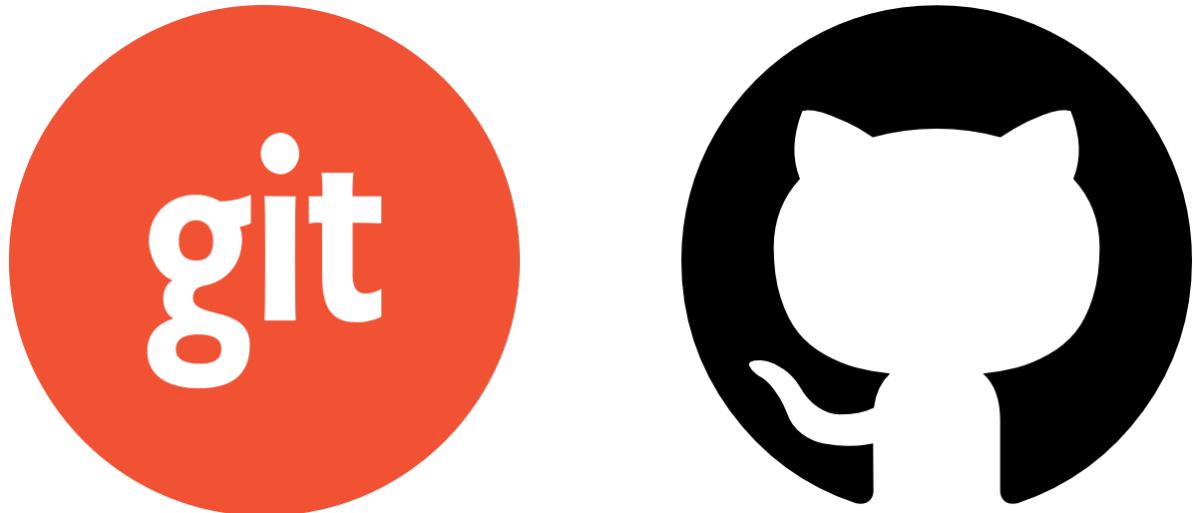


Figure 2.1: Git and Github

### 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 9fa7605ad1837aa44dfb9c711dc8bd60cab7c5d
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?

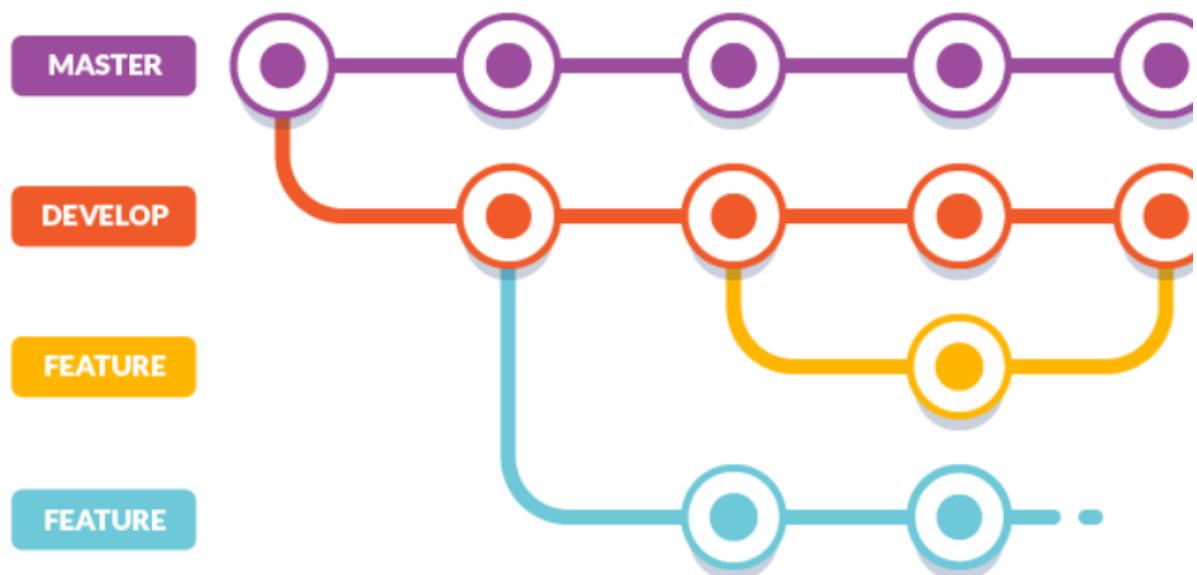


Figure 2.4: Seguimiento de Cambios con Git

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?

