

# Selenium WebDriver 기본 개념 및 환경 설정

# Selenium이란? 웹을 조작하는 마법



Selenium

사람이 직접 조작하지 않고도 웹사이트를 자동으로 움직이게 하는 도구

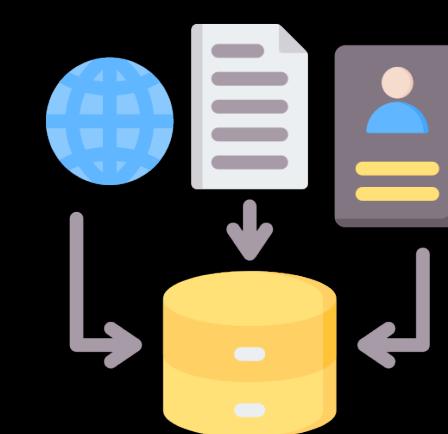
Selenium을 사용하면 할 수 있는 일



자동 로그인



자동 검색



데이터 크롤링



테스트 자동화

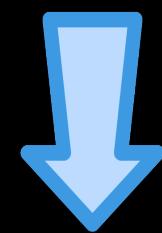


# 왜 웹 자동화가 필요할까?



## 웹사이트를 사람이 직접 조작하는 경우의 문제점

- 반복 작업이 많음 (매번 수동으로 로그인, 검색, 클릭해야 함)
- 사람이 직접 하면 실수가 발생할 가능성이 높음
- 많은 데이터를 처리하려면 시간이 오래 걸림



## Selenium을 활용한 자동화 테스트의 장점

- 같은 작업을 여러 번 반복해도 실수가 없음
- 사람이 직접 할 때보다 훨씬 빠르게 작업 가능
- 테스트 결과를 자동으로 기록하고 비교할 수 있음



# 웹사이트는 어떻게 동작할까?



## HTML의 기본 개념

- 웹페이지는 HTML 코드로 만들어짐
- HTML은 텍스트, 이미지, 버튼, 입력 창 등 웹사이트의 모든 요소를 포함
- 웹 요소를 찾아 조작하려면 HTML 구조를 이해해야 함

## 웹페이지에서 요소를 찾는 방법

웹사이트에서 버튼을 클릭하거나, 텍스트를 입력하거나, 특정 정보를 가져오려면 해당 요소를 찾아야 함  
Selenium은 ID, Class, Name 등 다양한 방법으로 요소를 찾을 수 있음



# HTML이란? 웹페이지를 만드는 기본 언어

## HTML

- **HTML**(HyperText Markup Language)은 웹페이지를 만드는 기본적인 언어
- 웹사이트의 모든 텍스트, 이미지, 버튼, 입력창 등의 요소를 HTML로 작성

## HTML 기본 구조

```
<!DOCTYPE html>
<html>
<head>
    <title>내 첫 번째 웹페이지</title>
</head>
<body>
    <h1>안녕하세요!</h1>
    <p>이것은 웹페이지의 기본 구조입니다.</p>
</body>
</html>
```

### **<html>** 태그

: 웹페이지의 시작과 끝을 나타냄

### **<head>** 태그

: 페이지 제목, 스타일 등의 설정을 포함

### **<body>** 태그

: 화면에 보이는 내용이 들어감



# HTML의 핵심 개념 - 태그, 속성, 계층구조

**HTML 요소(Tag)** : 웹페이지의 각 부분을 나타내는 코드

- <h1>: 제목
- <p>: 문단(텍스트)
- <button>: 버튼
- <input>: 입력 필드



# HTML의 핵심 개념 - 태그, 속성, 계층구조

**HTML 요소(Tag)**: 웹페이지의 각 부분을 나타내는 코드

- <h1>: 제목
- <p>: 문단(텍스트)
- <button>: 버튼
- <input>: 입력 필드
- :



# HTML의 핵심 개념 - 태그, 속성, 계층구조

**HTML 속성(Attribute)**: 태그에 추가적인 정보를 제공하는 역할

```
<input type="text" placeholder="이름을 입력하세요">
```



# HTML의 핵심 개념 - 태그, 속성, 계층구조

**HTML 계층 구조**: HTML은 부모 요소와 자식 요소로 이루어진 트리 구조

```
<div>
  <h1>제목입니다</h1>
  <p>본문 내용입니다.</p>
</div>
```

→ <div>는 **부모 요소**, <h1>과 <p>는 **자식 요소**



## CSS(Cascading Style Sheets)

- HTML로 만든 웹페이지를 더 보기 좋게 디자인하는 언어
- HTML은 내용(content)을 담당하고, CSS는 스타일(style)을 담당
- 웹사이트의 색상, 글꼴, 크기, 정렬 등을 조정 가능

