

# 웹 요소 탐색과 조작



# 테스트 자동화의 핵심은 정확한 요소 선택이다

- 웹 자동화 테스트는 사람이 하는 반복적인 UI 테스트를 대신 수행하는 것
- 정확한 요소를 찾지 못하면 테스트가 실패하거나 오작동 발생
- 웹사이트가 업데이트될 때에도 견고한 자동화 스크립트를 작성하는 것이 중요
- 동적 페이지, AJAX 로딩 요소 등 다양한 환경에서 올바른 요소 탐색 필요



## 다음과 같은 경우, 어떤 문제점이 발생할까?

- ID 값이 변경되는 로그인 버튼
- 페이지가 로딩되지 않은 상태에서 탐색을 시도
- AJAX로 동적 로딩되는 데이터가 아직 보이지 않는 상태



# 자동화에서 최적의 요소 탐색 방법 선택하기

## CSS Selector vs XPath

방법	특징	장점	단점
<b>CSS Selector</b>	스타일을 정의하는 방식과 동일한 문법으로 요소 선택	빠르고 간결한 문법	복잡한 요소 선택이 어려움
<b>XPath</b>	XML 기반의 경로 탐색 방식으로 요소 찾기	복잡한 요소도 정교하게 선택 가능	성능이 CSS Selector보다 느릴 수 있음

### 같은 요소를 CSS Selector와 XPath로 찾는 방법 비교

```
# CSS Selector 예제 (빠르고 간결)
element = driver.find_element(By.CSS_SELECTOR, "button.login")

# XPath 예제 (좀 더 유연한 탐색)
element = driver.find_element(By.XPATH, "//button[@class='login']")
```



# Selenium이 제공하는 주요 탐색 방법

코드 예제	특징
<code>driver.find_element(By.ID, "login")</code>	ID 기반 탐색 (빠름)
<code>driver.find_element(By.CLASS_NAME, "btn")</code>	클래스명 기반 탐색
<code>driver.find_element(By.NAME, "user_email")</code>	입력 필드 같은 요소 탐색
<code>driver.find_element(By.TAG_NAME, "h1")</code>	특정 태그의 요소 찾기
<code>driver.find_element(By.LINK_TEXT, "로그인")</code>	링크 텍스트로 탐색
<code>driver.find_element(By.PARTIAL_LINK_TEXT, "회원")</code>	부분 일치 링크 텍스트 탐색
<code>driver.find_element(By.CSS_SELECTOR, ".btn-primary")</code>	CSS 선택자 탐색
<code>driver.find_element(By.XPATH, "//button[@id='login'])")</code>	XPath를 활용한 정확한 탐색

- ID가 있는 경우 : `find_element(By.ID, "element_id")`
- 여러 개의 동일한 요소가 있을 경우 : `find_elements(By.CLASS_NAME, "btn")`
- 특정 텍스트를 포함하는 요소 찾기: `find_element(By.XPATH, "//p[contains(text(), '로그인')]")`



## 자동화에서 요소 탐색 오류를 방지하는 방법



**NoSuchElementException** : 찾으려는 요소가 존재하지 않음



요소가 올바르게 로드되었는지 확인 (WebDriverWait 활용)



**StaleElementReferenceException** : 요소가 갱신되어 더 이상 유효하지 않음



`find_element()`를 다시 실행하여 새로운 요소를 가져옴



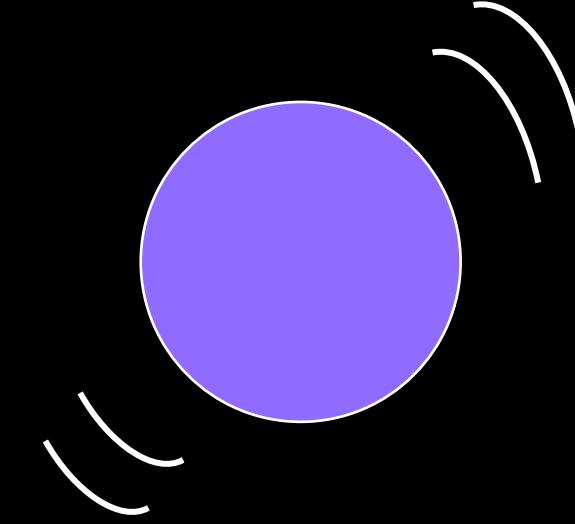
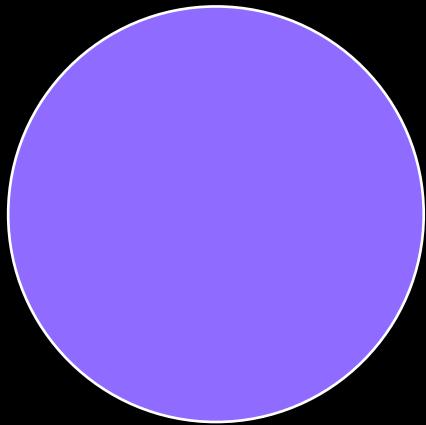
**ElementNotInteractableException** : 요소가 비활성화 상태로 클릭 또는 입력 불가