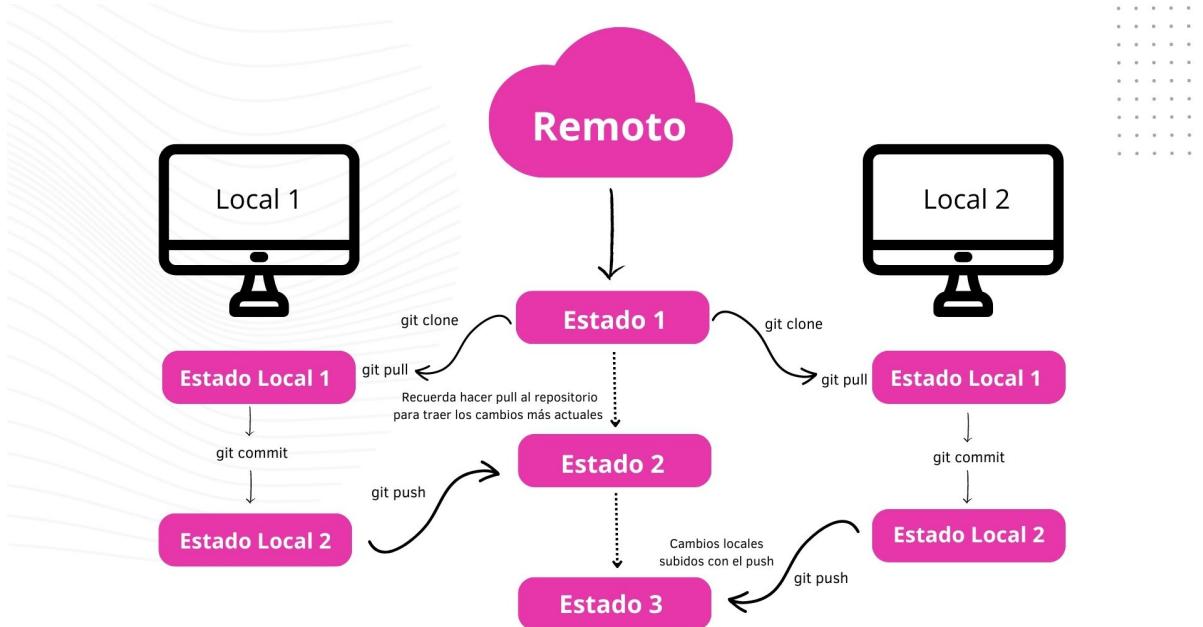


Figure 2.5: Ventajas de Git

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?