### CSS의 기본 문법

```
h1 {  
    color: blue; /* 글자 색상을 파란색으로 변경 */  
    font-size: 24px; /* 글자 크기를 24px로 설정 */  
    text-align: center; /* 가운데 정렬 */  
}
```



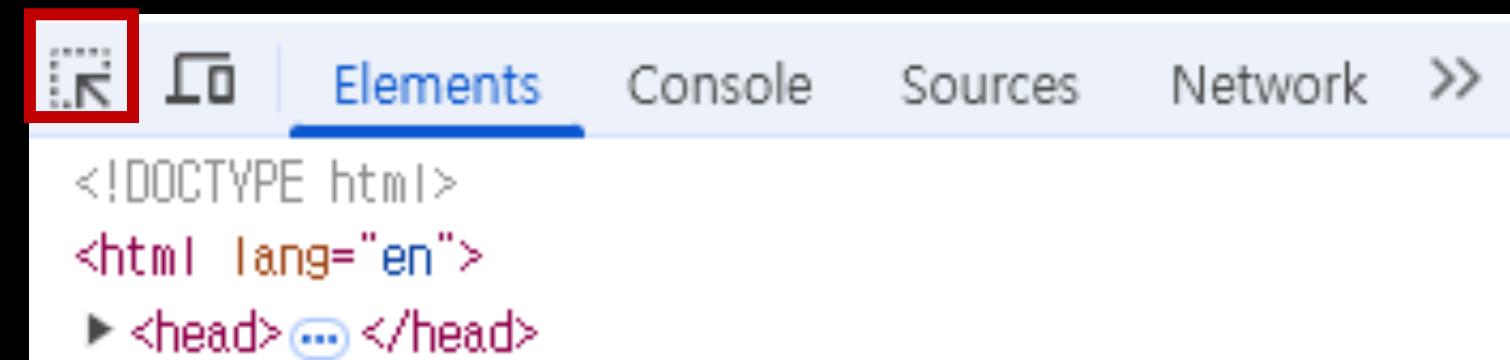
# 실제 웹사이트에서 HTML 요소를 찾는 방법

## 개발자 도구

- 브라우저에서 HTML 코드와 요소의 속성을 확인할 수 있는 기능
- 크롬 브라우저에서 F12 또는 마우스 우클릭 → '검사' 클릭

## 개발자 도구에서 웹 요소 찾기

1. '요소 선택' 버튼(WebElement icon)을 클릭하여 웹페이지의 특정 요소를 선택
2. 선택한 요소의 ID, Class, Name을 확인





# Selenium0 웹을 이해하는 방법

## 웹 요소

- 버튼, 입력창, 링크, 이미지 등 웹페이지의 개별적인 부분
- Selenium이 웹을 자동으로 조작하려면, 먼저 이 요소를 찾아야 함

## 요소를 찾는 3가지 주요 방법

### 1. ID

- HTML 요소에 부여된 고유한 값
- 한 페이지 내에서 유일하기 때문에 찾기 가장 쉬움

```
<input id="username" type="text" placeholder="아이디 입력">
```

### 2. Class

- 여러 요소가 같은 Class를 가질 수 있음
- 스타일 적용을 위해 주로 사용

```
<button class="login-btn">로그인</button>
```

