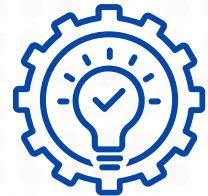


# ¿Quiénes somos?



# Lo que nos hace únicos

**NobleProg**



Soluciones a la medida



Entrenadores altamente calificados



Programas integrales



Cursos prácticos

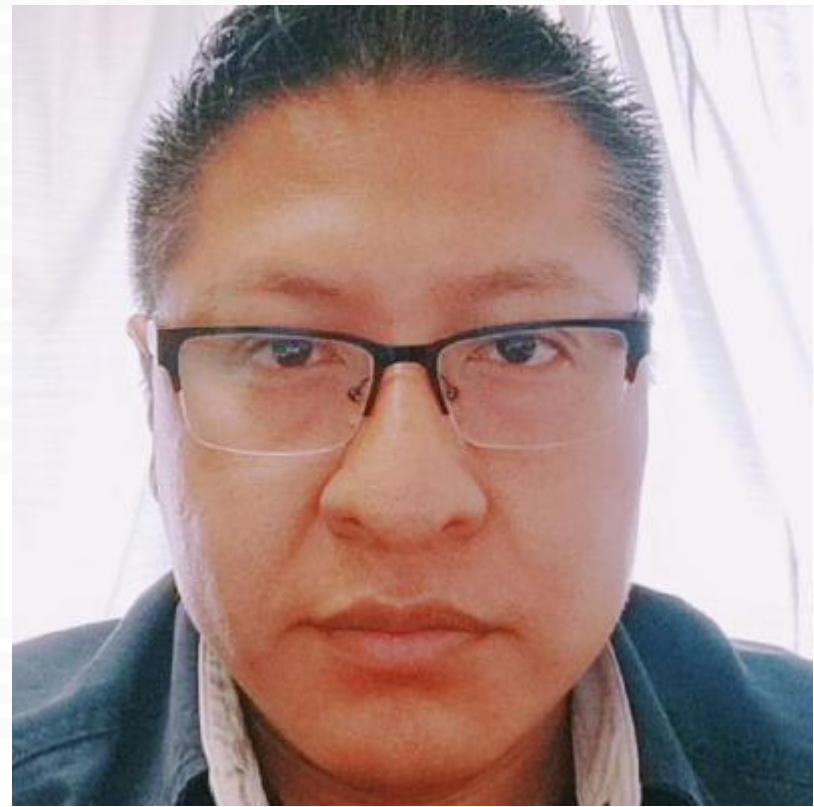


Atención personalizada



Grupos reducidos





Javier Antunez E. Ingeniero en Sistemas Computacionales egresado del Instituto Politécnico Nacional, con Maestría en Tecnologías de la Información.

Con más de **12 años** de experiencia en investigación aplicada en el Instituto de Investigaciones Eléctricas (hoy INEEL), actualmente colaboro en una empresa privada de desarrollo de software con certificación CMMI nivel 3, enfocada en calidad, automatización y mejora continua en procesos tecnológicos.



Para contribuir a la  
sustentabilidad energética  
el IIE evoluciona



# Selenium con Python para Automatización de Pruebas

1. Fundamentos de Selenium y Python
2. Arquitectura y Herramientas de Selenium
3. Automatización con Selenium WebDriver y Python
4. Escalabilidad, Prácticas Avanzadas y Cierre

# 1. Fundamentos de Selenium y Python

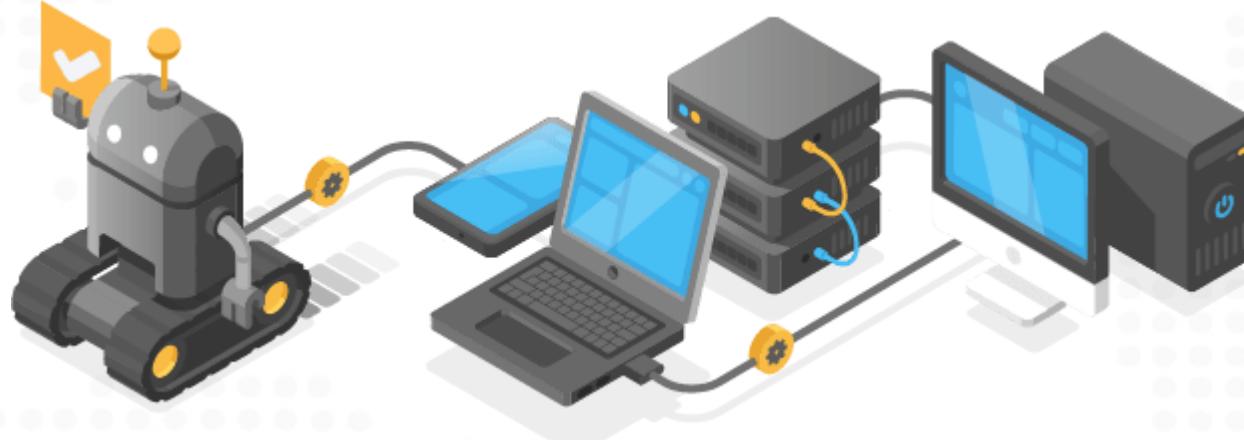
- Introducción a Selenium con Python.
- Python vs Java para escribir scripts de prueba.
- Instalación y configuración
- Seleccionar un IDE o editor de Python

# 1. Fundamentos de Selenium y Python

NobleProg

## Introducción a Selenium con Python

### ¿Qué son las pruebas automatizadas?



### ¿Por qué Selenium con Python?

- Python es fácil de aprender y leer.
- Integración con pytest, unittest, y otras herramientas de testing.
- Se adapta para pruebas unitarias, funcionales y de integración.



### Componentes de Selenium:

- WebDriver: Control del navegador por código
- Selenium Grid: Ejecución de pruebas en paralelo y distintos entornos
- Selenium IDE: Grabación de pruebas sin escribir código



# 1. Fundamentos de Selenium y Python

## Python vs Java para escribir scripts de prueba

NobleProg

### Python

- Sintaxis simple y fácil de aprender.
- Código más conciso y legible.
- Amplia comunidad en testing y automatización.
- Integración sencilla con frameworks como `pytest` o `unittest`.
- Muy útil para prototipos rápidos y scripts pequeños.



### Java

- Sintaxis más estricta y estructurada
- Fuerte tipado, lo que reduce errores en tiempo de compilación.
- Amplio soporte para frameworks de testing como JUnit y TestNG.
- Más utilizado en entornos corporativos grandes y proyectos de larga duración.



# 1. Fundamentos de Selenium y Python

NobleProg

## Python vs Java para escribir scripts de prueba

### Ventajas de Python vs Java en pruebas automatizadas

- Sintaxis más simple y legible.
- Frameworks de testing más ligeros:
  - Pytest.
  - Unittest.
  - Behave.
- Mayor velocidad en prototipado y ejecución.
- Integración fluida con herramientas modernas.
- Ideal para pruebas con IA, ML o procesamiento de datos.

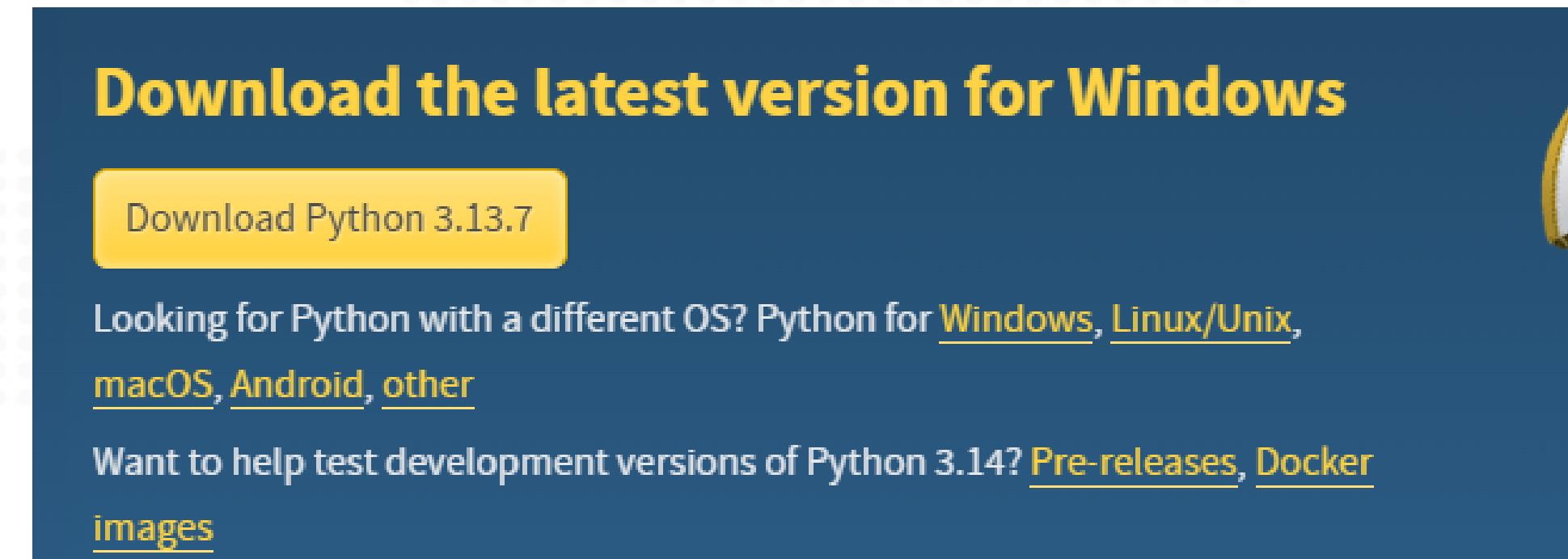
# 1. Fundamentos de Selenium y Python

## Instalación y configuración

NobleProg

### 1. Descarga de Python

- Ir a la página oficial: <https://www.python.org/downloads/>
- Elegir la versión más reciente estable (por ejemplo, Python 3.x)
- Descargar el instalador según el sistema operativo (Windows, macOS, Linux)