`execute_script()`를 사용하여 요소 활성화 후 실행



# 정적 요소와 동적 요소를 다루는 최적의 방법



## 정적 요소

- 페이지 로딩 시 바로 생성됨
- `find_element()`로 바로 탐색 가능

## 동적 요소

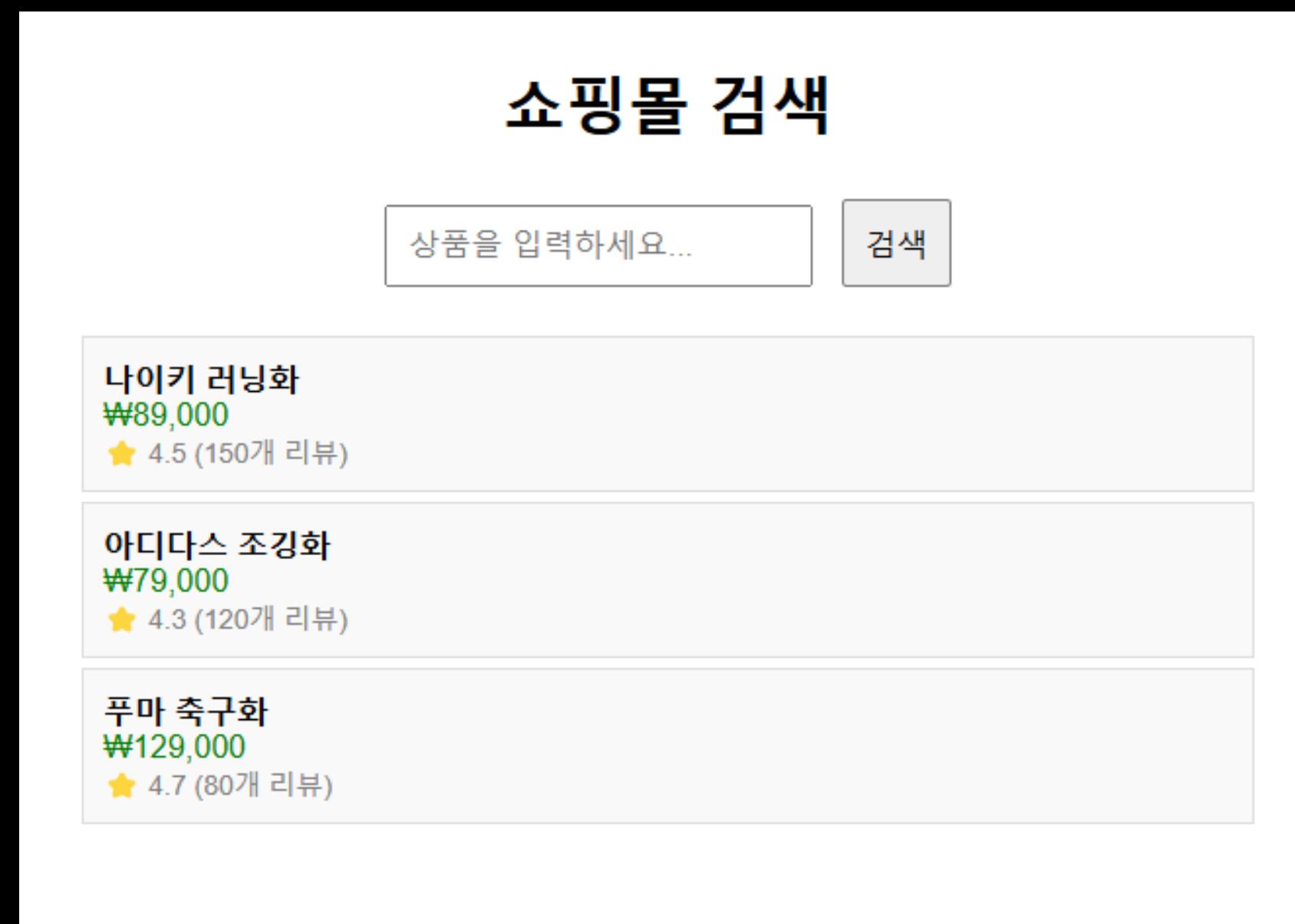
- AJAX, JavaScript 등으로 로딩됨
- `WebDriverWait`을 활용하여 요소가 나타날 때까지 대기

## 동적 요소를 다룰 때 필수적인 전략

- `WebDriverWait`을 활용하여 요소가 나타날 때까지 기다리기
- '`execute_script()`'를 사용하여 동적으로 생성된 요소를 조작하기
- '`find_elements()`'를 사용하여 동적 목록을 처리하고 필요한 데이터 추출하기

## 프로젝트1 소개 - index.html

- Selenium을 활용하여 웹사이트의 검색 기능을 자동화하는 방법 학습
- CSS Selector와 XPath를 활용하여 검색 결과에서 필요한 데이터를 추출하는 방법 이해
- 검색 후 상품 상세 정보가 올바르게 표시되는지 자동 검증 ('assert' 활용) 학습
- 'find\_element()'와 'find\_elements()'의 차이점을 학습하며, 다양한 요소 탐색 기법 익히기





# 텍스트를 입력하고 검색 버튼을 자동으로 클릭

## 웹 자동화에서 입력 필드와 버튼 조작의 중요성

- 사용자의 입력을 테스트하기 위해 입력 필드와 버튼을 자동화하는 것이 필수
- 검색, 로그인, 회원가입과 같은 기능은 Selenium을 이용한 UI 테스트에서 가장 많이 사용됨

## 실행 흐름 요약

1. 검색창 요소 찾기  
(find\_element())

2. 검색어 입력  
(send\_keys())

3. 검색 버튼 요소 찾기  
(find\_element())

4. 검색 버튼 클릭  
(click())



# 검색 결과에서 원하는 정보를 추출하는 방법

## 검색 결과를 크롤링하는 이유

- UI 테스트의 핵심은 검색 결과가 올바르게 표시되는지 확인하는 것
- 검색 후 상품명이 정확한지, 가격이 정상적으로 보이는지 자동 검증 필요

## 실행 흐름

```
# 검색 결과 가져오기
products = driver.find_elements(By.CLASS_NAME, "product-item")

# 첫 번째 상품 정보 가져오기
first_product = products[0]
product_name = first_product.find_element(By.CLASS_NAME, "product-title").text
product_price = first_product.find_element(By.CLASS_NAME, "product-price").text
product_review = first_product.find_element(By.CLASS_NAME, "product-review").text
```

```
DevTools listening on ws://127.0.0.1:9716/devtools/browser/0de9e048-6e3a-4868-8c0b-bdb2ba64ffea
상품명: 나이키 러닝화, 가격: ₩89,000, 리뷰 수: ★ 4.5 (150개 리뷰)
Enter를 누르면 창이 닫힙니다...
```



# 검색 결과 검증 (assert 활용)

## 자동 검증의 필요성

- UI 테스트 자동화에서는 사람이 직접 화면을 확인할 필요 없이 코드로 자동 검증 가능
  - 'assert'를 활용하면 예상 결과와 실제 결과를 비교하여 테스트의 성공 여부를 판별할 수 있음

- 검색된 상품 개수가 최소 1개 이상인지 확인
  - 첫 번째 검색 결과의 제목이 비어 있지 않은지 확인
  - 가격이 ₩ 기호로 시작하는지 확인

# 실습1 : 쇼핑몰 사이트 자동화 테스트

## 실습 과정

### 1. 특정 카테고리 상품 검색 및 정보 확인

- ✓ 검색창에 `“축구화”`를 입력하고 검색 버튼을 클릭하세요.
- ✓ 검색 결과에서 첫 번째 상품의 이름, 가격, 리뷰 개수를 가져오세요.
- ✓ 가져온 상품 정보를 터미널에 출력하세요.