### 3. Name

- 폼 요소(입력 필드, 체크박스, 버튼 등)에서 많이 사용

```
<input name="email" type="text">
```



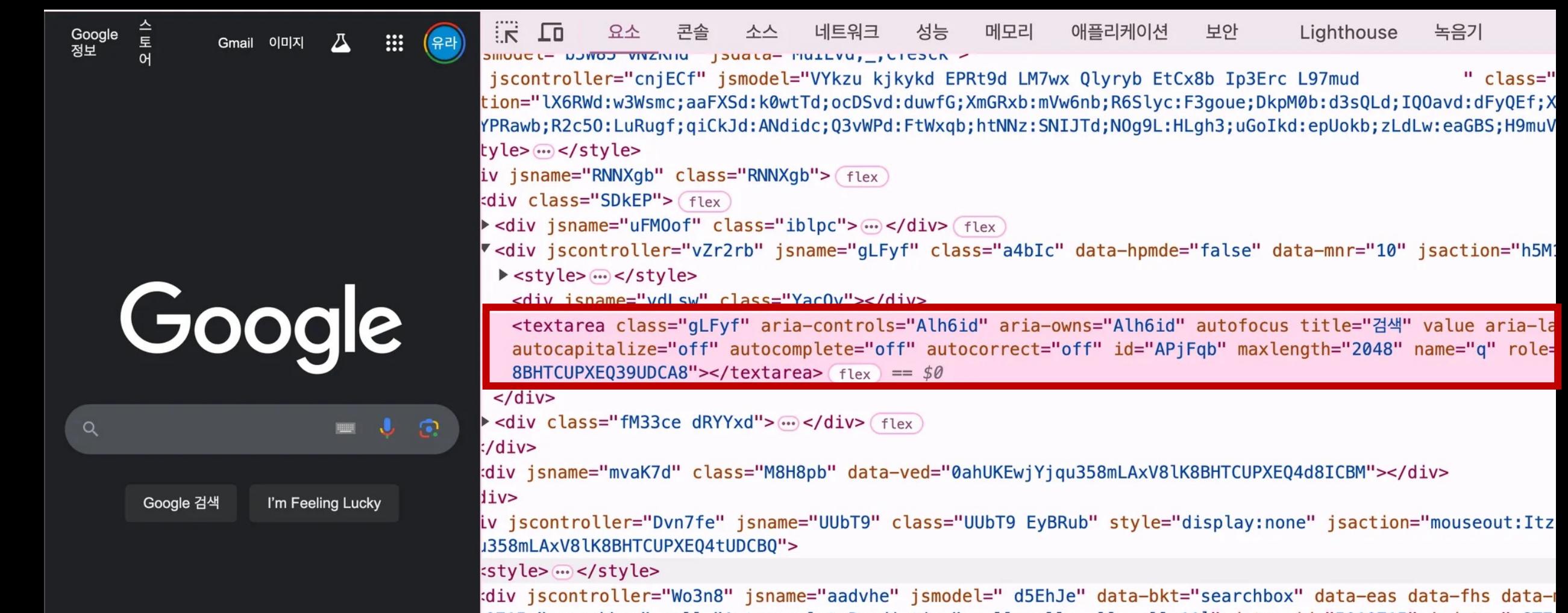
# 실습 1: 웹사이트에서 HTML 요소 찾아보기

## 실습 목표

- Chrome 개발자 도구를 사용하여 웹페이지에서 요소를 직접 찾아본다.
- ID, Class, Name을 이용해 Selenium 코드에서 어떻게 활용할 수 있는지 이해한다.

## 실습 진행 방법

- 구글에 접속 (<https://www.google.com>)
- 검색창을 클릭한 뒤 개발자 도구(F12) 실행
- 검색창의 name 값을 확인 (예: name="q")





# Selenium으로 할 수 있는 일

## Selenium이 할 수 있는 작업 예시

### 1. 브라우저 열기 & 웹페이지 이동

- 특정 웹사이트를 자동으로 실행 (예: 'google.com', 'naver.com')

### 2. 버튼 클릭

- 로그인 버튼, 검색 버튼 등 자동 클릭 가능

### 3. 텍스트 입력

- 검색창, 로그인 창 등 특정 입력 필드에 자동 입력

### 4. 데이터 가져오기 (크롤링)

- 웹페이지에서 텍스트, 이미지, 링크 등의 정보 추출

### 5. 폼 자동 입력

- 아이디/비밀번호 입력 후 로그인 자동화

### 6. 스크롤 & 팝업 조작

- 자동으로 페이지를 내리거나 팝업 창 닫기



## ☰ 실습 2: 내 첫 번째 웹페이지 만들기

### 실습 목표

- 기본적인 HTML 페이지를 만들어보고 직접 브라우저에서 실행해본다.
- Selenium이 웹페이지를 조작하는 원리를 이해하기 위해 HTML 구조를 직접 경험해본다.