# 1. Fundamentos de Selenium y Python

NobleProg

## Instalación y configuración

### 2. Instalación en Windows

- Ejecutar el instalador descargado.
- **Importante:** Marcar la opción “Add Python to PATH”
- Seleccionar **Install Now** o **Customize Installation** según necesidad



Verificar la instalación abriendo CMD y ejecutando:

**python --version**

**pip --version**

A screenshot of a Windows PowerShell window titled "Windows PowerShell". The command "python --version" was run, resulting in the output "Python 3.13.6". The command "pip --version" was also run, resulting in the output "pip 25.2 from C:\Users\javie\AppData\Local\Programs\Python\Python313\Scripts\pip.py". The PowerShell window has a dark theme.

# 1. Fundamentos de Selenium y Python

## Instalación y configuración

NobleProg

### 3. Configuración del entorno de desarrollo

- Configurar un entorno virtual para proyectos:

```
python -m venv .venv # Crea el entorno en una carpeta llamada .venv  
.venv\Scripts\activate.bat # Activar el entorno virtual
```

- Instalación del conjunto de paquetes básicos para pruebas automatizadas:

- pip install selenium
- pip install webdriver-manager
- pip install pytest

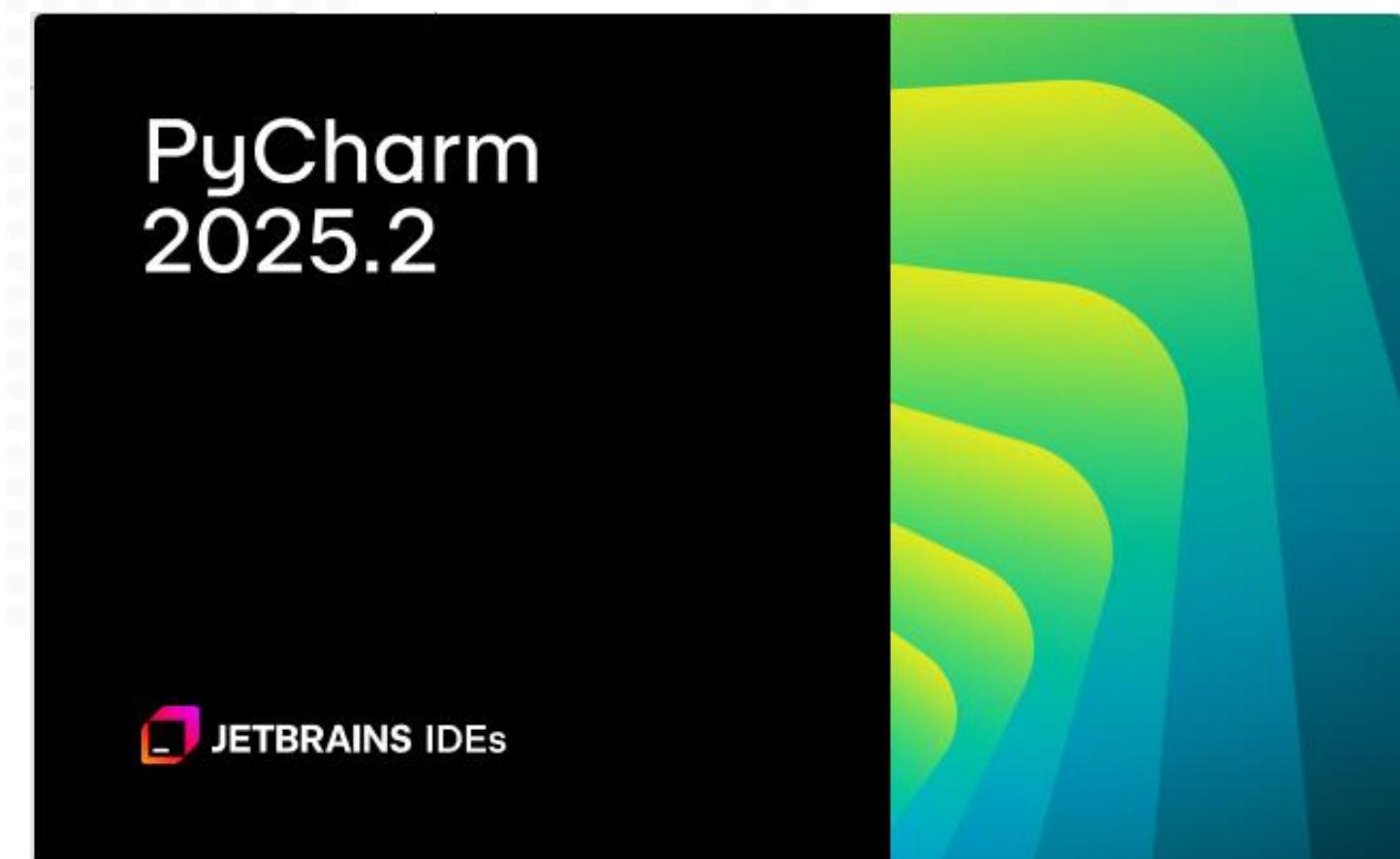
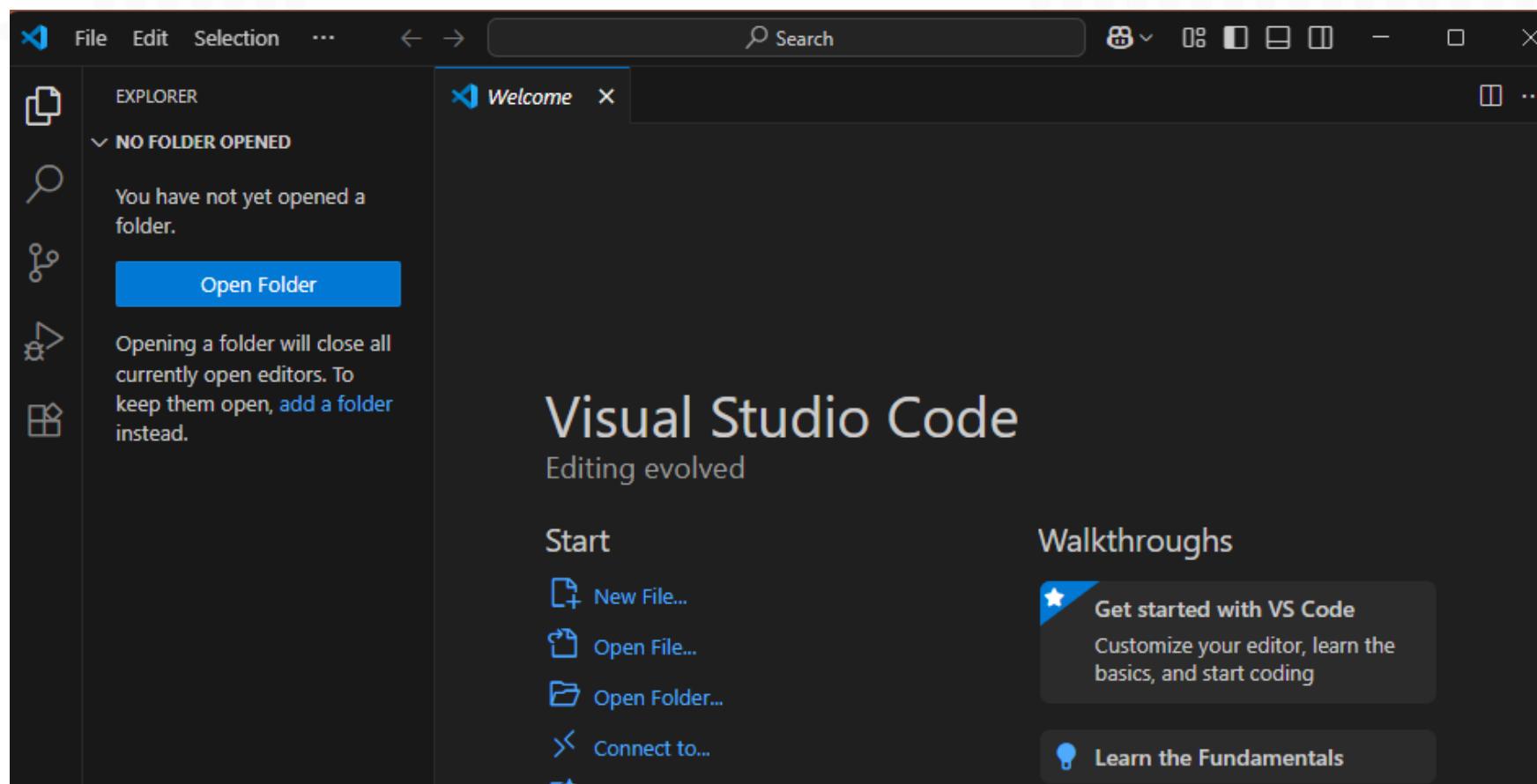
# 1. Fundamentos de Selenium y Python