 검색 결과  
- 상품명: 푸마 축구화  
- 가격: ₩129,000  
- 리뷰 수: ★ 4.7 (80개 리뷰)

### 2. 검색된 상품 개수 검증 (assert 활용)

- ✓ 검색 후, `display: block;` 상태인 상품 개수가 1개 이상인지 검증하세요.
- ✓ `assert`를 사용하여 검색된 상품이 없을 경우 오류 메시지를 출력하도록 구현하세요.
- ✓ 상품 가격이 "₩"로 시작하는지 확인하세요..

```
assert len(visible_products) > 0, "✖ 검색 결과가 없습니다!"  
^^^^^^^^^^^^^^^^^^^^^
```

```
AssertionError: ✖ 검색 결과가 없습니다!
```



# 버튼을 자동으로 클릭하고, 동작을 검증하자

기본적인 click() 사용법

```
button = driver.find_element(By.ID, "submit_btn")
button.click()
```

클릭 후 UI가 변경되었는지 확인 (is\_displayed())

```
message = driver.find_element(By.ID, "message")

if message.is_displayed():
    print("✓ 버튼 클릭 후 메시지가 정상적으로 표시되었습니다!")
else:
    print("✗ 메시지가 보이지 않습니다!")
```

클릭 전에 버튼이 활성화되어 있는지 확인 (is\_enabled())

```
button = driver.find_element(By.ID, "submit_btn")

if button.is_enabled():
    button.click()
else:
    print("✗ 버튼이 비활성화 상태입니다!")
```

## 버튼 클릭 테스트

클릭하세요

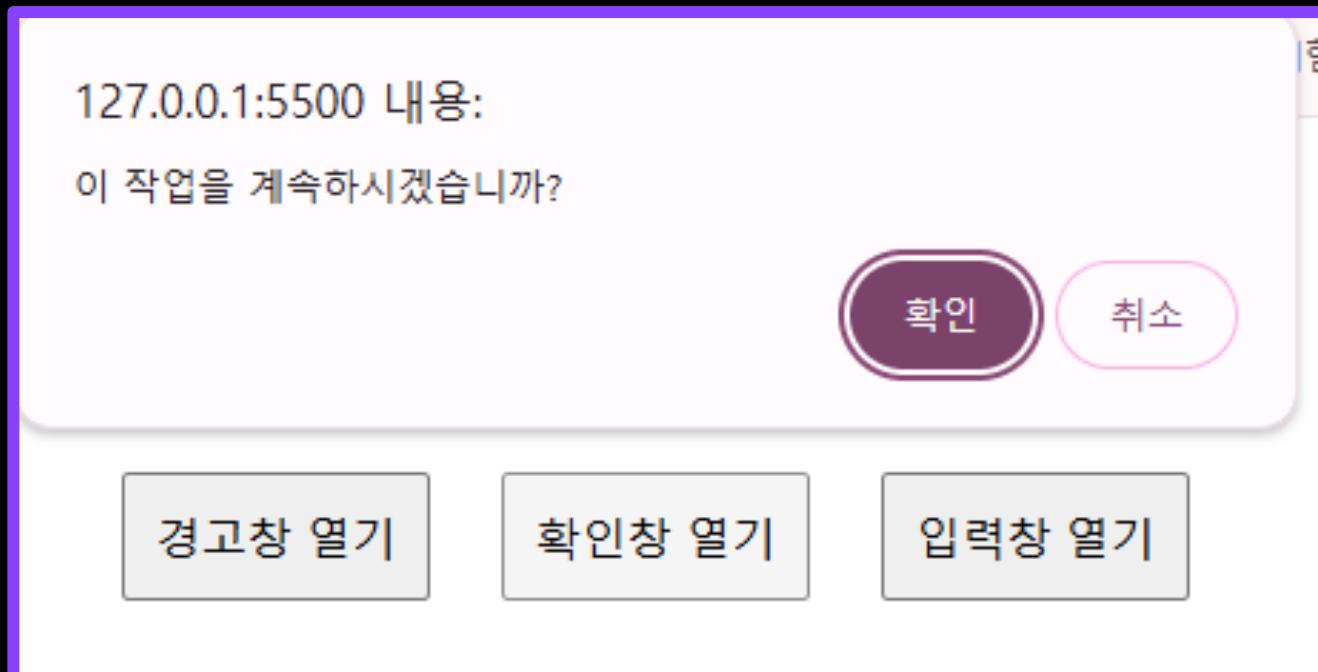
✓ 버튼이 정상적으로 클릭되었습니다!



# 팝업 창을 자동으로 처리하자

## JavaScript 팝업 창

- 웹사이트에서 JavaScript를 이용해 브라우저가 생성하는 대화형 팝업 창을 의미한다.
- 이 팝업은 사용자의 응답을 요구하는 UI 요소이며, 보통 다음과 같은 종류가 있다.



팝업 종류	특징	예제
<b>alert()</b>	확인 버튼만 있는 경고창	"경고: 이 작업은 취소할 수 없습니다!"
<b>confirm()</b>	확인/취소 버튼 제공	"이 작업을 계속하시겠습니까?"
<b>prompt()</b>	입력 필드 제공	"이름을 입력하세요."

## Selenium에서 팝업 창을 다루는 방법 (`switch_to.alert`)

Selenium은 팝업 창을 제어할 때 '`switch_to.alert`' 객체를 사용한다.

- `accept()` : "확인" 버튼 클릭
- `dismiss()` : "취소" 버튼 클릭
- `send_keys("입력값")` : `prompt()`에 값 입력



# 숨겨진 UI 요소를 조작하는 방법을 배워보자

## 숨겨진 요소

- 웹사이트에서 일부 UI 요소가 기본적으로 숨겨져 있고, 특정 이벤트가 발생해야 보이도록 설계된 경우가 많다.
- 이러한 요소들은 Selenium에서 바로 조작이 어렵기 때문에 'execute\_script()'를 사용해야 한다.

## Selenium에서 'execute\_script()'를 사용하는 이유

Selenium의 기본 동작 방식은 HTML 요소의 상태를 기반으로 조작하는 것이므로, 'display: none;' 상태인 요소에는 기본적으로 클릭하거나 입력할 수 없다.

```
# 숨겨진 입력창 찾기
hidden_input = driver.find_element(By.ID, "hidden_input")

# 요소를 강제로 보이게 만들기
driver.execute_script("arguments[0].style.display = 'block';", hidden_input)
```



# 로딩이 끝난 후 동적 요소를 자동으로 감지

## 동적 요소

일반적인 웹사이트는 HTML이 고정된 상태로 로드되지만, JavaScript를 활용해 동적으로 데이터를 불러오는 경우가 많음

### 동적 요소의 예시

- ✓ Ajax로 로딩되는 검색 결과
- ✓ 페이지가 로드된 후 나오는 팝업 창
- ✓ 버튼을 클릭한 후 새롭게 생성되는 요소

```
# 버튼이 활성화될 때까지 최대 10초 대기
wait = WebDriverWait(driver, 10)
button = wait.until(EC.element_to_be_clickable((By.ID, "dynamic_button")))
```

## WebDriverWait을 활용하여 동적 요소 대기하는 방법

Selenium에서는 'WebDriverWait'을 활용하여 특정 요소가 나타날 때까지 기다릴 수 있다. 이때, `expected_conditions` 모듈을 활용하여 다양한 조건을 지정할 수 있다.