### 실습 진행 방법

- HTML 파일 만들기
- index.html 파일을 생성하여 아래 코드 입력

```
<!DOCTYPE html>
<html>
<head>
    <title>테스트 페이지</title>
</head>
<body>
    <h1>안녕하세요, Selenium 자동화 실습입니다!</h1>
    <input type="text" id="search-box" placeholder="검색어 입력">
    <button id="search-btn">검색</button>
</body>
</html>
```

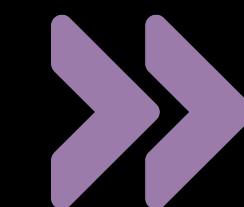


# HTML에서 입력 & 버튼 클릭 원리 이해하기

## 입력 필드(Input)

사용자가 텍스트를 입력할 수 있는 HTML 요소

```
<input type="text" id="username" placeholder="아이디 입력">
```

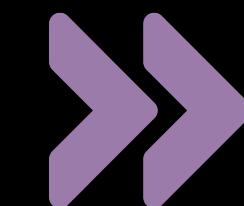


type="text" → 텍스트 입력 필드  
id="username" → Selenium이 찾을 수 있도록 지정한 ID

## 버튼(Button) 클릭

사용자가 버튼을 클릭하면 특정 동작 수행

```
<button id="login-btn">로그인</button>
```



id="login-btn" → Selenium이 버튼을 찾는 데 사용



## 퀴즈 타임



QUIZ 1번부터 5번을 풀어봐요!