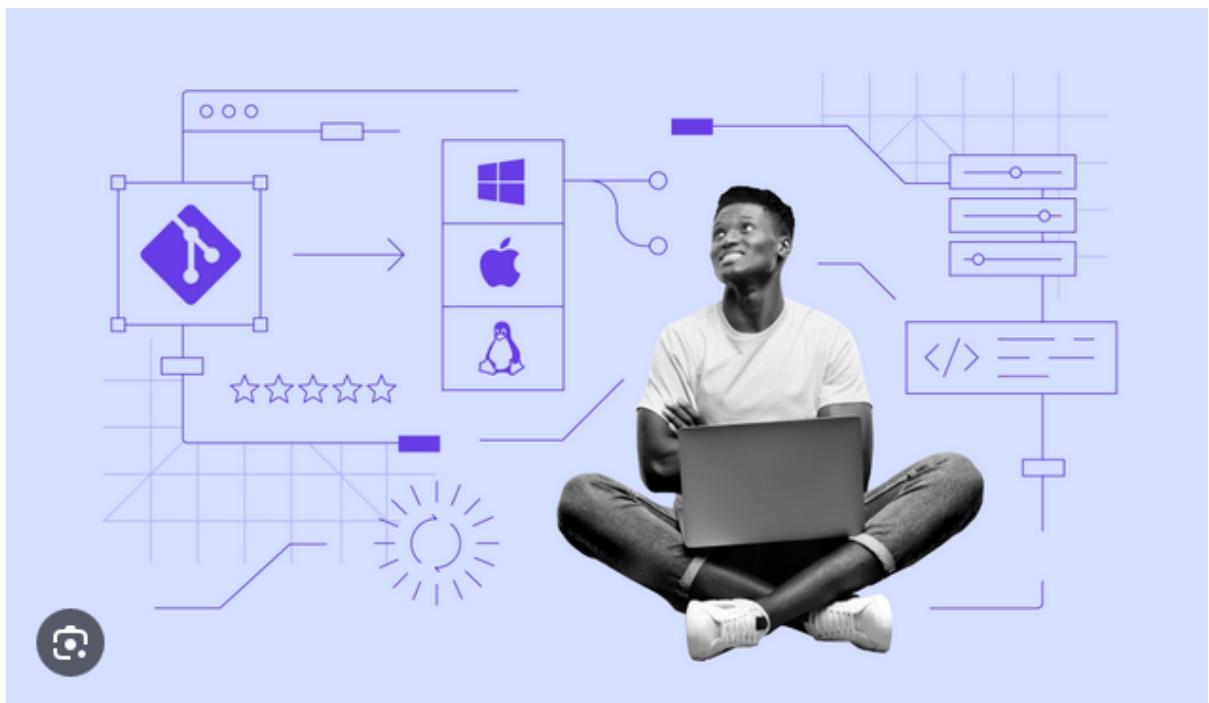


Figure 2.6: Git en Diferentes Sistemas Operativos

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

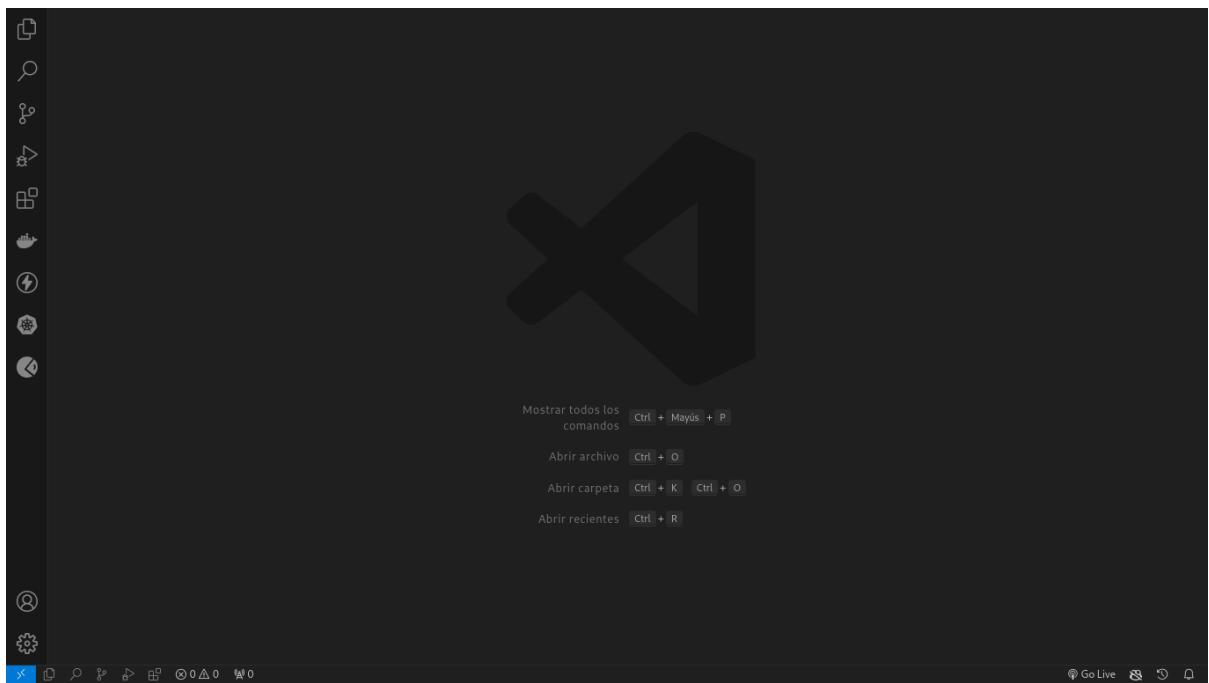


Figure 2.7: Visual Studio Code

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

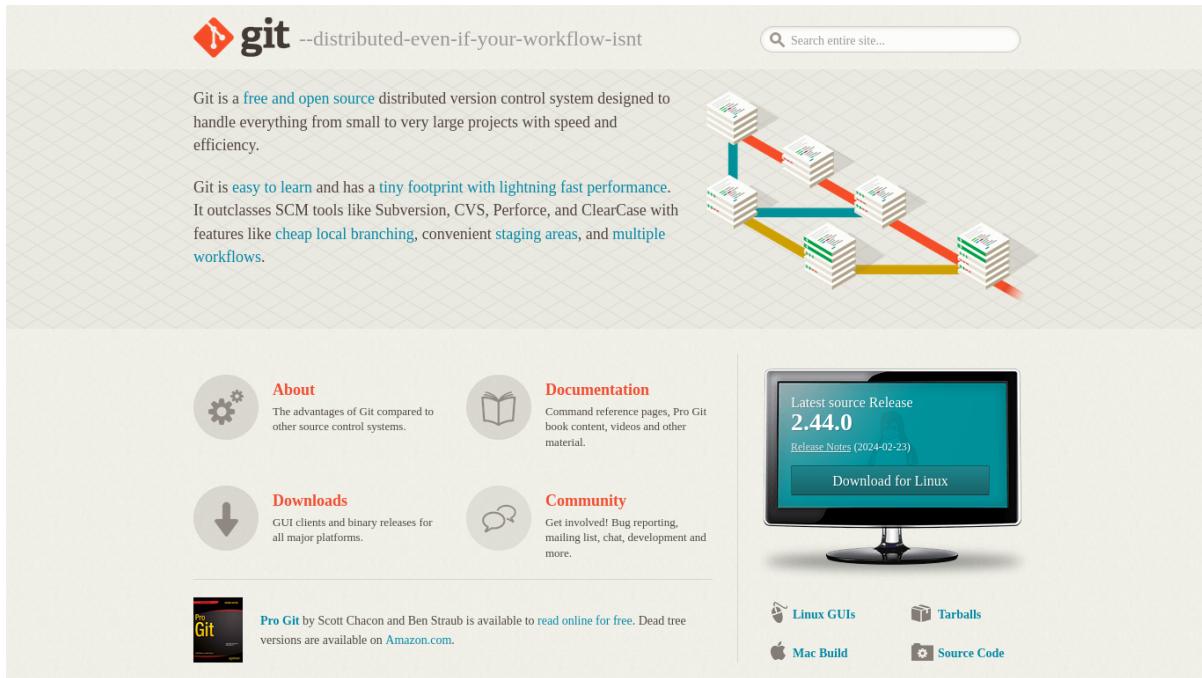


Figure 2.8: Git

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 ábrelo 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("Bienvenido,", 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!**”<sup>x</sup>

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.

The screenshot shows the GitHub Classroom interface. At the top, there's a banner with a warning about changes in assignment acceptance and starter code repositories. Below the banner, the navigation bar includes 'Classrooms / Curso de Django and React - Codings Academy / Hello World'. The main section displays an assignment titled 'Hello World'. It shows it's an individual assignment due on Feb 28, 2024, at 20:00 ET, and is currently active. There are links for the assignment URL (<https://classroom.github.com/a/Gcbhv0hp>), edit, and download. Below this, the 'Assignment Details' section shows 0 accepted assignments, 0 students, 0 assignment submissions, 0 submitted, and 0 not submitted. At the bottom, there are filters, a search bar, and sorting options.

**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.

The screenshot shows the GitHub Classroom interface for creating a new assignment. The main area is titled "Assignment basics". It includes fields for "Assignment title" (set to "Hello World"), "Deadline" (set to "02/28/2024, 08:00 PM"), and "Assignment status" (set to "Active"). The "Individual or group assignment" dropdown is set to "Individual assignment". A note states: "Assignment type cannot be changed after assignment creation." On the left sidebar, there are three tabs: "Assignment basics" (selected), "Starter code and environment", and "Grading and feedback". A yellow banner at the top says: "⚠ Assignment acceptance and starter code repositories will be changing on June 17, 2024. Review the changes and prepare your assignments." The GitHub Education logo is in the top right corner.

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

The screenshot shows the continuation of the GitHub Classroom assignment configuration. It includes sections for "Starter code and environment" and "Grading and feedback". In the "Starter code and environment" section, there is a "Find a GitHub repository" search bar containing "education/autograding-example-python". Below it, there is a "GitHub Codespaces" section with a note about enabling it for student repositories. In the "Supported editor" section, there is a checkbox for "Don't use an online IDE" which is checked. In the "Grading and feedback" section, there is a "Add autograding tests" section with a table showing one test named "Hello world test". There is also a "+ Add test" button.

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

The screenshot shows the GitHub Classroom interface. At the top, it says "GitHub Classroom". Below that, it says "Curso de Django and React - Codings Academy". There are several icons at the top right. The main content area has a heading "Accept the assignment — Hello World". It says: "Once you accept this assignment, you will be granted access to the hello-world-statick88 repository in the Coding-Academy-ec organization on GitHub." At the bottom, there is a green button labeled "Accept this assignment".