NobleProg

## Seleccionar un IDE o editor de Python

Instalar un editor o IDE:

- VS Code
- PyCharm

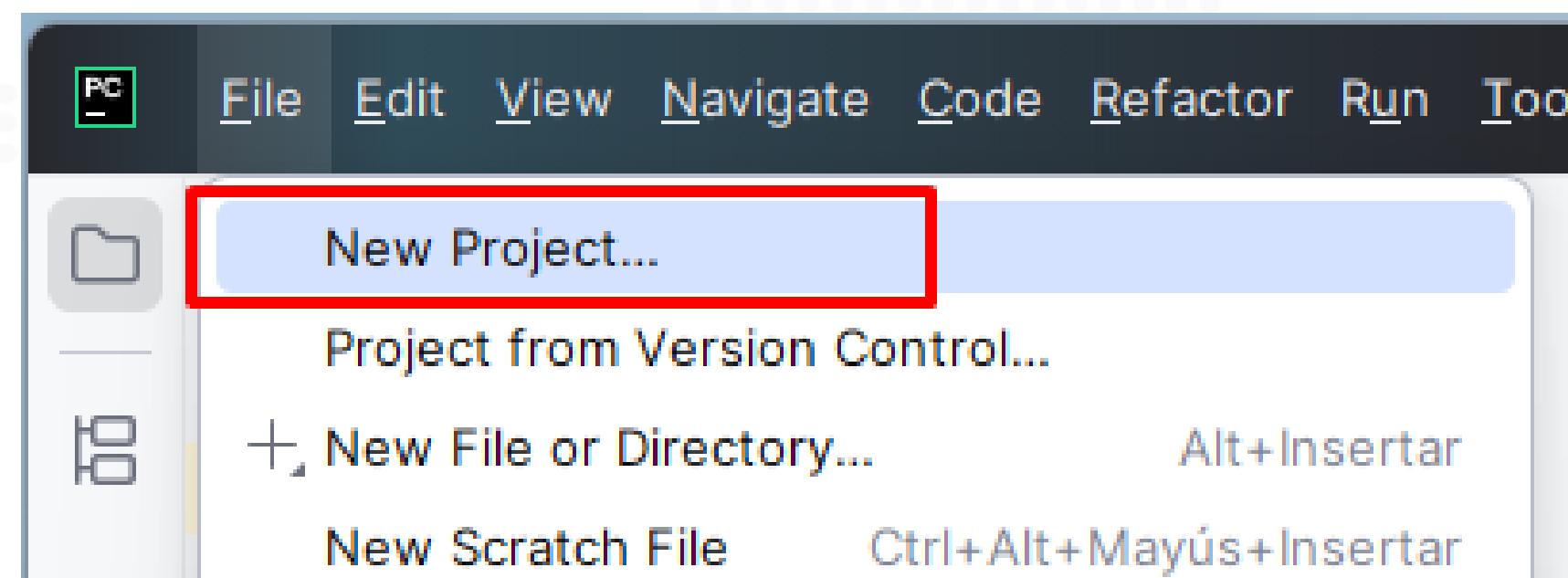


# 1. Fundamentos de Selenium y Python

## Seleccionar un IDE o editor de Python

Crear un nuevo proyecto en PyCharm:

1. Abrir PyCharm
2. Crear un nuevo proyecto
  - a. Haz clic en "New Project".
3. Configurar el proyecto
  - a. Location (Ubicación): Elige la carpeta donde se guardará tu proyecto



# 1. Fundamentos de Selenium y Python

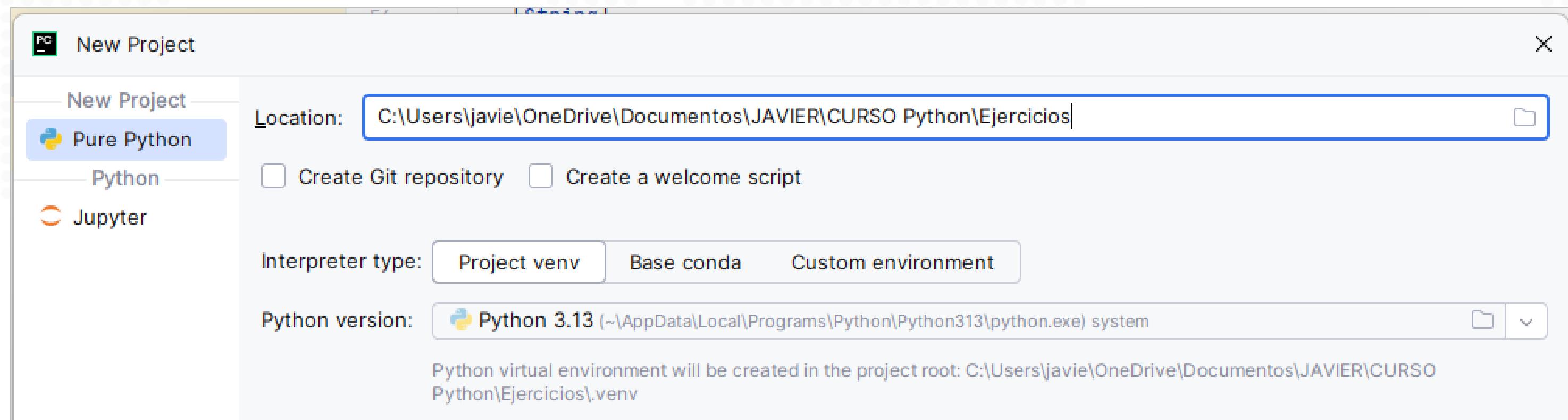
NobleProg

## Seleccionar un IDE o editor de Python

### 3. Configurar el proyecto

#### b. Python Interpreter (Intérprete de Python), aquí se puede elegir:

- **New Virtualenv environment** → crea un entorno virtual aislado para tu proyecto.
- **Conda environment** (si usas Anaconda).
- **Existing interpreter** → si ya tienes instalado Python en tu máquina.



# 1. Fundamentos de Selenium y Python

NobleProg

## Seleccionar un IDE o editor de Python

### 4. Finalizar

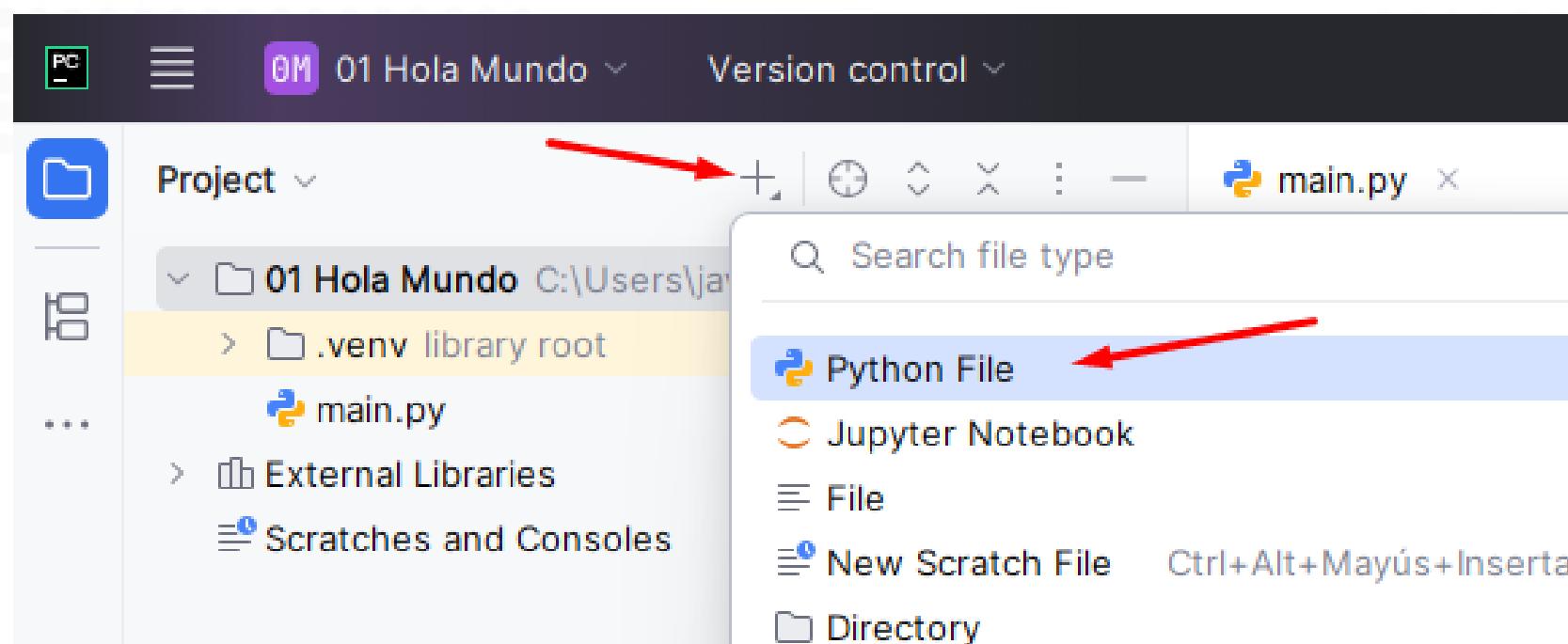
- Haz clic en *Create*.