- `element_to_be_clickable` : 요소가 클릭 가능해질 때까지 대기
- `visibility_of_element_located` : 요소가 화면에 표시될 때까지 대기
- `presence_of_element_located` : 요소가 HTML에 추가될 때까지 대기 (표시 여부 상관없음)

# 버튼 클릭 후 UI 변경 사항 자동 검증

## 버튼 클릭 후 UI 변경을 자동으로 검증해야 하는 이유

- 웹사이트에서는 버튼을 클릭한 후 페이지가 변경되거나, 새로운 요소가 나타나는 경우가 많다
- 테스트 자동화에서는 버튼을 클릭한 후, 실제로 기대한 변화가 발생했는지 확인하는 과정이 필요

## Selenium에서 UI 변경 사항을 검증하는 방법

1. `is_displayed()`: 요소가 화면에 표시되었는지 확인

```
element = driver.find_element(By.ID, "confirmation_message")
if element.is_displayed():
    print("✓ 메시지가 정상적으로 표시되었습니다!")
else:
    print("✗ 메시지가 표시되지 않았습니다.")
```

2. `is_enabled()`: 버튼이 활성화되어 있는지 확인

```
button = driver.find_element(By.ID, "submit_button")
if button.is_enabled():
    print("✓ 버튼이 활성화되었습니다!")
else:
    print("✗ 버튼이 비활성화 상태입니다.")
```

3. `text` 속성을 활용하여 UI 변경 확인

```
element = driver.find_element(By.ID, "status_message")
if "완료" in element.text:
    print("✓ 상태 메시지가 '완료'로 변경됨!")
else:
    print("✗ 상태 메시지가 올바르게 변경되지 않음.")
```

## 프로젝트2 소개 - index.html

- 운동 동호회 웹사이트에서 특정 운동을 검색하고 참가 신청을 자동화하는 과정 이해
- Selenium을 활용하여 검색창 입력, 버튼 클릭, UI 변경 사항 검증까지 전체적인 프로세스를 학습
- 동적으로 생성되는 요소를 감지하고 자동으로 조작하는 기술 습득 (WebDriverWait)
- 실제 웹 테스트 환경을 모방한 연습을 통해 Selenium의 실무 활용도를 높이기





# 검색어만 입력하면 끝!

## 검색 기능 자동화

웹사이트에서 정보를 찾기 위해 사용자는 검색창에 키워드를 입력하고 검색 버튼을 클릭하는 과정을 거친다. Selenium을 활용하면 이 과정을 자동으로 수행할 수 있으며, 테스트 자동화에서도 자주 활용된다.

- 검색창 자동 입력 (`send_keys()`)

```
search_box.send_keys("축구")
```

- 검색 버튼 클릭 (`click()`)

```
search_button.click()
```

- 검색 결과 확인 (`find_elements()`)

```
clubs = driver.find_elements(By.CLASS_NAME, "club-item")
print(f"검색 결과 {len(clubs)}개 발견됨!")
```

```
DevTools listening on ws://127.0.0.1:13266/devtools/browser/095f0b75-850a-4607-9052-899e07eea95d
✓ 검색 결과 1개 발견됨!
Enter를 누르면 창이 닫힙니다...■
```

# 참가 신청 버튼을 자동으로 눌러보자!

## 참가 신청 버튼 자동화

- 사용자가 웹사이트에서 특정 모임을 선택하고 참가 신청을 하는 과정은 일반적으로 버튼 클릭을 통해 이루어진다.
- Selenium을 활용하면 이 버튼을 자동으로 찾아 클릭하고, UI가 정상적으로 변경되었는지 검증할 수 있다.

- 참가 신청 버튼 클릭 (`click()`)

```
apply_button.click()
```

- 참가 신청 후 UI 변경 사항 검증 (`is_displayed()`)

```
if confirmation_message.is_displayed():
    print("✓ 참가 신청이 완료되었습니다!")
else:
    print("✗ 참가 신청이 정상적으로 처리되지 않았습니다.")
```

- 참가 신청 후 버튼 상태 변경 (`is_enabled()`)

```
if not apply_button.is_enabled():
    print("✓ 참가 신청 후 버튼이 비활성화됨!")
```



# 페이지 새로고침 후 참가 신청 상태 유지 확인

## UI 테스트에서 상태 유지 확인이 중요한 이유

웹사이트에서 사용자가 참가 신청 버튼을 클릭한 후, 페이지를 새로고침했을 때 신청 상태가 유지되지 않는다면, 이는 테스트 중 오류로 간주될 수 있다.

- **페이지 새로고침 후 참가 신청 상태 유지 확인**

```
driver.refresh()
```

- **JavaScript 기반 요소 상태 확인 (execute\_script())**

```
status = driver.execute_script("return document.getElementById('confirmation_message').style.display;")  
if status == "block":  
    print("✅ 참가 신청 상태가 유지됨!")
```

- **중복 신청 방지 기능 테스트 (click() 차단)**

```
if not apply_button.is_enabled():  
    print("✅ 중복 신청 방지 기능이 정상 작동 중!")
```



