Prova: Python e Selenium

Parte 1: Questões Teóricas (5 questões)

1. **Explique a diferença entre Selenium IDE e Selenium WebDriver. (2 pontos)**

Selenium IDE

- **Objetivo Principal:** Ferramenta voltada para gravação e reprodução de testes automáticos, sem necessidade de programação.
- **Características:**
    - Foco em automação simples e prototipagem rápida.
    - A interface gráfica permite gravar interações no navegador e reproduzir como scripts.
    - Não oferece suporte a testes avançados ou dinâmicos.
- **Exemplo de Aplicação:** Gravar um teste básico para verificar se o botão de login de um site está funcionando corretamente.

Selenium WebDriver

- **Objetivo Principal:** Biblioteca que permite criar scripts personalizados de automação usando linguagens de programação.
- **Características:**
    - Foco em automação de testes complexos e robustos.
    - Interage diretamente com os navegadores, proporcionando maior controle e flexibilidade.
    - Requer conhecimentos de programação, ideal para integração com frameworks.
- **Exemplo de Aplicação:** Criar um fluxo automatizado de compra em uma loja virtual, preenchendo formulários e validando mensagens de erro.

Diferenças-Chave:

1. **Nível de Complexidade:** Selenium IDE é mais simples e visual, enquanto WebDriver é poderoso, mas exige programação.
2. **Capacidades:** Selenium IDE é limitado a testes básicos, enquanto o WebDriver suporta testes avançados e dinâmicos.
3. **Casos de Uso:** IDE é ideal para iniciantes e protótipos rápidos; WebDriver é voltado para automações robustas e personalizadas.

2. Quais são os principais tipos de localizadores (locators) usados no Selenium WebDriver para encontrar elementos na página? Explique dois deles. (2 pontos)

Principais tipos de localizadores no Selenium WebDriver:

1. **ID:** Usa o atributo único `id` para localizar elementos. É rápido e preciso.
    - **Exemplo:** `driver.findElement(By.id("username"))`

2. **CSS Selector:** Busca elementos com base em seletores CSS, útil para estruturas complexas.
    - **Exemplo:** `driver.findElement(By.cssSelector("input[type='password']"))`

Outros tipos de localizadores incluem: Name, Tag Name, Link Text, Partial Link Text, CSS Selector, e XPath, cada um com finalidades e níveis de precisão diferentes.

3. O que é um WebElement no Selenium? Dê um exemplo de como interagir com um WebElement usando Python. (2 pontos)

No Selenium, um **WebElement** representa um elemento HTML único em uma página web, como botões, campos de texto, links ou tabelas. Esse objeto é retornado ao localizar o elemento com um dos localizadores disponíveis (como `id`, `name`, ou `CSS selector`). A partir dele, podemos realizar várias ações ou recuperar informações.

Principais métodos para interagir com um WebElement:

- **`click()`**: Realiza um clique no elemento, como em um botão.
- **`send_keys()`**: Insere texto em campos de entrada.
- **`get_attribute()`**: Recupera o valor de um atributo do elemento.
- **`text`**: Obtém o texto exibido dentro do elemento.

Exemplo em Python

Aqui está como interagir com um WebElement usando o Selenium WebDriver em Python:

```
from selenium import webdriver


# Iniciar o WebDriver (neste caso, usando o Chrome)
```

```
driver = webdriver.Chrome()

# Abrir uma página da web
driver.get("https://example.com")

# Localizar um campo de entrada por ID e interagir com ele
campo_texto = driver.find_element("id", "example-id")
campo_texto.send_keys("Texto de exemplo")  # Enviar texto para o campo

# Localizar um botão e clicar nele
botao = driver.find_element("id", "submit-button")
botao.click()  # Clicar no botão

# Fechar o navegador
driver.quit()
```

4. No Selenium WebDriver, o que acontece se você tentar interagir com um elemento que ainda não está visível ou carregado na página? Qual comando você usaria para resolver isso? (2 pontos)

Quando você tenta interagir com um elemento que ainda não foi carregado ou não está visível na página, o Selenium WebDriver pode lançar exceções, como:

- **ElementNotInteractableException**: O elemento foi encontrado na página, mas não está visível ou não é interativo (por exemplo, um botão oculto).
- **NoSuchElementException**: O Selenium não consegue encontrar o elemento na página, possivelmente porque ele ainda não foi carregado ou não está na DOM.

Solução: Espera explícita

Para evitar essas exceções, você deve garantir que o elemento esteja visível e interativo antes de tentar interagir com ele. Isso pode ser feito usando **WebDriverWait** em conjunto com **ExpectedConditions**, que permite aguardar até que a condição desejada seja atendida.

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

# Inicializa o WebDriver
driver = webdriver.Chrome()

# Navega até a página desejada
driver.get("https://www.example.com")

# Espera até que o botão esteja visível antes de clicar
botao = WebDriverWait(driver, 10).until(
    EC.visibility_of_element_located((By.id, "botao-id"))
)

# Agora o botão pode ser clicado com segurança
botao.click()

# Fecha o navegador
driver.quit()
```

Explicação:

1. **WebDriverWait**: Espera até que uma condição seja atendida (no exemplo, até o botão ficar visível).
2. **EC.visibility_of_element_located**: Verifica se o elemento está visível na página.
3. Após garantir que o elemento está visível, podemos interagir com ele normalmente (por exemplo, clicando no botão).

5. Cite duas limitações do Selenium IDE que podem levar à escolha do Selenium WebDriver em projetos maiores. (2 pontos)

1. **Falta de Suporte para Testes Avançados:** O Selenium IDE é uma ferramenta focada em gravação e reprodução, ideal para automação simples e testes básicos. No entanto, ele não é adequado para testes mais avançados, como:
    - Interações com múltiplas janelas ou abas.
    - Execução de loops ou testes condicionais complexos.
    - Criação de testes modulares ou integrados com frameworks de automação de testes.
2. Em projetos maiores, onde testes mais complexos e personalizados são necessários, o **Selenium WebDriver** é a melhor escolha, pois permite a escrita de scripts em várias linguagens de programação (como Java, Python ou C#), o que proporciona mais controle e flexibilidade.
3. **Execução Limitada a Navegadores e Ambientes Específicos:** O Selenium IDE só funciona em navegadores específicos, principalmente o Chrome e o Firefox, e não é capaz de rodar testes em diferentes tipos de ambientes (como dispositivos móveis ou navegadores variados).
   O **Selenium WebDriver**, por outro lado, oferece uma solução mais robusta e multiplataforma, podendo ser executado em diferentes navegadores e sistemas operacionais. Ele também pode ser integrado a ferramentas como **Selenium Grid**, o que permite a execução paralela de testes em diferentes ambientes, ampliando a escalabilidade e a cobertura de testes.