### 5. Probar que funciona

- Dentro del proyecto, crea un nuevo archivo Python:

- Clic derecho en el proyecto → *New* → *Python File* → *main.py*

- Haz clic derecho en el archivo y selecciona Run 'main'.



The screenshot shows the 'Run' tool window in PyCharm. At the top, it says 'Run main'. Below that, there's a 'Logs' tab. The output pane shows the command 'C:\Users\javiel\OneDrive\Documentos\Curso Python\01 Hola Mundo\main.py' followed by the text 'Hola mundo.' in red, indicating an error or warning. At the bottom, it says 'Process finished with exit code 0'.

## 2. Arquitectura y Herramientas de Selenium

- Descripción de la arquitectura de Selenium
- Selenium IDE
- Selenium WebDriver
- Selenium Grid

## 2. Arquitectura y Herramientas de Selenium

NobleProg

### Descripción de la arquitectura de Selenium

Selenium es un conjunto de herramientas de automatización que permite controlar navegadores web de forma programática para realizar pruebas funcionales, de regresión y validación de interfaces.

Los componentes principales de la arquitectura de Selenium:

- Selenium IDE
- Selenium WebDriver
- Selenium Grid



## 2. Arquitectura y Herramientas de Selenium

NobleProg

### Selenium IDE

Selenium IDE (*Integrated Development Environment*) es una herramienta de grabación y reproducción que permite automatizar interacciones en un navegador sin necesidad de programar código complejo.

#### Ventajas

- Muy fácil de usar, ideal para principiantes.
- No requiere conocimientos avanzados de programación.
- Útil para crear prototipos rápidos de pruebas automatizadas.
- Permite probar escenarios simples de forma inmediata.



#### Limitaciones

- No es tan robusto para pruebas complejas.
- Menos flexible que Selenium WebDriver.
- No recomendado para proyectos grandes, solo para pruebas rápidas o de aprendizaje.



## 2. Arquitectura y Herramientas de Selenium

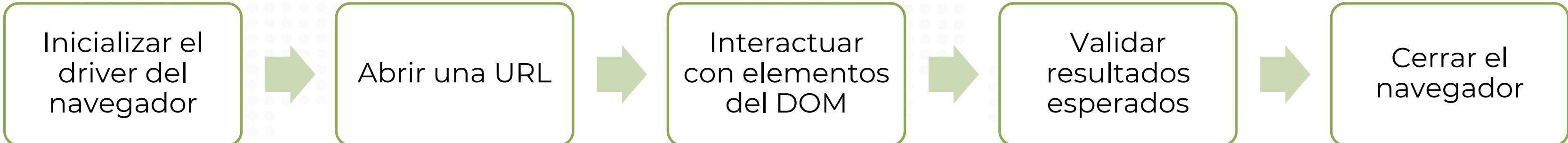
### Selenium WebDriver

Selenium WebDriver es el componente principal de Selenium que permite automatizar acciones en navegadores web.

#### Recomendaciones:

- Usa WebDriverManager para no tener que descargar manualmente los binarios.

```
# Inicializar navegador con WebDriver Manager (descarga automático del driver)  
driver = webdriver.Chrome(ChromeDriverManager().install())
```



## 2. Arquitectura y Herramientas de Selenium

### WebDriver API

NobleProg

#### ¿Qué ofrece?

```
# Abrir una página  
driver.get("https://www.google.com")
```

```
# Buscar un elemento (por ejemplo, el input de búsqueda)  
search_box = driver.find_element(By.NAME, "q")  
search_box.send_keys("Selenium WebDriver en Python")  
search_box.submit()
```

```
# Cerrar navegador  
driver.quit()
```

Abrir y cerrar el navegador

Navegar entre páginas

Localizar y manipular elementos del DOM

Enviar texto a formularios

Hacer clic en botones o enlaces

Capturar textos, atributos y estados

Manejar ventanas emergentes, alertas y frames.

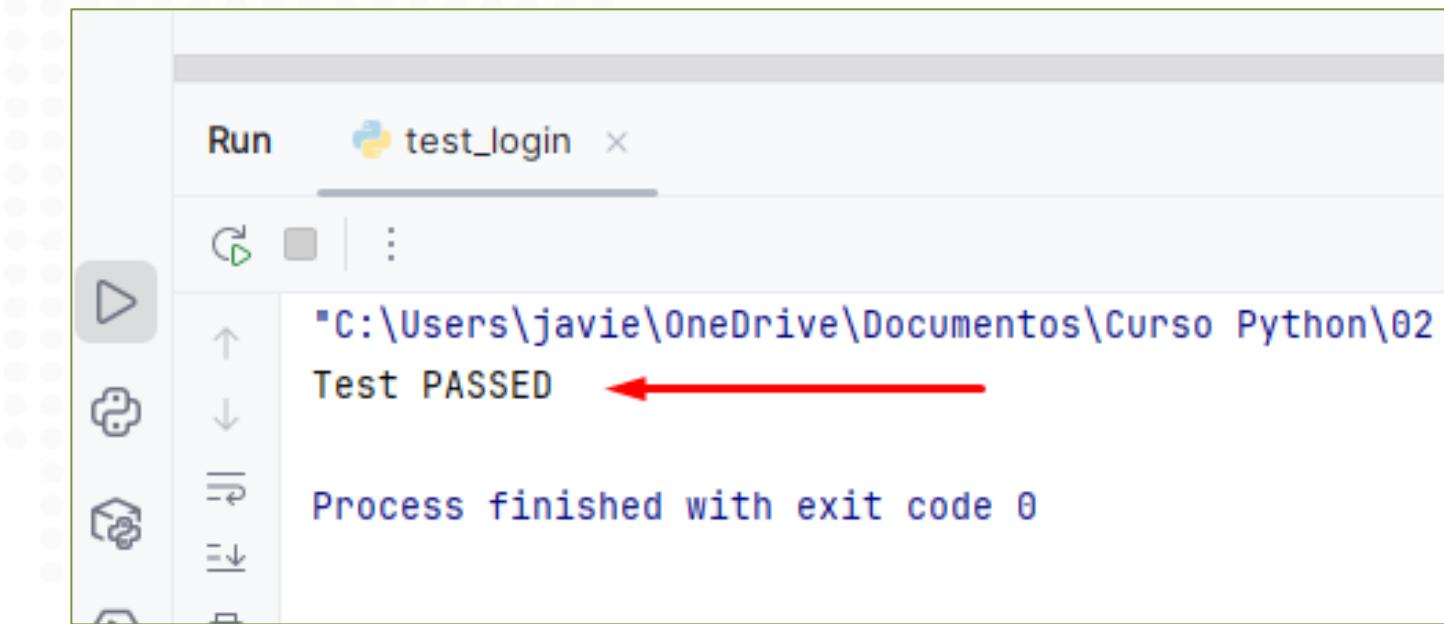
## 2. Arquitectura y Herramientas de Selenium

### WebDriver API

NobleProg

#### Practica:

1. Crear nuevo proyecto en PyCharm
2. Instalar Selenium
  - pip install selenium
3. Crear un archivo Python (test\_login.py)
4. Ingresa en un Login con usuario y contraseña
  - Url: [https://the-internet.herokuapp.com/login?utm\\_source=chatgpt.com](https://the-internet.herokuapp.com/login?utm_source=chatgpt.com)
  - Username: *tomsmith*
  - Password: *SuperSecretPassword!*
- Ejecutar en PyCharm



## 2. Arquitectura y Herramientas de Selenium

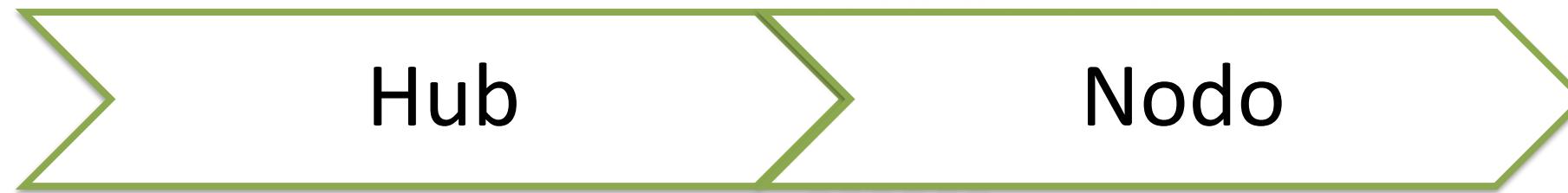
### Selenium Grid

NobleProg

Selenium Grid es un componente de Selenium que permite ejecutar pruebas automatizadas en múltiples máquinas, navegadores y sistemas operativos al mismo tiempo. Su objetivo principal es distribuir la carga de ejecución para acelerar las pruebas y validar tu aplicación en distintos entornos.