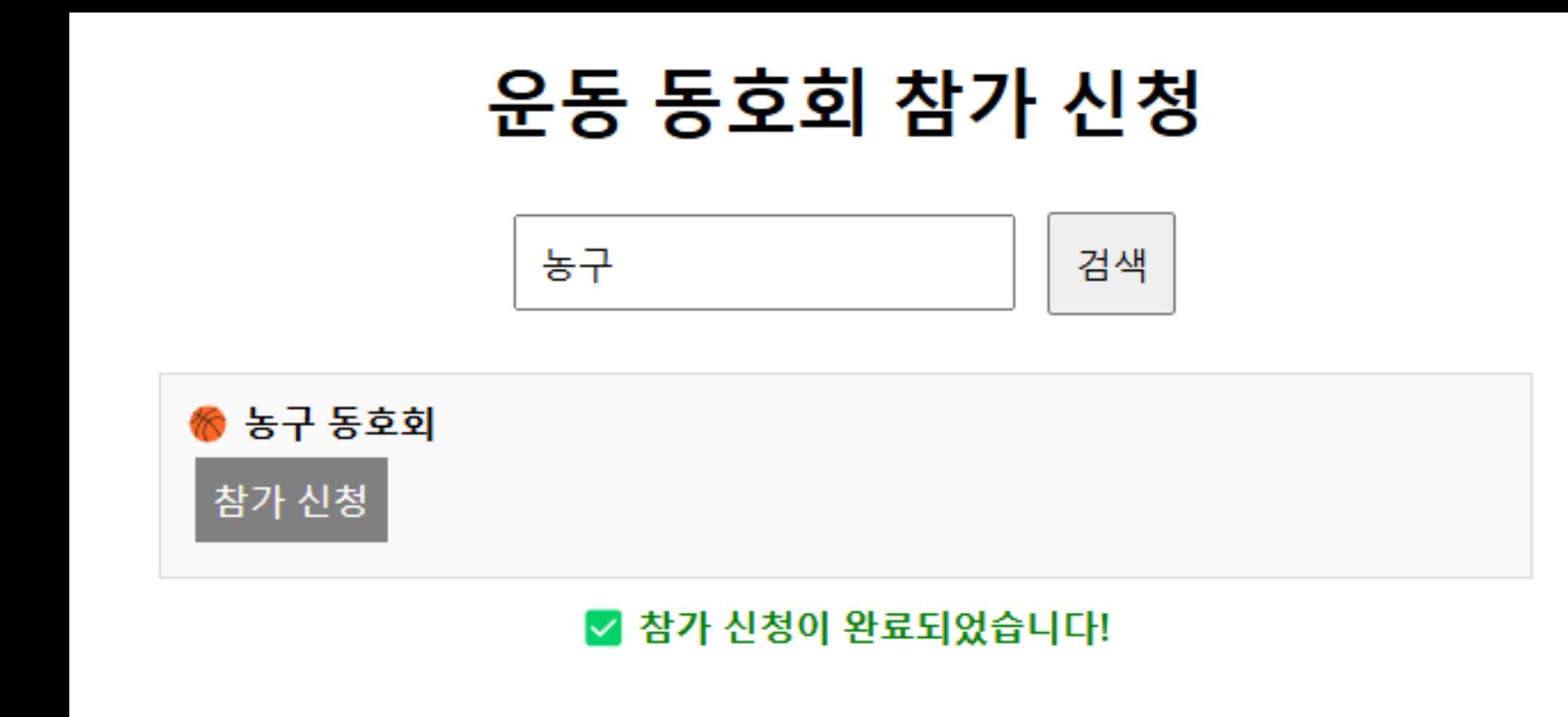
## 실습 2: 참가 신청 버튼 클릭 & 신청 완료 확인

### 실습 목표

- Selenium을 사용하여 검색어 입력 → 검색 버튼 클릭 → 참가 신청 버튼 클릭을 자동화
- 'display: block;' 상태인 동호회만 선택하여 참가 신청
- 참가 신청 완료 메시지가 정상적으로 표시되는지 검증

### 실습 과정

1. 검색창에 "농구" 입력
2. 검색 버튼 클릭
3. 첫 번째 동호회의 참가 신청 버튼 클릭
4. 참가 신청 완료 메시지가 정상적으로 표시되는지 확인



```
DevTools listening on ws://127.0.0.1:9229
✓ 참가 신청 버튼 클릭 성공!
✓ 참가 신청 완료 메시지 확인됨!
Enter를 누르면 창이 닫힙니다...
```

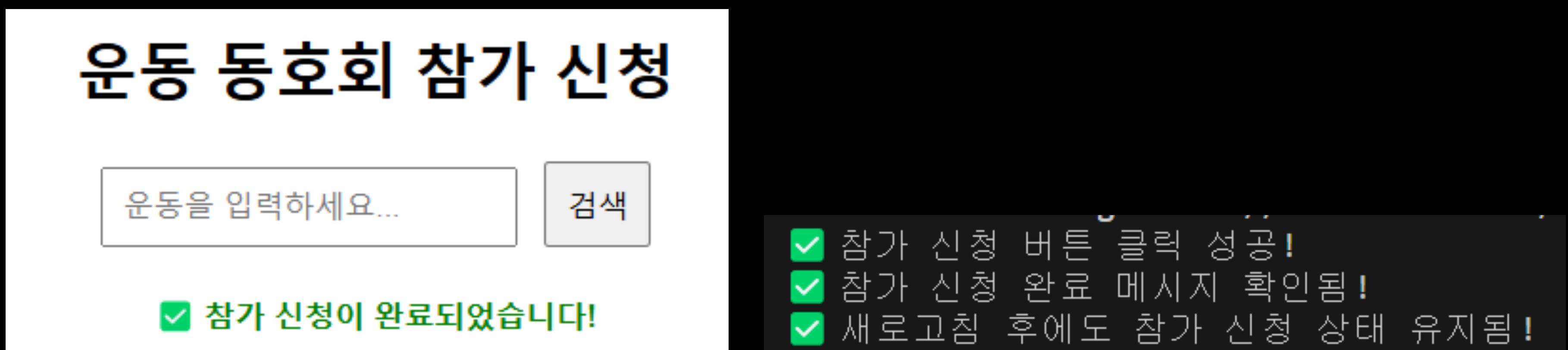
## ☰ 실습3 : 새로고침 후 참가 신청 상태 유지

### 실습 목표

- Selenium을 사용하여 참가 신청 후 페이지 새로고침 → 참가 신청 상태 유지 여부 확인
- 참가 신청 완료 메시지가 새로고침 후에도 유지되는지 검증

### 실습 과정

1. 페이지 새로고침
2. 새로고침 후 참가 신청 완료 메시지가 유지되는지 확인





## 퀴즈 타임



QUIZ 1번부터 8번을 풀어봐요!