## 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.

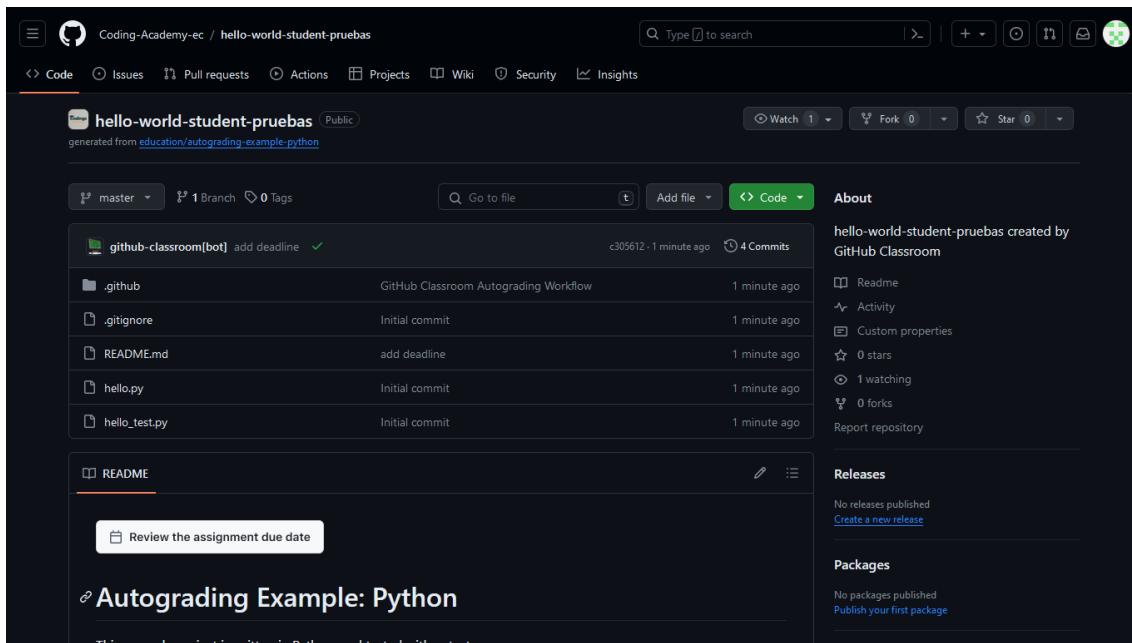
The screenshot shows the GitHub Classroom interface after accepting an assignment. At the top, it says "GitHub Classroom". Below that, it says "Join the GitHub Student Developer Pack". It says: "Verified students receive free GitHub Pro plus thousands of dollars worth of the best real-world tools and training from GitHub Education partners — for free. For more information, visit 'GitHub Student Developer Pack'." At the bottom, there is a green button labeled "Apply".

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



The screenshot shows the GitHub Classroom interface. At the top, it says "GitHub Classroom". Below that, there's a circular icon with a graduation cap and hands. The main message says "You're ready to go!" followed by "You accepted the assignment, Hello World.". It also says "Your assignment repository has been created:" and provides a link to the repository: <https://github.com/Coding-Academy-ec/hello-world-student-pruebas>. A note says "We've configured the repository associated with this assignment (update)". Below that, it says "Your assignment is due by Feb 28, 2024, 20:00 ET". On the right, there's a callout for the "GitHub Student Developer Pack" with a "Join" button.

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



The screenshot shows the GitHub repository page for "hello-world-student-pruebas". The repository is public and was generated from [education/autograding-example-python](#). It has 1 branch and 0 tags. The master branch contains files: .github-classroom[bot] add deadline, .github, .gitignore, README.md, hello.py, and hello\_test.py. All files were committed 1 minute ago. There is a "Review the assignment due date" button. The repository has 1 watch, 0 forks, and 0 stars. It has 1 watching and 0 forks. The "About" section shows it was created by GitHub Classroom. The "Releases" section shows no releases published. The "Packages" section shows no packages published.

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.

```

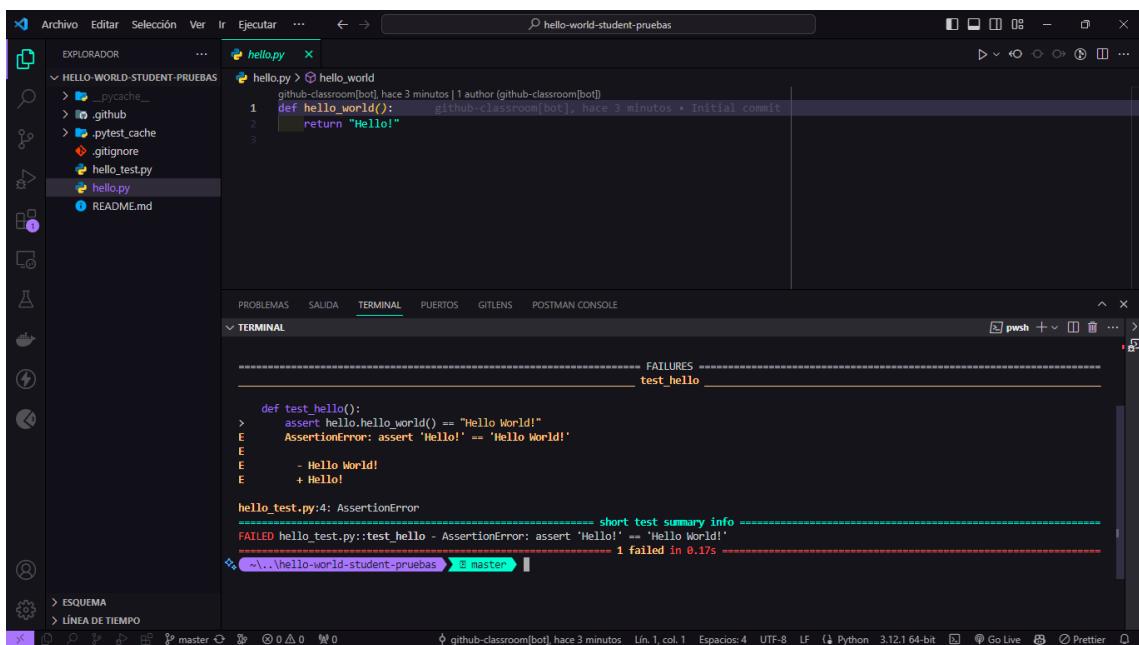
**Desarrollar la Tarea:** Los estudiantes trabajan en la tarea, realizando los cambios necesarios y realizando commits de manera regular para mantener un historial de su trabajo.

### 💡 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.