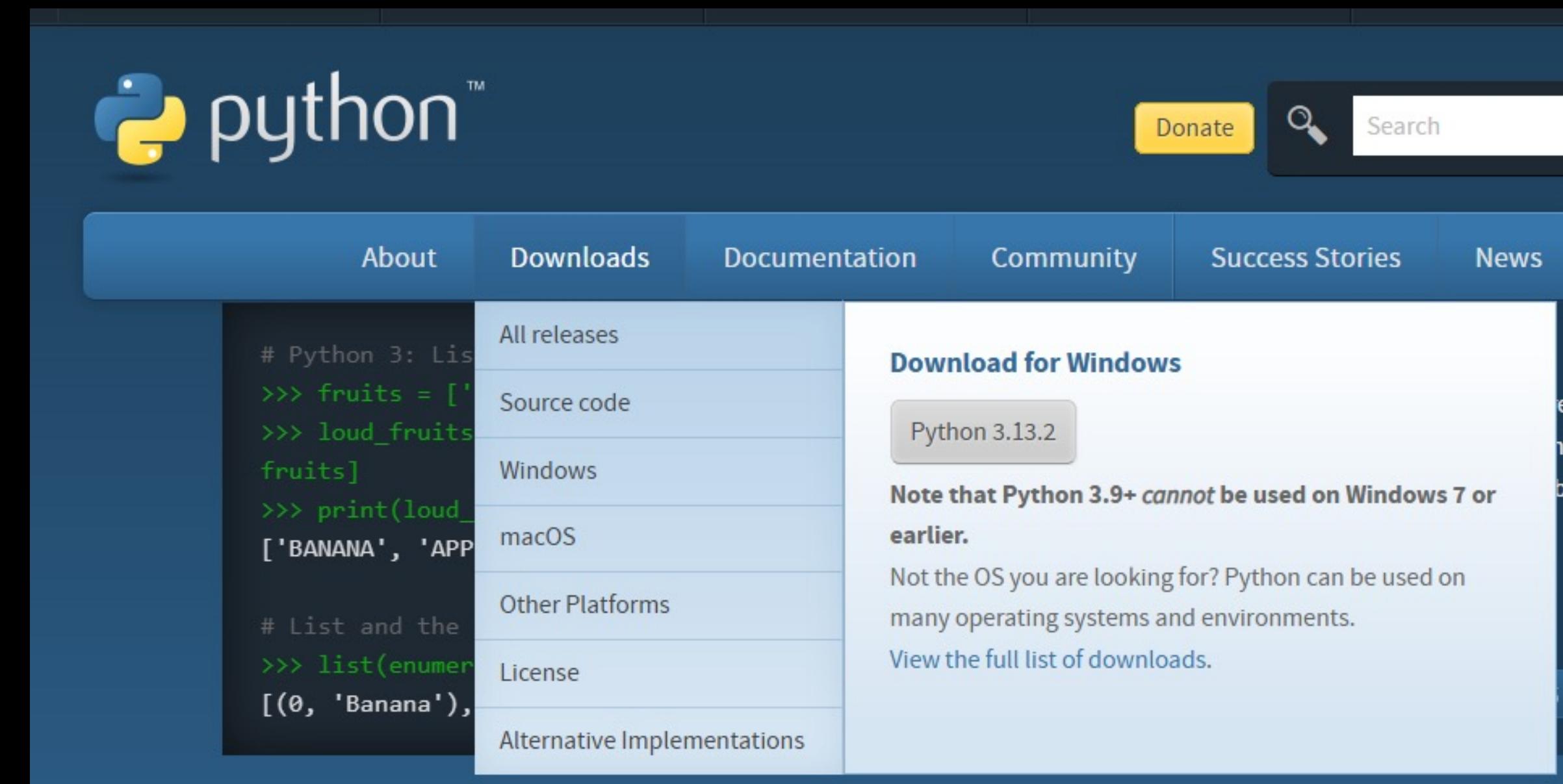
# Python 설치하기

Selenium을 사용하려면?

*Selenium을 사용하여 웹 자동화를 하기 위해 필요한 준비 사항:*

1. Python 설치

<https://www.python.org/>





# VS Code 설치하기

## 2. VS Code (코드 편집기) 설치

<https://code.visualstudio.com/>

The screenshot shows the official Visual Studio Code website. At the top, there is a navigation bar with links to 'Visual Studio Code', 'Docs', 'Updates', 'Blog', 'API', 'Extensions', 'FAQ', and 'GitHub Copilot'. To the right of the navigation bar is a search bar. Below the navigation bar, there is a promotional message: 'Get GitHub Copilot Free in VS Code!' with a small rocket icon. The main headline on the page reads 'Your code editor. Redefined with AI.' in large, bold, dark text. At the bottom, there are two prominent buttons: a blue button labeled 'Download for Windows' and a white button labeled 'Get Copilot Free'. Below these buttons, there is a link to 'Web, Insiders edition, or other platforms'.

Visual Studio Code Docs Updates Blog API Extensions FAQ GitHub Copilot

Get GitHub Copilot Free in VS Code!

# Your code editor. Redefined with AI.

[Download for Windows](#) [Get Copilot Free](#)

[Web, Insiders edition, or other platforms](#)



# Selenium 설치 및 ChromeDriver 실행

## 3. Selenium 라이브러리 설치

```
pip install selenium
```

## 4. ChromeDriver 실행

```
from selenium import webdriver  
  
driver = webdriver.Chrome() # 웹드라이버 실행
```



# 브라우저를 연결해주는 ChromeDriver

## ChromeDriver

- Selenium이 Chrome 브라우저를 조작할 수 있도록 해주는 드라이버
- 버전이 맞지 않으면 동작하지 않으므로 주의해야 함!



## ChromeDriver 자동 설치 방법

```
import chromedriver_autoinstaller  
  
# 최신 버전 자동 설치  
chromedriver_autoinstaller.install()
```



# 브라우저 자동으로 열어보기

## Selenium이 브라우저를 실행하는 원리

- Selenium은 웹 브라우저 자동화 도구로, Python을 통해 직접 브라우저를 열고 조작할 수 있음
- 브라우저를 자동 실행하기 위해서는 WebDriver가 필요함
- WebDriver는 Selenium이 브라우저와 상호작용할 수 있도록 도와주는 중간 매개체 역할을 수행

```
import time
from selenium import webdriver
import chromedriver_autoinstaller

# 최신 ChromeDriver 자동 설치
chromedriver_autoinstaller.install()

# WebDriver를 통해 Chrome 실행
driver = webdriver.Chrome()

# 특정 웹사이트 접속 (Google)
driver.get("https://www.google.com")

# 3초 동안 유지 후 종료
time.sleep(3)
driver.quit()
```

webdriver.Chrome() → Chrome 브라우저 실행  
driver.get("URL") → 지정한 웹사이트 열기  
time.sleep(3) → 3초간 대기  
driver.quit() → 브라우저 종료