# AJAX란? 비동기 데이터 로딩의 핵심!

## AJAX

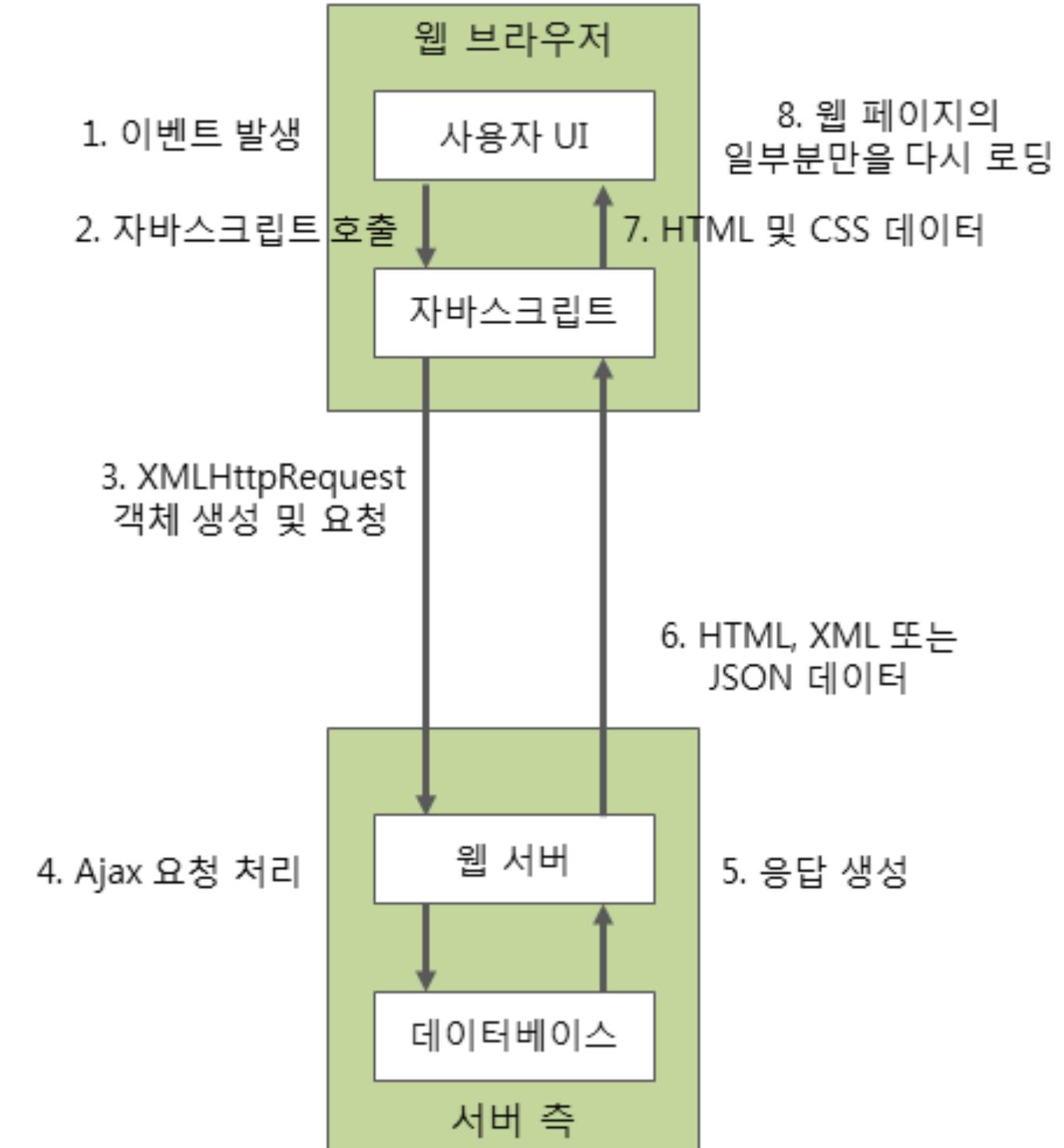
### (Asynchronous JavaScript and XML)

: 웹 페이지 전체를 새로고침하지 않고 일부 데이터만 갱신하는 기술

### AJAX 작동 방식

1. 클라이언트 요청 발생 ('XMLHttpRequest' 또는 'fetch()')
2. 서버가 데이터 응답 (JSON, XML 등)
3. 클라이언트가 받은 데이터를 즉시 웹 페이지에 반영

< Ajax를 이용한 웹 응용 프로그램의 동작 원리 >



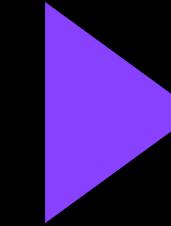


# AJAX 데이터 크롤링 최적화하기



문제

AJAX 데이터는 로딩 시간이 다 다른



해결 방안

WebDriverWait을 활용한 동적 요소 대기

## WebDriverWait

- 특정 요소가 나타날 때까지 기다리는 기능
- presence\_of\_element\_located(),  
visibility\_of\_element\_located() 등  
다양한 옵션 지원

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

wait = WebDriverWait(driver, 10)
element = wait.until(EC.presence_of_element_located((By.ID, "data")))
```

# 끝없는 데이터! 무한 스크롤 완전 정복

## 무한 스크롤

- 웹사이트에서 일정 부분까지 스크롤하면 자동으로 추가 데이터를 로드하는 방식
- AJAX를 사용하여 서버에서 추가 데이터를 가져와 화면에 표시
- 쇼핑몰, 뉴스, 소셜미디어(예: 인스타그램, 트위터)에서 자주 사용됨

```
# 스크롤 내리기 함수
def scroll_down():
    driver.execute_script("window.scrollTo(0, document.body.scrollHeight);")
    time.sleep(2) # 데이터 로딩 시간 고려

# 일정 횟수 반복하여 스크롤
for _ in range(5):
    scroll_down()

# 스크롤 후 데이터 확인
items = driver.find_elements(By.CLASS_NAME, "item")
print(f"총 {len(items)}개의 아이템을 가져왔습니다.")
```

## Selenium에서 무한 스크롤 처리하기

- 일반적인 `find_element()`로는 모든 데이터를 가져올 수 없음
- `execute_script()`를 활용하여 스크롤을 아래로 내리면서 데이터를 지속적으로 로드



# 보이지 않는 데이터까지 캐치하라

## AJAX 데이터 검증이 중요한 이유

- AJAX 요청은 즉시 반영되지 않고 일정 시간 후에 데이터를 로드
- 자동화 테스트 시 로딩이 완료되었는지 확인하는 과정 필요
- 동적 요소를 안정적으로 처리하지 않으면 'NoSuchElementException' 발생

## AJAX 데이터 크롤링 시 문제 해결



NoSuchElementException 발생



WebDriverWait을 사용하여 요소가 로드될 때까지 기다림



일부 데이터가 빠지는 문제



sleep()이 아닌 WebDriverWait으로 동적 요소 처리

# 데이터는 가져왔는데… 제대로 된 걸까?

## 데이터 검증이 필요한 이유

- 크롤링한 데이터가 실제 원하는 정보인지 확인 필요
- 일부 웹사이트는 데이터를 숨기거나 동적으로 처리
- get\_attribute()를 활용하여 데이터 정확성 체크 가능

## get\_attribute()

- Selenium에서 특정 HTML 요소의 속성(attribute) 값을 가져올 때 사용하는 메서드
- HTML 태그 내에 있는 id, class, title, href, data-\* 등의 속성 값을 가져오는 데 사용

```
item = driver.find_element(By.CLASS_NAME, "product")
title = item.get_attribute("title")
price = item.get_attribute("data-price")

print(f"상품 제목: {title}, 가격: {price}원")
```



## 프로젝트3 소개 – index.html

- Selenium을 활용하여 웹사이트에서 스터디 모임 일정을 자동으로 조회하고 신청
- AJAX로 데이터를 불러오는 사이트에서 동적 데이터를 처리하는 방법 학습
- 참가 신청 버튼을 자동으로 클릭하고 정상적으로 신청되었는지 검증

The screenshot shows a web application interface for managing study meetings. On the left, there is a sidebar with the title '스터디 모임 목록' (Study Meeting List) and a message indicating new meetings are being loaded. A large purple arrow points from this sidebar to the main content area on the right. The main content area also has a title '스터디 모임 목록'. It lists four study events:

- 파이썬 기초 스터디 (Python Basic Study) - Date: 2025-02-25, Join Application button
- 웹 개발 심화반 (Web Development Advanced Class) - Date: 2025-03-02, Join Application button
- 머신러닝 초급 (Machine Learning Beginner) - Date: 2025-03-10, Join Application button
- 데이터 분석 스터디 (Data Analysis Study) - Date: 2025-03-15, Join Application button



# AJAX 데이터, 직접 가져와보자

## 스터디 모임 목록은 어떻게 로딩될까?

- AJAX를 활용한 데이터 로딩은 HTML이 처음 로드될 때 데이터를 포함하지 않음
- 대신, JavaScript가 서버에 요청을 보내고 응답을 받은 후 화면에 동적으로 추가

## Selenium에서 AJAX 데이터를 처리하는 방법

- WebDriverWait을 사용하여 AJAX가 데이터를 완전히 로드할 때까지 기다림
- presence\_of\_element\_located()를 사용하여 요소가 나타날 때까지 대기