#### Arquitectura general

Componente	Descripción
Hub	Servidor central que recibe las pruebas y decide dónde ejecutarlas
Nodo	Son las máquinas (virtuales o físicas) donde se ejecutan los navegadores reales



## 2. Arquitectura y Herramientas de Selenium

### Selenium Grid

NobleProg

En versiones más recientes de Selenium (4+), Selenium Grid puede ejecutarse en modo **Standalone**, **Hub-Node**, o **Docker/Grid Server** (más moderno y flexible con capacidades distribuidas usando herramientas como Docker y Kubernetes).

¿Cuál elegir según tu contexto?

- **Standalone (Modo independiente )**: Útil para pruebas locales rápidas, debugging o entornos CI simples.
- **Hub-Node (modo Clásico)**: Ideal si tienes varias máquinas físicas o virtuales y quieres distribuir la carga.
- **Docker/Grid Server (modo distribuido moderno)**: perfecto para pipelines CI/CD, entornos dinámicos, y pruebas masivas en paralelo.

## 2. Arquitectura y Herramientas de Selenium

### Selenium Grid

NobleProg

#### Practica

##### 1. Descargar Selenium Server

- [https://www.selenium.dev/downloads/?utm\\_source=chatgpt.com](https://www.selenium.dev/downloads/?utm_source=chatgpt.com)
- Guardarlo en carpeta C:\selenium-grid

##### 2. Instalar Java

- [https://adoptium.net/temurin/releases/?utm\\_source=chatgpt.com](https://adoptium.net/temurin/releases/?utm_source=chatgpt.com)
- Establecer la variable JAVA\_HOME
- java --version

##### 3. Levantar el Hub

- java -jar selenium-server-4.35.0.jar hub
- El hub quedará escuchando en <http://localhost:4444>

##### 4. Levantar un Nodo (ej. Chrome)

- java -jar selenium-server-4.35.0.jar node
- --port 5555
- --max-sessions 3

### **3. Automatización con Selenium WebDriver y Python**

- Python scripting essentials para automatización de pruebas
- Interacciones con elementos web
- Interacciones con elementos web avanzado
- Sincronización de pruebas
- Creando una prueba unitaria (POM)
- Accediendo a una base de datos

### 3. Automatización con Selenium WebDriver y Python

#### Python scripting essentials para automatización de pruebas

NobleProg

Tipos de datos y estructuran en Python:

- **str**: Cadena de texto
- **int**: Número entero
- **bool**: Valor lógico
- **list**: Colección ordenada y mutable
- **dict**: Colección de pares clave-valor
- **tuple**: Colección ordenada e inmutable

```
nombre = "Javier"    # str (cadena de texto)
edad = 30            # int (entero)
altura = 1.75         # float (decimal)
es_activo = True     # bool (booleano)
```

Python **no** es un lenguaje fuertemente tipado en tiempo de compilación, por lo que no restringe el tipo en tiempo de ejecución.

### 3. Automatización con Selenium WebDriver y Python

NobleProg

#### Python scripting essentials para automatización de pruebas

```
nombre: str = "Javier"      # str (cadena de texto)
edad: int = 3030            # int (entero)
altura: float = 1.751.75    # float (decimal)
es_activo: bool = True      # bool (booleano)
```

Esto no cambia la ejecución, pero ayuda a mantener el código más claro.

```
# list -> Colección ordenada y mutable (ejemplo: lista de usuarios de prueba)
usuarios: list = ["admin", "tester", "invitado"]
```

```
# dict -> Colección de pares clave-valor (ejemplo: credenciales por usuario)
credenciales: dict = {
    "admin": "admin123",
    "tester": "test123"
}
```

```
# tuple -> Colección ordenada e inmutable (ejemplo: resoluciones de pantalla soportadas para pruebas)
resoluciones: tuple = (("1920x1080"), ("1366x768"), ("1280x720"))
```

# 3. Automatización con Selenium WebDriver y Python

Python scripting essentials para automatización de pruebas

NobleProg

## Practica

- Crear nuevo proyecto en PyCharm
- Crear un archivo Python (data\_types.py)
- Crear un programa donde se usen los siguientes tipos de datos:
  - Int
  - String
  - Float
  - bool
  - List
  - dict
  - Tuple

```
# string (cadena de texto)
nombre = "Javier"
print("Nombre (string):", nombre, type(nombre))

print(f"Hola, me llamo {nombre}.")
```

### 3. Automatización con Selenium WebDriver y Python

Python scripting essentials para automatización de pruebas

NobleProg

Uso de `if/else` para validar condiciones:

```
if actual_title == expected_title:  
    print("El título de la página es correcto.")  
else:  
    print(f"Título incorrecto. Se obtuvo: {actual_title}")
```

Uso de `for` para bucles:

```
# Conjunto de datos (usuario, contraseña)  
datasets = [  
    ("user1", "pass1"),  
    ("user2", "pass2"),  
]  
for username, password in datasets:  
    print(f"Probando con usuario: {username} y contraseña: {password}")
```

### 3. Automatización con Selenium WebDriver y Python

Python scripting essentials para automatización de pruebas

NobleProg

Uso de while para bubbles

```
while True:  
    try:  
        element = driver.find_element(By.ID, "id-search-field")  
        print("El elemento está presente.")  
        break  
    except NoSuchElementException:  
        if time.time() - start_time > timeout:  
            print("Timeout: el elemento no apareció.")  
            break  
        time.sleep(1) # espera un segundo antes de volver a intentar
```

# 3. Automatización con Selenium WebDriver y Python

Python scripting essentials para automatización de pruebas

**NobleProg**

## Practica

- Crear nuevo proyecto en PyCharm
- Crear un archivo Python (data\_types.py, structure\_control)
- Crear un programa donde se usen las siguientes estructuras de control:
  - if/else
  - for
  - while

### 3. Automatización con Selenium WebDriver y Python

NobleProg

#### Python scripting essentials para automatización de pruebas

Funciones: bloque de código que realiza una tarea específica

```
def nombre_funcion(parametros):
    """Docstring: descripción de lo que hace la función"""
    # Bloque de código
    return resultado

def calcular_area_rectangulo(base: float, altura: float) -> float:
    """Calcula el área de un rectángulo dado su base y altura."""
    return base * altura

area = calcular_area_rectangulo(5.0, 3.0)
print(area) # Resultado: 15.0
```

- **def**: palabra clave para definir funciones.
- **Nombre descriptivo**: usa convención *snake\_case*.
- **Parámetros**: pueden tener tipos (base: float) y valores por defecto (altura=10).
- **Docstring**: triple comillas para documentar la función.
- **return**: devuelve el resultado. Si no se incluye, la función retorna *None*.

### 3. Automatización con Selenium WebDriver y Python

Python scripting essentials para automatización de pruebas

NobleProg

#### Manejo de excepciones (try/except)

```
try:  
    # Código que puede fallar  
    resultado = 10 / 0  
except ZeroDivisionError:  
    print("No se puede dividir entre cero.")
```

#### Lanzar excepciones propias con raise

```
def validar_edad(edad):  
    if edad < 0:  
        raise ValueError("La edad no puede ser negativa.")
```

```
conexion = None  
try:  
    conexion = conectar_base_datos()  
    datos = conexion.obtener_datos()  
except ConnectionError:  
    print("Error de conexión")  
else:  
    print("Datos obtenidos correctamente")  
finally:  
    # Cerrar conexión siempre, haya error o no  
    if conexion:  
        conexion.cerrar()  
    print("Recursos liberados")
```

### 3. Automatización con Selenium WebDriver y Python

Python scripting essentials para automatización de pruebas

NobleProg

#### Tipos comunes de Exeption

- **SyntaxError:** Ocurre cuando Python no puede entender tu código.
- **NameError:** Sucede cuando no se encuentra un nombre local o global.
- **TypeError:** Causado por una operación o función aplicada a un objeto de tipo inapropiado.
- **ValueError:** Se produce cuando una función recibe un argumento con el tipo correcto pero valor inapropiado.
- **IndexError:** Se activa al intentar acceder a un índice que está fuera de rango.