# ID, Name, Class의 차이점

## [ID, Class, Name의 차이점]

속성	특징	사용 예제
ID	유일한 값 (한 페이지에 하나)	<input id="username">
Class	여러 요소가 같은 Class를 가질 수 있음	<button class="btn">
Name	입력 필드에서 많이 사용됨	<input name="email">

### 어떤 경우에 어떤 속성을 사용할까?

- ID가 있으면 ID 사용 (가장 빠름)
- Class는 동일한 요소가 여러 개 있을 때 유용
- Name은 폼 입력 요소에서 자주 사용됨
- ID, Class, Name이 없으면 CSS Selector나 XPath 사용

# find\_element() - 웹 요소 찾기

## find\_element()

- Selenium이 특정 HTML 요소를 찾을 때 사용하는 메서드
- 'find\_element("속성", "값")' 형식으로 사용

```
search_box = driver.find_element("name", "q") # 검색창 찾기  
search_box.send_keys("Selenium 자동화") # 검색어 입력
```

## find\_element() vs find\_elements()

함수	역할	예제
<b>find_element()</b>	하나의 요소만 찾음 (첫 번째 요소)	driver.find_element("name", "q")
<b>find_elements()</b>	여러 개의 요소를 리스트로 반환	driver.find_elements("class name", "button")

## Selenium에서 요소를 찾을 때 발생할 수 있는 문제

- 웹페이지가 완전히 로드되기 전에 요소를 찾으려고 하면 'NoSuchElementException' 오류 발생
- 특히, JavaScript로 동적으로 로딩되는 요소는 일정 시간이 지난 후에만 나타남
- 따라서, 요소가 나타날 때까지 기다려야 함

## 해결 방법: WebDriverWait 사용

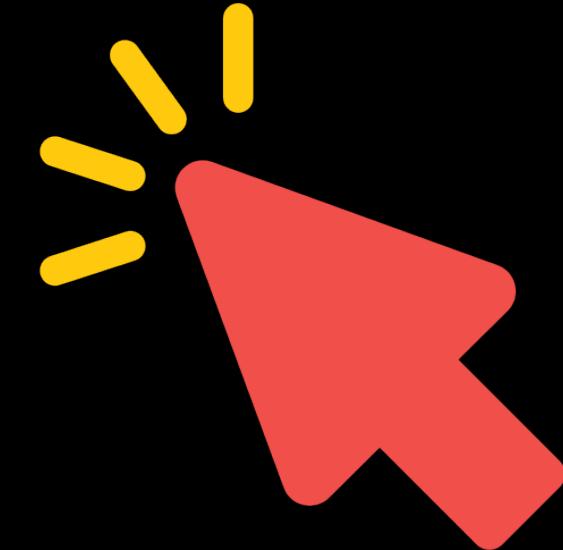
```
from selenium.webdriver.support.ui import WebDriverWait
from selenium.webdriver.support import expected_conditions as EC

# 요소가 나타날 때까지 최대 10초 동안 대기
search_box = WebDriverWait(driver, 10).until(
    EC.presence_of_element_located((By.ID, "searchBox"))
)
```

# Selenium으로 버튼 클릭 자동화하기

## Selenium에서 버튼 클릭하는 방법

Selenium은 `find_element()`를 사용해 특정 버튼을 찾고, `.click()`메서드를 실행하여 클릭할 수 있다.



```
element = driver.find_element(By.ID, "button_id") # 요소 찾기  
element.click() # 버튼 클릭
```

1. `find_element(By.ID, "button_id")` → 검색 버튼 찾기
2. `.click()` → 버튼 클릭 실행
3. 버튼을 찾을 수 없는 경우 'NoSuchElementException' 오류 발생 가능



# send\_keys() 사용법과 동작 원리

## send\_keys()

- 키보드 입력을 대신하여 텍스트 필드에 값을 입력하는 Selenium 메서드
- 사용자가 직접 타이핑하는 것과 같은 효과를 가짐
- 검색창, 로그인 폼, 입력 필드 자동화에 활용

### 자동 입력 후 검색 실행하는 방법

```
from selenium.webdriver.common.keys import Keys

search_box.send_keys("Selenium 자동화")
search_box.send_keys(Keys.RETURN) # 엔터 키 입력
```