```
# AJAX 데이터가 로드될 때까지 대기
wait = WebDriverWait(driver, 10)
study_list = wait.until(EC.presence_of_element_located((By.CLASS_NAME, "study-item")))

# 모임 목록 가져오기
studies = driver.find_elements(By.CLASS_NAME, "study-item")
for study in studies:
    print(study.text)
```

# 버튼 클릭까지 자동으로! 참가 신청 완료

```
# 참가 신청 버튼 찾기
apply_button = driver.find_element(By.CLASS_NAME, "apply-btn")

# 클릭 시도
apply_button.click()

# 신청 완료 상태 확인
wait = WebDriverWait(driver, 10)
confirmation = wait.until(EC.presence_of_element_located((By.CLASS_NAME, "confirmation-msg")))

assert "신청 완료" in confirmation.text
print("스터디 참가 신청이 정상적으로 완료되었습니다.")
```

## 참가 신청 버튼 자동 클릭 과정

- `find_element()`를 사용하여 참가 신청 버튼 찾기
- `click()` 메서드를 활용하여 버튼 클릭

## 신청 완료 상태 확인

- `WebDriverWait`을 사용하여 신청 상태 변경 여부 확인
- `assert`를 활용하여 신청 완료 메시지가 있는지 검증



## 일반적인 무한 스크롤 방식의 한계

- 기본 무한 스크롤 방식(execute\_script())은 단순히 스크롤을 아래로 내리면서 데이터를 로드하는 방식
- 하지만, 스터디 모임 목록과 같은 비동기 데이터는 일정 개수 이상 로드될 때까지만 스크롤하는 방식이 더 효율적
- 무작정 스크롤을 내리는 것이 아니라, 필요한 데이터가 충분히 로드되었는지 확인하면서 스크롤을 멈추는 방식이 필요

```
# 새로운 스터디 일정이 로드되었는지 확인
new_items = WebDriverWait(driver, 5).until(
    EC.presence_of_all_elements_located((By.CLASS_NAME, "study-item")))
)

if len(new_items) > 20: # 예: 일정이 20개 이상이면 중단
    break
```

## 개선된 무한 스크롤 방식 (스터디 모임 자동 크롤링)

- execute\_script()로 스크롤을 내리는 것은 동일하지만, WebDriverWait을 활용하여 새로운 일정이 로드되었는지 확인
- 특정 개수 이상의 일정이 로드되면 자동으로 크롤링 중단



# 실습4 : 스크롤을 내리면 스터디가 보일까?

## 실습 목표

- 무한 스크롤을 구현하여 새로운 스터디 목록을 자동으로 가져오기
- 스크롤을 일정 횟수(3회) 내린 후 최종 로딩된 스터디 개수를 출력
- 스크롤을 내릴 때마다 새로운 스터디가 추가되는지 검증

## 실습 과정

- 초기 로딩된 스터디 개수를 출력
- 페이지를 아래로 스크롤
- 새로운 데이터가 로드될 때까지 대기
- 스크롤을 3회 반복하며 스터디 개수 확인
- 최종 로드된 스터디 개수를 출력

DevTools listening on ws://127.0.0.1:9229

✓	초기 로딩된 스터디 개수: 6개
▼	스크롤 다운 (1회)...
✓	현재 로딩된 스터디 개수: 12개
▼	스크롤 다운 (2회)...
✓	현재 로딩된 스터디 개수: 18개
▼	스크롤 다운 (3회)...
✓	현재 로딩된 스터디 개수: 24개
✓	최종 로딩된 스터디 개수: 24개



# UI 테스트 자동화란?

## UI 테스트 자동화

- 사람이 직접 UI를 확인하지 않고, Selenium 등의 자동화 도구를 사용하여 UI를 검증하는 과정
- 버튼 클릭, 페이지 이동, 입력 필드 검사, UI 요소의 상태(보임/숨김, 활성화 여부 등)를 자동화 가능

## UI 테스트 자동화의 필요성



시간 절약



일관성 유지



회귀 테스트

## UI 테스트 자동화 주요 개념

- `is_displayed()` → 요소가 화면에 표시되는지 확인
- `is_enabled()` → 버튼 또는 입력 필드가 활성화되어 있는지 확인
- `get_attribute()` → 특정 속성 값 확인 (예: `value`, `class`, `href`)



# 이 버튼, 정말 눌릴 수 있을까?

## Selenium을 사용하여 UI 요소 검증하기

- `is_displayed()` → 요소가 화면에 보이는지 확인
- `is_enabled()` → 요소가 활성화(클릭 가능) 상태인지 확인
- 두 함수 모두 True/False 반환



```
# 버튼 찾기
button = driver.find_element(By.ID, "submit_button")

# 버튼이 화면에 표시되는지 확인
if button.is_displayed():
    print("✓ 버튼이 화면에 표시됩니다!")

# 버튼이 활성화 상태인지 확인
if button.is_enabled():
    print("✓ 버튼이 활성화되어 있습니다!")
```



# 스크린샷 찍고, UI 비교 자동화!

## UI 비교 테스트

- 페이지 업데이트 후 기존 UI와 변경된 UI를 비교하여 자동으로 차이를 감지
- 예를 들어, 로그인 버튼의 위치가 변경되었는지, 특정 요소가 사라졌는지를 체크 가능

```
# 스크린샷 저장  
driver.save_screenshot("before.png")  
print("✓ 스크린샷이 저장되었습니다!")
```

```
# 두 개의 이미지 비교  
image1 = Image.open("before.png")  
image2 = Image.open("after.png")  
  
diff = ImageChops.difference(image1, image2)  
  
if diff.getbbox():  
    print("✗ UI 변경이 감지되었습니다!")  
else:  
    print("✓ UI 변경 없음!")
```

# ☰ 테스트 결과, 눈으로만 확인할 건가요?

## 왜 로그가 필요할까?

- 테스트 자동화는 수백 개의 UI 요소를 검증하는 과정
- 매번 콘솔에서 하나씩 체크하는 것은 비효율적
- 테스트 결과를 로그로 기록하면 나중에 변경 사항을 추적할 수 있음