### 3. Automatización con Selenium WebDriver y Python

Python scripting essentials para automatización de pruebas

**NobleProg**

#### Practica:

- Crear nuevo proyecto en PyCharm
- Crear un archivo Python (functions.py)
- Crear un programa para calcular el IMC (Índice de Masa Corporal) de una persona usando lo siguiente:
  - Funciones
  - Manejo de excepciones

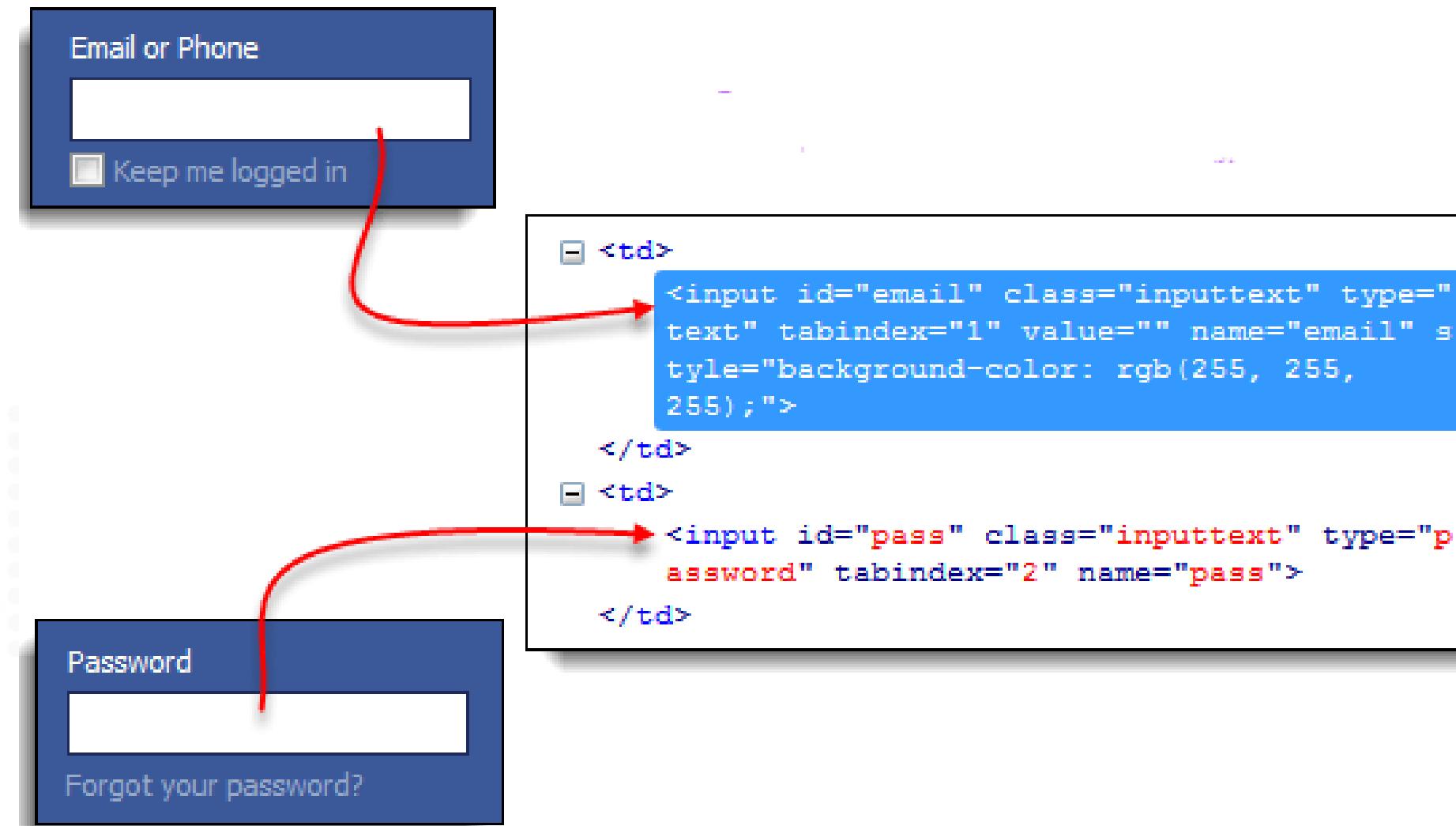
### 3. Automatización con Selenium WebDriver y Python

NobleProg

#### Interacciones con elementos web

##### ¿Qué son los Localizadores?

Son estrategias o maneras para identificar elementos HTML dentro del DOM (*Document Object Model*) de una página web para interactuar con ellos durante la automatización.



### 3. Automatización con Selenium WebDriver y Python

#### Interacciones con elementos web

NobleProg

Principales estrategias de búsqueda:

Estrategia	Ejemplo de uso
By.ID	(By.ID, "username")
By.NAME	(By.NAME, "q")
By.CLASS_NAME	(By.CLASS_NAME, "btn-primary")
By.TAG_NAME	(By.TAG_NAME, "a")
By.LINK_TEXT	(By.LINK_TEXT, "Iniciar sesión")
By.PARTIAL_LINK_TEXT	(By.PARTIAL_LINK_TEXT, "Iniciar")
By.CSS_SELECTOR	(By.CSS_SELECTOR, "input[type='text']")
By.XPATH	(By.XPATH, "//input[@id='username']")

# 3. Automatización con Selenium WebDriver y Python

NobleProg

## Interacciones con elementos web

```
<html>
<body>
<style>
.information {
    background-color: white;
    color: black;
    padding: 10px;
}
</style>
<h2>Contact Selenium</h2>

<form>
    <input type="radio" name="gender" value="m" />Male &nbsp;
    <input type="radio" name="gender" value="f" />Female <br>
    <br>
    <label for="fname">First name:</label><br>
    <input class="information" type="text" id="fname" name="fname" value="Jane"><br><br>
    <label for="lname">Last name:</label><br>
    <input class="information" type="text" id="lname" name="lname" value="Doe"><br><br>
    <label for="newsletter">Newsletter:</label>
    <input type="checkbox" name="newsletter" value="1" /><br><br>
    <input type="submit" value="Submit">
</form>

<p>To know more about Selenium, visit the official page
<a href ="www.selenium.dev">Selenium Official Page</a>
</p>

</body>
</html>
```

• Por name: "gender": Esto selecciona el primero; debes filtrar por *value* o usar *find\_elements*.

• Por XPath:

• //input[@type='radio' and @value='m']  
(Male)  
• //input[@type='radio' and @value='f']  
(Female)

• Por CSS Selector:

• input[type='radio'][value='m']  
• input[type='radio'][value='f']

### 3. Automatización con Selenium WebDriver y Python

#### Interacciones con elementos web avanzado

NobleProg

#### Xpath

//etiqueta[@atributo='valor']

// → busca en cualquier parte del documento.

etiqueta → el nombre del elemento (input, div, a, etc.).

@atributo='valor' → condición para identificar el elemento.

Ejemplo:

//input[@type='radio' and @value='m']

//input[@type='radio' and @value='f']

### 3. Automatización con Selenium WebDriver y Python

#### Interacciones con elementos web avanzado

NobleProg

##### CSS Selector

**etiqueta[atributo='valor']**

etiqueta → el nombre del elemento (input, div, a, etc.).

[] → Solo se usan cuando se está filtrando por atributos

Ejemplo:

input[name='gender']

input[type='radio'][value='m']

a[href='www.selenium.dev']

#fname → Por Id

.information → Por nombre de clase

### 3. Automatización con Selenium WebDriver y Python

NobleProg

#### Construyendo un camino

Se refiere al proceso de definir la ruta exacta para encontrar un elemento HTML dentro de la estructura jerárquica de la página web (el DOM).

#### ¿Dónde se aplica?

Al usar XPath y CSS Selectors

```
<div class="formulario">
  <label>Nombre</label>
  <input type="text" name="nombre">
</div>
```

Un camino absoluto en XPath sería:  
/html/body/div[1]/input

Un camino relativo (más estable y recomendado):  
//div[@class='formulario']/input[@name='nombre']

### 3. Automatización con Selenium WebDriver y Python

Construyendo un camino

**NobleProg**

Practica:

- Crear nuevo proyecto en PyCharm
- Crear un archivo Python (locators.py)
- Crear un programa para llenar un formulario de registro:
  - <https://demoqa.com/automation-practice-form>

### 3. Automatización con Selenium WebDriver y Python

NobleProg

#### Sincronización de pruebas

Cuando automatizas con Selenium, tu script puede ser más rápido que el navegador. Si intentas interactuar con un elemento que aún no ha cargado, obtendrás errores como:

- NoSuchElementException
- ElementNotInteractableException
- StaleElementReferenceException

La sincronización garantiza que Selenium espere el momento correcto para interactuar con el DOM.

### 3. Automatización con Selenium WebDriver y Python

NobleProg

#### Sincronización de pruebas

- Esperas implícitas (`implicitly_wait`)
  - Establece un tiempo de espera global para encontrar elementos.

```
driver = webdriver.Chrome()  
driver.implicitly_wait(10) # Se aplica a TODOS los find_element()
```

# Todos estos métodos usarán la espera de 10 segundos

```
driver.find_element(By.ID, "elemento1")  
driver.find_elements(By.TAG_NAME, "div")
```

```
boton = driver.find_element(By.ID, "mi-botón")
```

# Pero si el botón no es clickeable, falla inmediatamente

```
boton.click() # Puede fallar si el botón está deshabilitado
```

### 3. Automatización con Selenium WebDriver y Python

#### Sincronización de pruebas

NobleProg

- Esperas explícitas (WebDriverWait)
  - Espera hasta que se cumpla una condición específica.
- Puedes esperar por
  - Presencia en el DOM (`presence_of_element_located`)
  - Visibilidad (`visibility_of_element_located`)
  - Clickabilidad (`element_to_be_clickable`)
  - Texto específico (`text_to_be_present_in_element`)

### 3. Automatización con Selenium WebDriver y Python

#### Sincronización de pruebas

NobleProg

- Esperas explícitas (WebDriverWait)

```
from selenium.webdriver.support import expected_conditions as EC  
from selenium.webdriver.common.by import By
```

*# Esperar hasta que se cumpla una condición específica*  
elemento = wait.until(EC.presence\_of\_element\_located((By.ID, "mi-elemento")))

### 3. Automatización con Selenium WebDriver y Python

#### Sincronización de pruebas

NobleProg

##### Practica:

- Crear nuevo proyecto en PyCharm
- Crear un archivo Python (waits.py)
- Crear un programa para llenar un formulario de registro usando las esperas explicitas:
  - <https://demoqa.com/automation-practice-form>

### 3. Automatización con Selenium WebDriver y Python

NobleProg

#### Creando una prueba unitaria

Una **prueba unitaria** en Python es un pequeño test que verifica automáticamente que una función o método específico haga lo que esperamos.