```

hello.py > hello.world
You hace 1 segundo | 2 authors (You and others)
1 def hello_world():
2     return "Hello World!" You, hace 1 segundo * Uncommitted changes
3

PROBLEMAS SALIDA TERMINAL PUERTOS GITLENS POSTMAN CONSOLE
TERMINAL
~\.\hello-world-student-pruebas > master ~1 git add .\hello.py
~\.\hello-world-student-pruebas > master ~1 git commit -m "Update Hello World! in hello.py"
[master 285741e] Update Hello World! in hello.py
1 file changed, 1 insertion(+), 1 deletion(-)
~\.\hello-world-student-pruebas > master git push -u origin main
You, hace 1 segundo Lin. 2, col. 25 Espacios: 4 UTF-8 LF Python 3.12.1 64-bit Go Live Prettier

```

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

### 💡 Tip

Se recomienda hacer las pruebas en local antes de enviar los cambios al repositorio en Github

**hello-world-student-pruebas** Public  
generated from education/autograding-example-python

master 1 Branch 0 Tags

student-pruebas Update Hello World! in hello.py 45s ago 5 Commits

Some checks haven't completed yet  
1 queued check  
GitHub Classroom Workflow / Autograding (push) Queued - Waiting to run this check... Details

Update Hello World! in hello.py now  
Initial commit 9 minutes ago

About  
hello-world-student-pruebas created by GitHub Classroom

Readme Activity Custom properties 0 stars 1 watching 0 forks Report repository

Releases No releases published Create a new release

Packages No packages published Publish your first package

Autograding Example: Python

Este Action lo que hace es evaluar los cambios realizados

The screenshot shows an Autograde session summary. Under the 'Autograde' tab, it indicates a success message: 'succeeded now in 6s'. Below this, a log window displays the command run: 'Run education/autograding@v1'. The log output shows the execution of a test script, 'hello\_test.py', which resulted in 1 passed test in 0.01s. The final status is 'All tests passed' with a score of 'Points 100/100'.