```
import logging

# 로그 설정
logging.basicConfig(filename="test_results.log", level=logging.INFO, format="%(asctime)s - %(message)s")

# 로그 작성 예제
logging.info("✓ UI 테스트 시작")
logging.warning("⚠ 예상과 다른 UI 요소 발견")
logging.error("✗ 버튼이 클릭되지 않음")
logging.info("✓ UI 테스트 종료")
```

**로그 파일**

2025-02-20 10:05:31 - ✓ 버튼이 정상적으로 표시됨
2025-02-20 10:06:12 - ⚠ 버튼이 비활성화되었거나 보이지 않음
2025-02-20 10:07:45 - ✗ 버튼을 찾을 수 없음: NoSuchElementException



# UI가 변했다? 자동으로 감지해보자

## UI 변경 감지

- 웹사이트 업데이트 후 UI가 변경되었는지 자동으로 확인
- 특정 UI 요소의 텍스트, 색상, 위치, 속성 값 등이 바뀌었는지 assert를 활용하여 감지

```
# 예상되는 버튼 텍스트
expected_text = "로그인"

# 실제 버튼 텍스트 가져오기
actual_text = driver.find_element("id", "login_button").text

# UI 변경 감지
assert actual_text == expected_text, f"✖ 예상: '{expected_text}', 실제: '{actual_text}'"

print("✓ UI가 변경되지 않았습니다!")
driver.quit()
```



# 일정 관리 - 일정 추가도 자동으로!

## 프로젝트4 소개 – index.html

- Selenium을 활용하여 웹사이트에서 일정 추가 기능을 자동화
- 입력 필드에 날짜와 내용을 입력한 후 추가 버튼을 클릭하여 새로운 일정이 정상적으로 등록되는지 검증
- find\_elements()를 사용하여 추가된 일정이 목록에 표시되는지 확인

The screenshot shows a user interface for managing events. At the top, there is a header with the title '일정 관리 시스템'. Below the header, there is a date input field labeled '연도 - 월 - 일' and a button with a delete icon. Underneath is a text input field labeled '일정 제목 입력'. A large, light-gray button below these fields is labeled '일정 추가'. At the bottom of the interface, there is a sidebar titled '일정 목록' (Event List) featuring a calendar icon. It displays two event entries: '2025-02-25 - 팀 회의' and '2025-03-01 - 프로젝트 리뷰'.

# 자동으로 날짜 입력하고 일정 추가해보자

## Selenium을 사용하여 날짜 입력하고 버튼 클릭하기

- `send_keys()` → 날짜 입력 필드에 날짜를 입력
- `click()` → 추가 버튼을 클릭하여 일정 추가

## 일정 추가 과정 자동화 (실제 브라우저 동작 순서)

1. 사용자가 날짜 입력 필드에 원하는 날짜를 입력
2. 일정 제목 입력 필드에 일정 내용을 입력
3. "일정 추가" 버튼을 클릭
4. 추가된 일정이 일정 목록에 정상적으로 등록되는지 확인

일정 관리 시스템

1 연도 - 월 - 일

2 일정 제목 입력

3 일정 추가

4 일정 목록

2025-02-25 - 팀 회의
2025-03-01 - 프로젝트 리뷰
2025-03-10 - 팀 미팅



# 일정이 추가되었는지 자동으로 확인해보자

## 일정이 목록에 정상적으로 추가되었는지 검증하기

- `find_elements()` → 일정 목록을 가져와 개수 확인
- 일정 추가 전과 후의 개수를 비교하여 새로운 일정이 등록되었는지 확인

## 일정 추가 확인 과정 (자동화 테스트 순서)

1. 일정 추가 전 `find_elements()`를 사용하여 일정 개수 확인
2. Selenium을 사용하여 날짜와 제목 입력 후 "일정 추가" 버튼 클릭
3. 일정 추가 후 다시 `find_elements()`를 사용하여 개수 비교
4. `assert`를 사용하여 일정이 정상적으로 추가되었는지 검증

```
# ✅ 일정 목록 가져오기 (추가 후)
schedules_after = driver.find_elements(By.CLASS_NAME, "schedule-item")
print(f"✅ 추가 후 일정 개수: {len(schedules_after)}개")

# ✅ 일정이 정상적으로 추가되었는지 검증
assert len(schedules_after) > len(schedules_before), "✖ 일정이 추가되지 않았습니다!"

print("✅ 일정이 정상적으로 추가되었습니다!")
```



## 추가된 일정이 화면에 잘 보이는지 확인해보자

### 일정 추가 후 UI 요소가 정상적으로 표시되는지 검증

- `is_displayed()` → 요소가 화면에 표시되는지 확인
- 일정이 추가된 후 `assert`를 사용하여 요소가 화면에 나타났는지 검증

### UI 검증 과정 (자동화 테스트 순서)

1. Selenium을 사용하여 날짜와 제목 입력 후 "일정 추가" 버튼 클릭
2. `find_elements()`를 사용하여 일정 목록 가져오기
3. 추가된 일정이 `is_displayed()`를 통해 정상적으로 보이는지 확인
4. `assert`를 활용하여 추가된 일정이 표시되지 않으면 테스트 실패 처리

```
# ✅ 가장 마지막에 추가된 일정 찾기  
last_schedule = schedules[-1]
```

```
# ✅ 추가된 일정이 화면에 표시되는지 검증  
assert last_schedule.is_displayed(), "✗ 일정이 화면에 표시되지 않습니다!"  
  
print("✅ 추가된 일정이 정상적으로 표시됩니다!")
```

```
DevTools listening on ws://127.0.0.1:1963/d  
✅ 추가된 일정이 정상적으로 표시됩니다!
```

# 실습5: 일정 추가 후 UI가 표시되는지 검증

## 실습 목표

- Selenium을 사용하여 일정 추가 후 UI가 정상적으로 반영되는지 검증
- find\_elements()를 활용해 추가 전/후 일정 개수를 비교
- is\_displayed()를 사용하여 추가된 일정이 정상적으로 화면에 표시되는지 확인

## 실습 과정

- 일정 추가 전 현재 일정 개수 확인
- 날짜 입력 필드에 "2025-03-25" 설정
- 일정 제목 입력 ("월간 회의")
- "일정 추가" 버튼 클릭
- 일정 추가 후 새롭게 추가된 일정 개수 확인
- is\_displayed()를 활용하여 추가된 일정이 정상적으로 표시되는지 검증

```
DevTools listening on ws://127.0.0.1:9229
✓ 추가 전 일정 개수: 2개
✓ 추가 후 일정 개수: 3개
✓ 일정이 정상적으로 화면에 표시됨!
```



## 퀴즈 타임



QUIZ 9번부터 16번을 풀어봐요!