En Python, el módulo estándar para esto es **unittest**.

EL objetivo de una prueba unitaria es:

- Devolver los resultados esperados ante entradas conocidas.
- Maneja correctamente errores y excepciones.
- Cumple con las reglas de negocio definidas.

### 3. Automatización con Selenium WebDriver y Python

NobleProg

#### Creando una prueba unitaria

##### Patrón de diseño POM (Page Object Model)

Es un patrón de diseño que propone crear una clase por cada página web que automatizas.

Cada clase encapsula:

- Los **elementos** de la página (campos, botones, etc.).
- Las **acciones** que se pueden realizar sobre ellos (click, escribir, validar).

Esto desacopla la lógica de interacción del test, mejora la reutilización y facilita el mantenimiento.

### 3. Automatización con Selenium WebDriver y Python

#### Creando una prueba unitaria

NobleProg

#### Ciclo de vida de una función de prueba

```
class MiTest(unittest.TestCase):
    def setUp(self):
        # 1. Inicializa recursos antes de cada prueba
        print("🔧 setUp: inicializando entorno")

    def test_funcion(self):
        # 2. Lógica que quieres validar
        print("🚀 test_funcion: ejecutando prueba")

    def tearDown(self):
        # 3. Libera recursos después de cada prueba
        print("🧹 tearDown: limpiando entorno")
```

**unittest.TestCase**: Hereda el comportamiento del framework de testing estándar de Python **unittest**. Y podemos usar métodos de aserción como `assertEqual`, `assertTrue`, etc.

**self** es una referencia a la instancia actual de la clase (en Java o C#, el equivalente es **this**)

### 3. Automatización con Selenium WebDriver y Python

#### Creando una prueba unitaria

NobleProg

##### Practica:

- Crear nuevo proyecto en PyCharm
- Crear la siguiente estructura
  - pages\login\_page.py
  - tests\test\_login.py
- Crear una prueba unitaria para validar el inicio de sesión de esta página:  
<https://www.saucedemo.com/>

### 3. Automatización con Selenium WebDriver y Python

NobleProg

#### Creando una prueba unitaria

##### Funciones lamda:

Las funciones Lambda son funciones anónimas que solo pueden contener una expresión.

La sintaxis de una función lambda es: **lambda args: expresión**.

- Primero se escribe la palabra clave **lambda**
- Después los argumentos que necesites separados por coma
- Dos puntos :
- Y por último la expresión que será el cuerpo de la función.

##### Ejemplo:

```
lambda x: x > 0
```

```
mi_lista = [18, -3, 5, 0, -1, 12]
lista_nueva = list(filter(lambda x: x > 0, mi_lista))
print(lista_nueva) # [18, 5, 12]
```

### 3. Automatización con Selenium WebDriver y Python

NobleProg

#### Creando una prueba unitaria

Producto a agregar:

- "Sauce Labs Backpack"

Funcion **lamda**:

- Devuelve una tupla de localización que Selenium puede usar:

```
button = lambda name: (
    By.XPATH, f"//div[text()='{name}']/ancestor::div[@class='inventory_item']//button[text()='Add to cart']")
)
```

El XPath generado busca:

1. Un <div> cuyo texto sea exactamente el nombre del producto.
2. Luego sube al contenedor padre con ancestor::div[@class='inventory\_item'] .
3. Dentro de ese contenedor, busca el <button> correspondiente.

### 3. Automatización con Selenium WebDriver y Python

NobleProg

#### Accediendo a una base de datos

Muchas veces necesitamos validar datos contra la base de datos, por ejemplo:

- Verificar que un usuario existe antes de probar login.
- Validar que un registro se creó después de una acción en la UI.
- Reutilizar datos dinámicos (ej. credenciales de prueba)

La conexión a BD se hace con librerías específicas:

- sqlite3 (para SQLite)
- pyodbc (para SQL Server)
- mysql-connector-python (para MySQL)

### 3. Automatización con Selenium WebDriver y Python

#### Accediendo a una base de datos

NobleProg

La estructura propuesta para el proyecto es:

```
tests/  
    └── test_login.py      # unittest con casos de prueba  
pages/  
    └── login_page.py     # POM: elementos y acciones de la página  
utils/  
    └── db_helper.py       # Clase para conexión y queries
```

### 3. Automatización con Selenium WebDriver y Python

NobleProg

#### Accediendo a una base de datos

##### Practica:

- Crear nuevo proyecto en PyCharm
- Crear la siguiente estructura
  - pages\login\_page.py
  - tests\test\_login.py
  - utils\db\_helper.py
- Crear una prueba unitaria para validar el inicio de sesión de esta página, con credenciales obtenidas desde base de datos: <https://www.saucedemo.com/>

## 4. Escalabilidad, Prácticas Avanzadas y Cierre

- Desarrollar un marco de prueba
- Ejecución de suites de prueba contra múltiples navegadores
- Integración continua con Jenkins

## 4. Escalabilidad, Prácticas Avanzadas y Cierre

NobleProg

### Desarrollar un marco de prueba

#### Marco de prueba (Test Framework):

Un marco de prueba es un conjunto de **herramientas, librerías, buenas prácticas y patrones** que definen cómo estructurar, organizar y ejecutar pruebas automatizadas.

No es solo código, sino una arquitectura para pruebas que busca orden, escalabilidad y mantenibilidad.

# 4. Escalabilidad, Prácticas Avanzadas y Cierre

NobleProg

## Desarrollar un marco de prueba

### Objetivos principales

- Estandarizar cómo se escriben y ejecutan las pruebas.
- Separar responsabilidades (datos, lógica de negocio, casos de prueba).
- Reutilizar código (evitar duplicación de pasos).
- Mejorar mantenibilidad (si cambia la app, modificamos solo una parte).
- Integración continua (ejecución automática en CI/CD).

# 4. Escalabilidad, Prácticas Avanzadas y Cierre

NobleProg

## Desarrollar un marco de prueba

Componentes típicos de un Test Framework

- Gestión de pruebas
  - Framework de pruebas base (ej. unittest, pytest, TestNG, JUnit).
  - Manejo de setup/teardown.
- Automatización de UI / API
  - Selenium.
- Patrón de diseño
  - Page Object Model (POM), encapsula lógica de las páginas.
- Gestión de datos de prueba
  - Base de datos (SQL Server, MySQL, SQLite).

# 4. Escalabilidad, Prácticas Avanzadas y Cierre

## Desarrollar un marco de prueba

**NobleProg**

- Utilidades y helpers
  - Conexión a BD (pyodbc, sqlite3).
  - Esperas explícitas (WebDriverWait).
- Integración con CI/CD
  - Jenkins, GitHub

## 4. Escalabilidad, Prácticas Avanzadas y Cierre

### Ejecución de suites de prueba contra múltiples navegadores

NobleProg

Es la capacidad de ejecutar la misma suite de pruebas automatizadas en diferentes navegadores web (Chrome, Firefox, Edge, Safari, etc.), para asegurar que la aplicación funciona correctamente sin importar el navegador del usuario. Esto se conoce como Cross-Browser Testing.

#### ¿Por qué es importante?

- Cada navegador interpreta HTML, CSS y JavaScript de forma ligeramente distinta.
- Garantiza compatibilidad y experiencia de usuario consistente.
- Detecta errores específicos de navegador

## 4. Escalabilidad, Prácticas Avanzadas y Cierre

NobleProg

### Ejecución de suites de prueba contra múltiples navegadores

#### Estrategias para ejecutar en múltiples navegadores

- Localmente con WebDriver
  - Parametrizar las pruebas para cambiar de driver.
- Ejecución paralela en Selenium Grid
  - Permite distribuir pruebas en nodos con diferentes navegadores.
  - Acelera ejecución (ejecuta en paralelo).
- Nube (BrowserStack, Sauce Labs, LambdaTest)
  - Infraestructura lista con cientos de combinaciones de navegadores, versiones y dispositivos.
  - Ideal para pruebas en móviles y navegadores antiguos.