### 💡 Tip

Se recomienda hacer las pruebas en local antes de enviar los cambios al repositorio 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.

## Hello World

This screenshot shows a GitHub Classroom assignment page for 'Hello World'. The assignment is marked as 'Individual assignment' due on 'Feb 28, 2024, 20:00 ET' and is currently 'Active'. The URL is <https://classroom.github.com/a/Gcbhv0hp>. The assignment details show 1 accepted assignment, 1 student, 1 submission (1 submitted, 0 not submitted), and 1 passed student (1/1 Passed). A progress bar indicates 100% completion. The total students section lists 'student-pruebas' as having submitted the assignment, with a commit history showing 'Latest commit 2 minutes ago' and a score of 100/100. A 'Repository' link is also present.



© 2024 GitHub, Inc.

[Terms](#)

[Privacy](#)

[Security](#)

[Status](#)

[Docs](#)

[Contact GitHub](#)

[Pricing](#)

[API](#)

[Training](#)

[Blog](#)

[About](#)



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

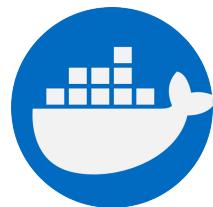


Figure 6.1: Docker

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 Introducción

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

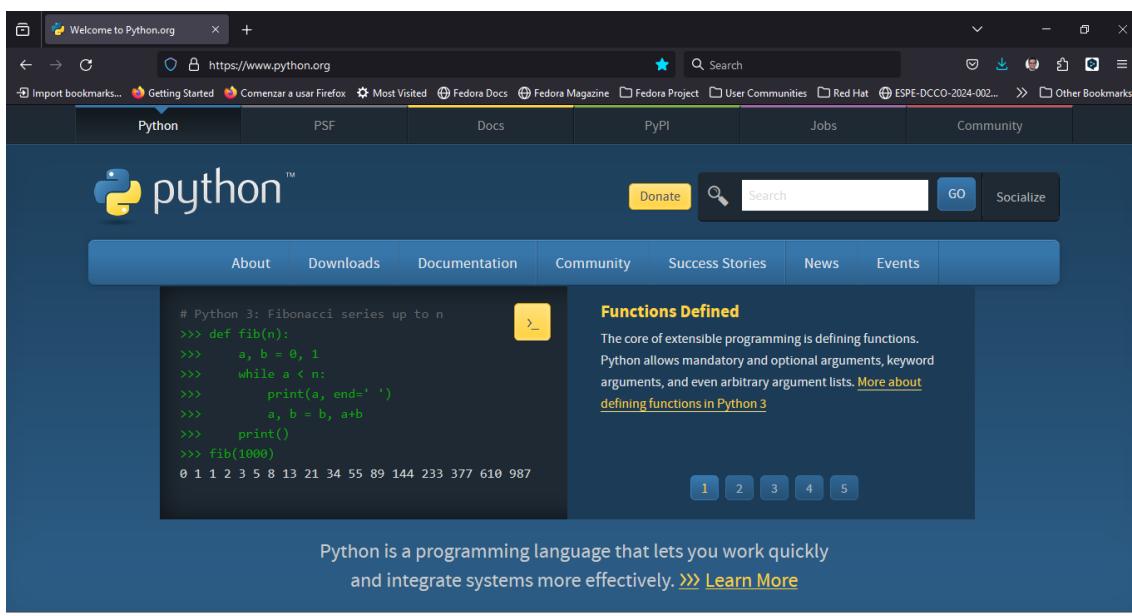
Python se puede instalar en Windows, Mac y Linux. En este tutorial, veremos cómo instalar Python en Windows.

## 10.0.2 Paso 2: Instalación de Python en Windows

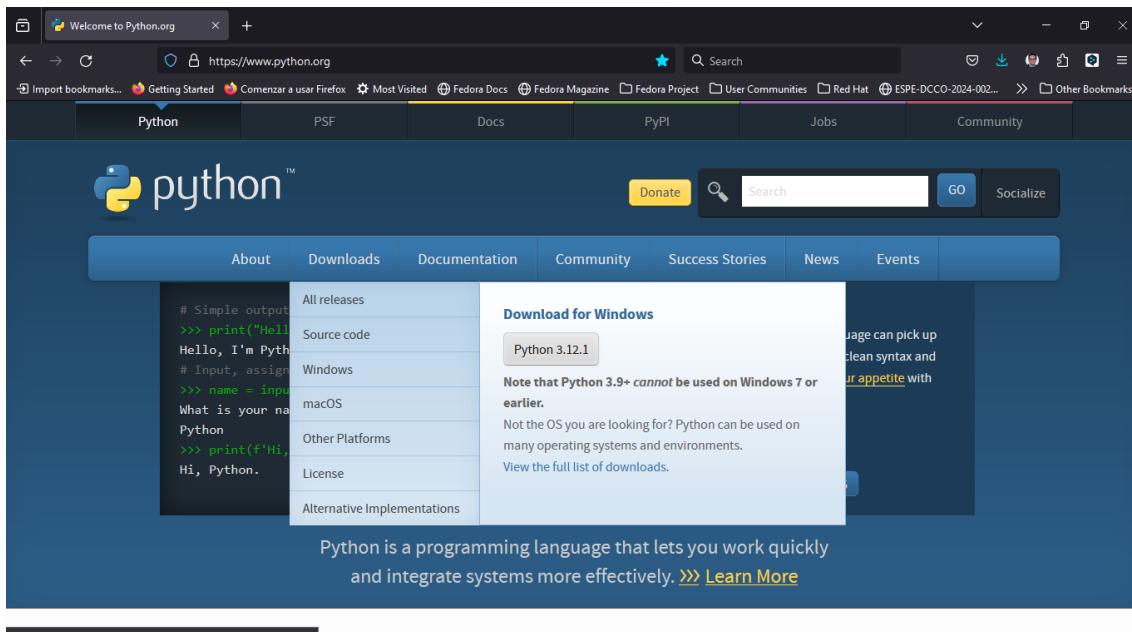
### 1. Descargar Python

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/>



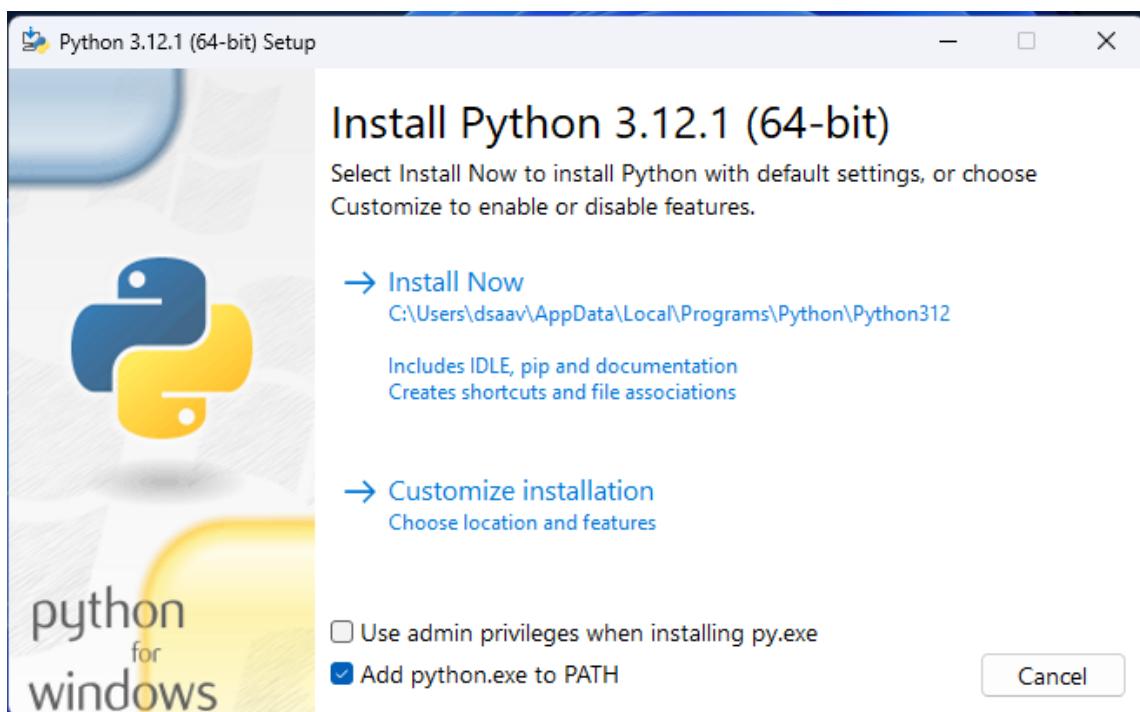
En la página de descargas, haga clic en el botón de descarga para la última versión de Python. En el momento de escribir este tutorial, la última versión de Python es 3.12.1



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. Impresión 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.

A screenshot of a PowerShell window titled "dsaav :: pwsh". The window shows the following text:

```
PowerShell 7.4.1
Loading personal and system profiles took 1113ms.
~ python --version
Python 3.12.1
~ |
```

The background of the window is dark blue with a green leaf pattern.

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

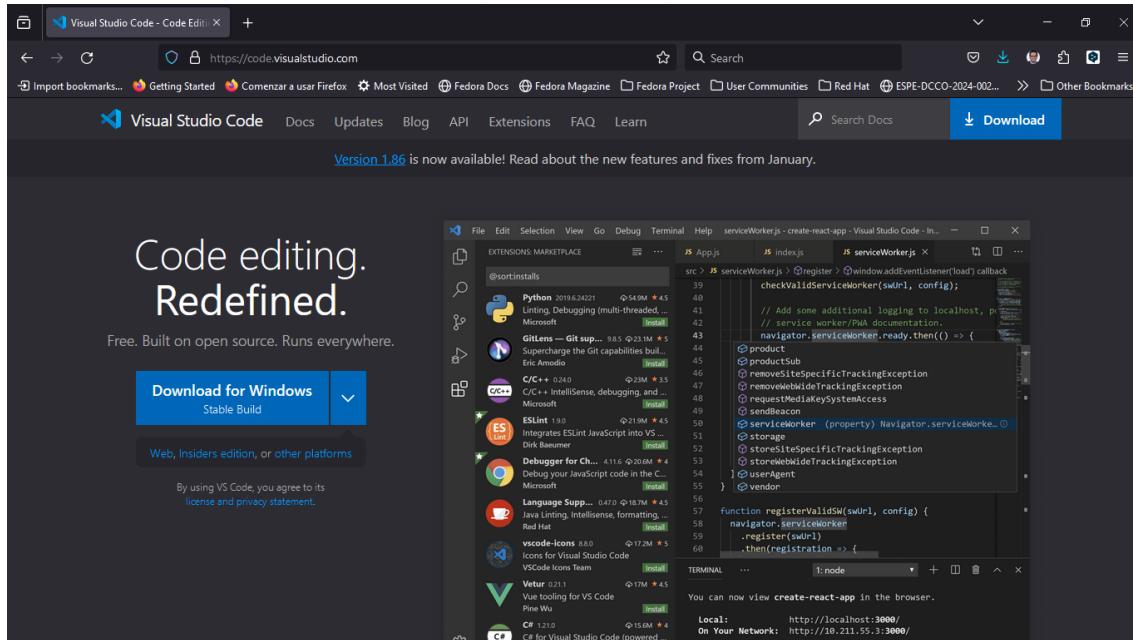
Ahora que hemos instalado Python en nuestra computadora, podemos comenzar a escribir y ejecutar programas en Python. Para hacer esto, necesitamos un editor de texto para escribir nuestro código y un intérprete de Python para ejecutar nuestro código.

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

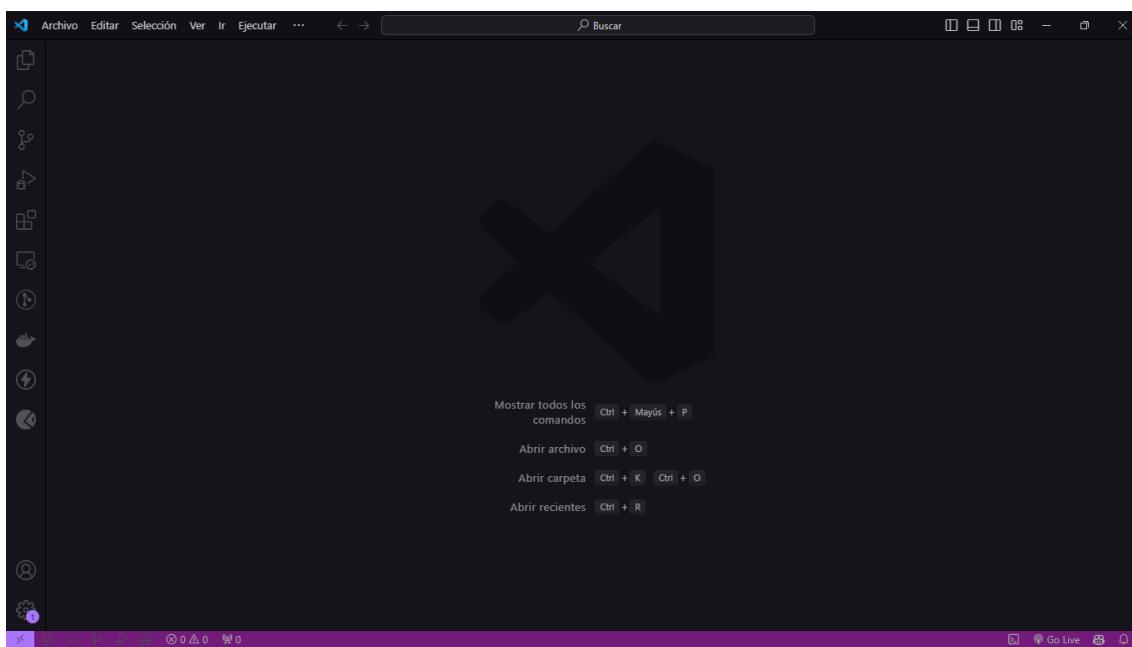
Para instalar Visual Studio Code, vaya al sitio web oficial de Visual Studio Code en el siguiente enlace:

<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.

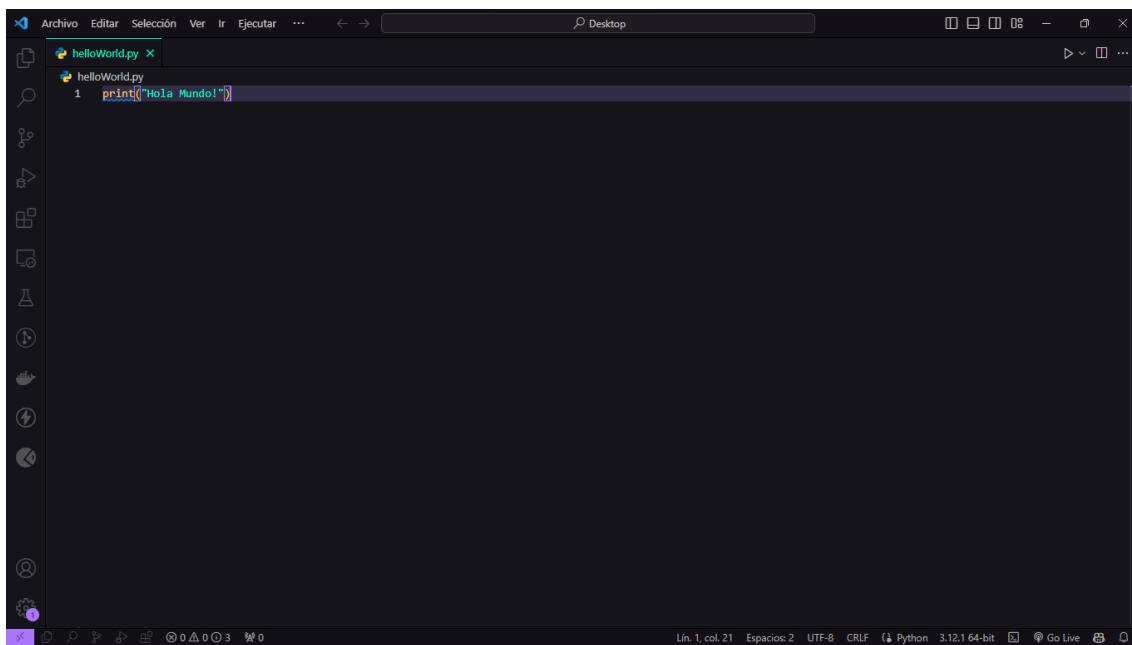


## 2. Crear un nuevo archivo de Python

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.

The screenshot shows a dark-themed code editor window titled "helloWorld.py". The code in the editor is:

```
1 print("Hola Mundo!")
```

Below the editor, the terminal output shows the program running:

```
[Running] python -u "c:\Users\dsav\Desktop\helloWorld.py"
Hola Mundo!
```

At the bottom right of the terminal area, it says "[Done] exited with code=0 in 0.149 seconds". The status bar at the bottom of the window displays "Lin. 1, col. 21 Espacios: 2 UTF-8 CRLF Python 3.12.1 64-bit Go Live".

¡Felicitaciones!

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:  
    print("Cinco es mayor que dos")
```

(1)

- 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  
y: str = "Hola Mundo"
```

(1)  
(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  
y = "Hola Mundo"
```

(1)  
(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**.

- $\geq$ : 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**.

- $\leq$ : 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):
    print(x)
```

(1)  
(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:
    print(x)
    x += 1
```

(1)  
(2)  
(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:
    print(x)
    x += 1
    if x == 3:
        break
```

(1)  
(2)  
(3)  
(4)  
(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:
    x += 1
    if x == 3:
        continue
    print(x)

```

(1)  
(2)  
(3)  
(4)  
(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:
    pass

```

(1)  
(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):
    return x + y

```

(1)  
(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):
    return x + y

```

(1)  
(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)
except:                               (2)
    print("Ocurrió un error")          (3)

```

- 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)
except:                               (2)
    print("Ocurrió un error")          (3)

```

- 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)
except:                               (2)
    print("Ocurrió un error")          (3)
finally:                             (4)
    print("Finalizó la ejecución")     (5)

```

- 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):  
    return x + y
```

- 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):  
    return x + y  
  
z = suma(5, 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**.
- 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):  
    return x + y  
  
z = suma(5, 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**.
- 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__":
    main()
```

(12)  
(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 propuestos 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 característica principal de las tuplas es que son inmutables, por lo que no se pueden modificar.

## 26 Funciones integradas 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
=====
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': 'Ingeniero'}
```

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': 'Ingeniero'}
```

Respuesta

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

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.



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.

Ejemplo:

```
class Perro:  
    def __init__(self, raza, color, edad):  
        self.raza = raza  
        self.color = color  
        self.edad = edad  
  
    def ladrar(self):  
        print("Guau! Guau!")
```

Ejercicio:

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

```

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:

```

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

```
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:

```

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

```

Ejercicio:

Imprimir el nombre de la persona “Juan”.

Ver respuesta

```
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

```
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

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

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

# 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
```

```
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:

```
# 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:

```
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.

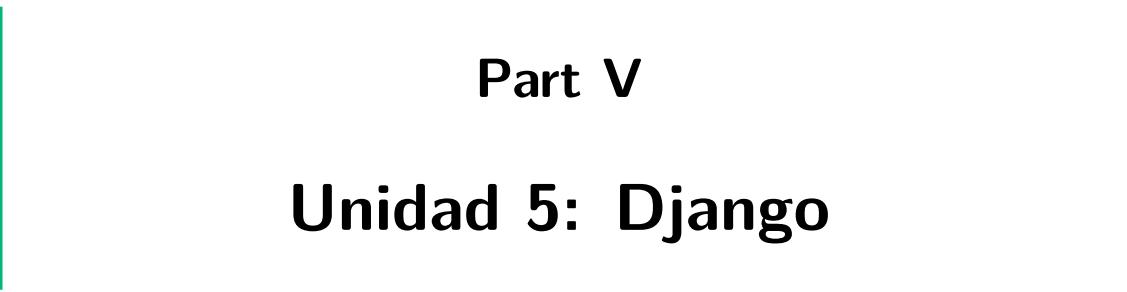
Te enseñaré cómo implementar funciones de búsqueda y filtrado para encontrar productos específicos en base a diferentes criterios, como el nombre, la categoría o el precio.

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

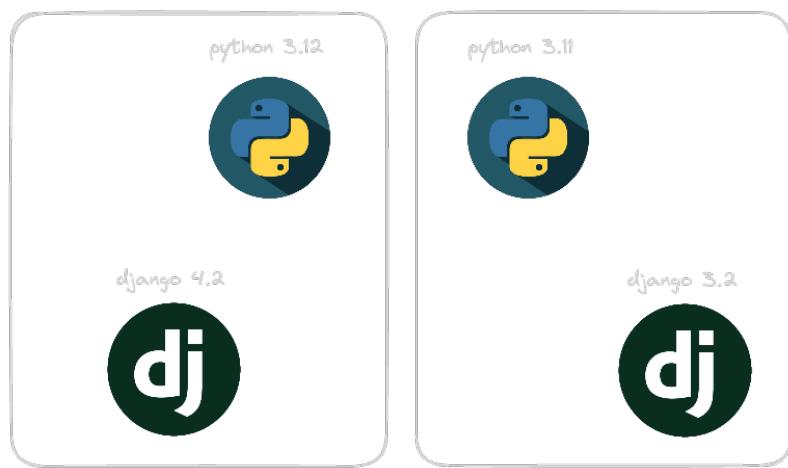


Figure 42.2: Virtual Environment

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.

```

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"])

```

 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