# 4. Escalabilidad, Prácticas Avanzadas y Cierre

NobleProg

## Trabajando con Selenium Grid

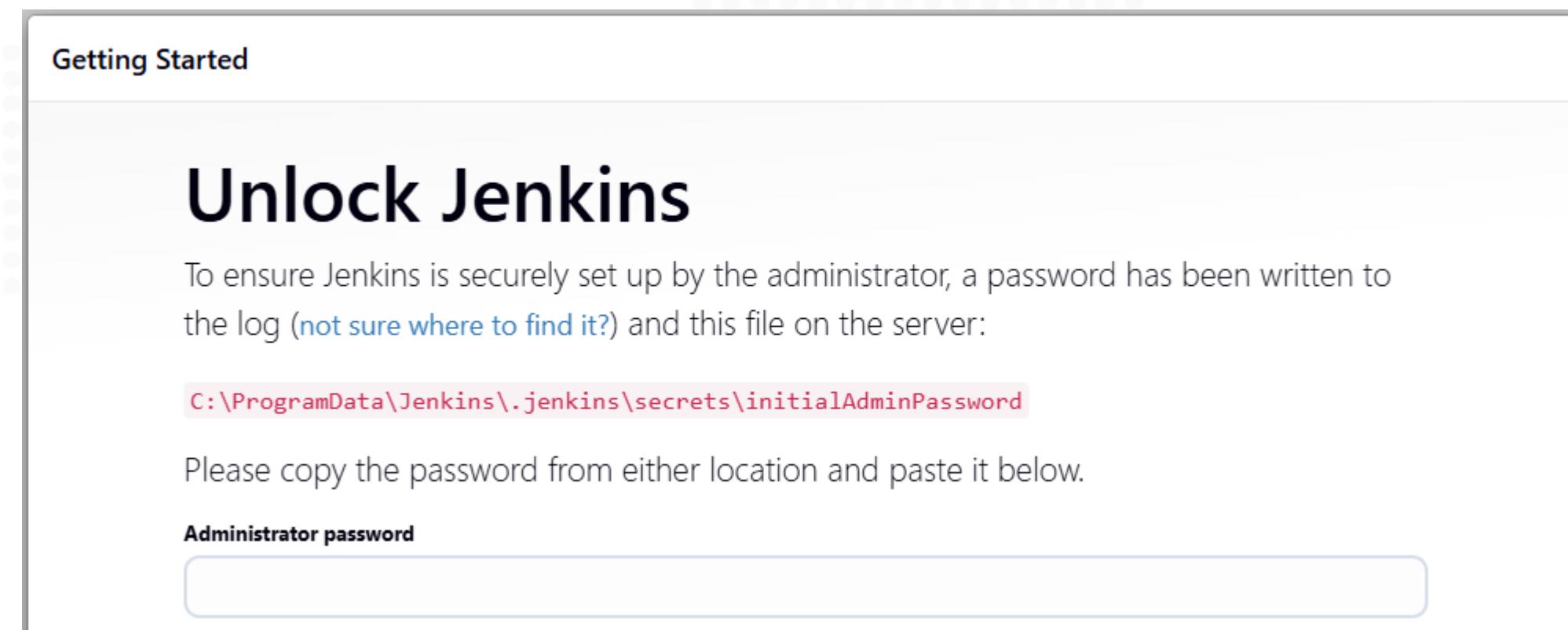
### Instalar Jenkins (en Windows o Linux)

#### ► En Windows (instalador)

1. Descarga el instalador desde: <https://www.jenkins.io/download/>
2. Ejecuta el instalador y sigue el asistente.
3. Una vez instalado, Jenkins se ejecutará como un servicio en <http://localhost:8080>

### Primer acceso y desbloqueo

- La primera vez que accedes a Jenkins, te pedirá un token de desbloqueo.



# 4. Escalabilidad, Prácticas Avanzadas y Cierre

## Trabajando con Selenium Grid

NobleProg

### Plugins recomendados

Desde "Administrar Jenkins > Administrar plugins" instala los siguientes:

- Git Plugin
- ShiningPanda

New Item

Enter an item name

Select an item type

 Freestyle project  
Classic, general-purpose job type that checks out from up to one SCM, executes build steps serially, followed by post-build steps like archiving artifacts and sending email notifications.

 Pipeline  
Orchestrates long-running activities that can span multiple build agents. Suitable for building pipelines (formerly known as workflows) and/or organizing complex activities that do not easily fit in free-style job type. →

 Multi-configuration project  
Suitable for projects that need a large number of different configurations, such as testing on multiple environments, platform-specific builds, etc.

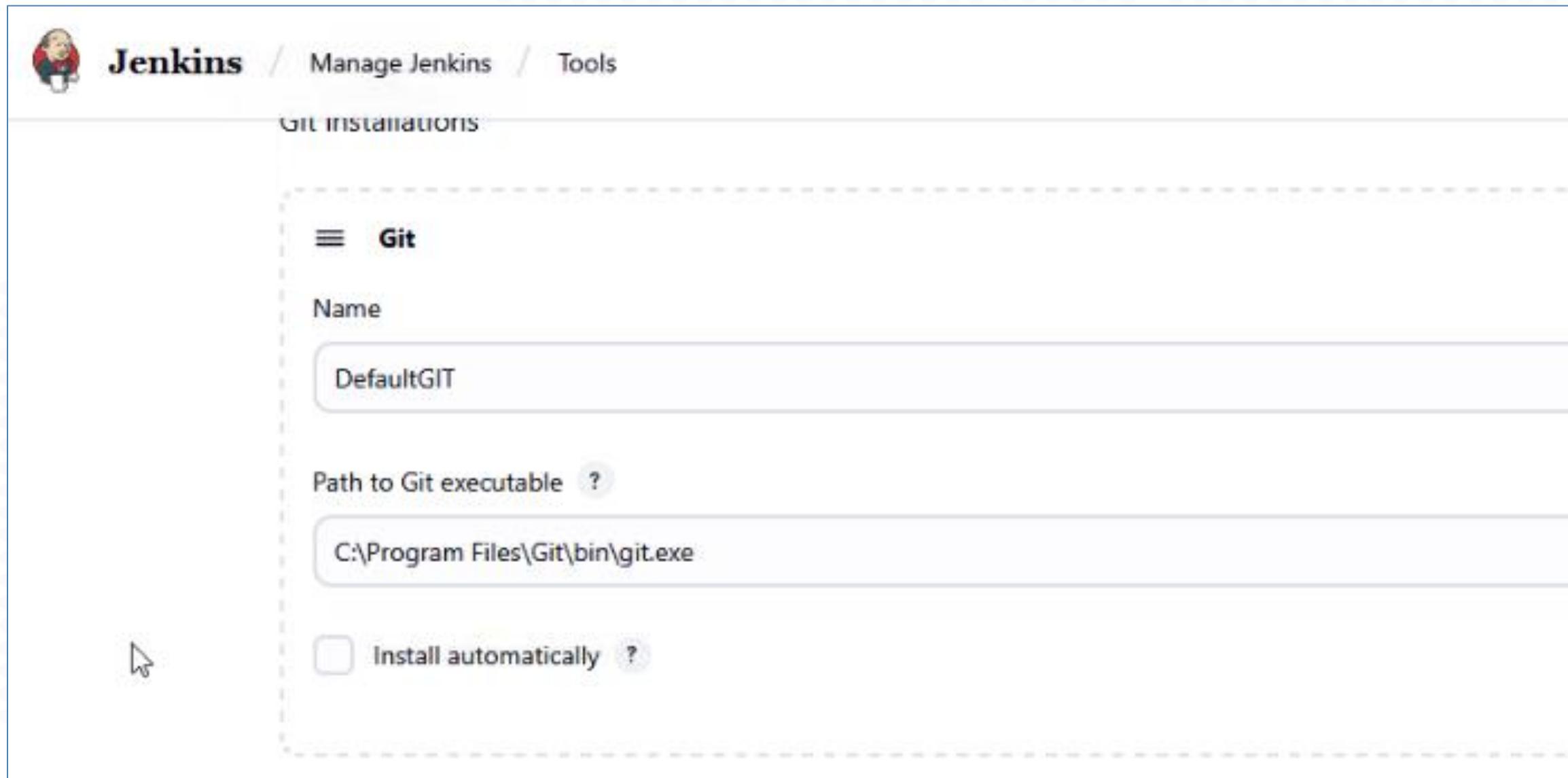
# 4. Escalabilidad, Prácticas Avanzadas y Cierre

## Trabajando con Selenium Grid

NobleProg

### Configuración

#### Instalar y configurar Git



# 4. Escalabilidad, Prácticas Avanzadas y Cierre

## Trabajando con Selenium Grid

NobleProg

### Configuración

The screenshot shows the Jenkins dashboard with the following sections:

- Build Queue:** Shows "No builds in the queue."
- Build Executor Status:** Shows 0/2 executors available.
- Build History:** A table with columns: S (Status), W (Worker), Name, Last Success, Last Failure, and Last Duration.

S	W	Name	Last Success	Last Failure	Last Duration
Green checkmark	Cloud icon	Login_Pipeline	19 min #49	25 min #46	14 sec
Green checkmark	Sun icon	PipelineLogin	4 min 42 sec #2	N/A	14 sec

# ¿Por qué elegirnos otra vez?



## Impulsamos tu éxito

Ofrecemos soluciones integrales en capacitación y consultoría para alcanzar tus metas.

## Presencia global

Estamos donde tú estás. Representación y servicio en tu país y a nivel mundial.

## Flexibilidad total

Nos adaptamos a tus necesidades, brindándote exactamente lo que requieres.

## Optimización garantizada

Te ayudamos a optimizar tus procesos de capacitación para obtener resultados superiores.

## Respuestas rápidas

¡Contáctanos y recibe respuestas rápidas y eficientes siempre!

iDescubre cómo transformar tu potencial con nuestros cursos especializados!  
Para visualizar nuestro catálogo de cursos, **visita nuestro sitio web,**  
**selecciona tu país y encuentra la capacitación perfecta para ti.**

[www.nobleprog.mx](http://www.nobleprog.mx)