일부 웹사이트에서는 'send\_keys()'로 입력한 후 엔터 키('Keys.RETURN')을 눌러야 검색이 실행될 수도 있음



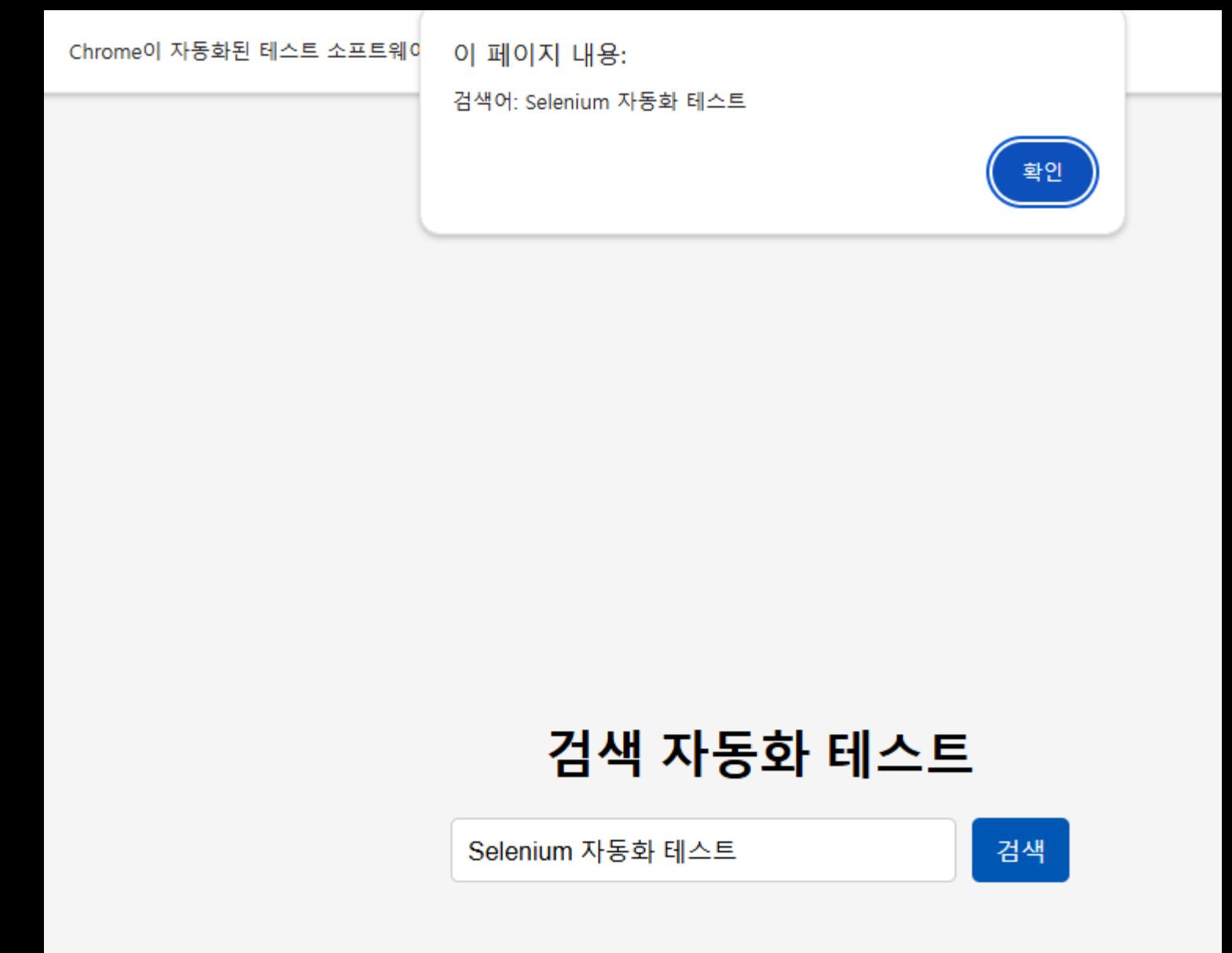
## 실습 3: 검색창 자동 입력 & 버튼 클릭하기

### 실습 목표

- Selenium을 사용하여 웹사이트의 검색창을 자동으로 조작하는 방법을 배운다.
- index.html을 직접 실행하고, 검색어 입력 & 검색 버튼 클릭을 자동화한다.

### 실습 개요

1. 로컬 HTML 파일(index.html)을 생성하여 실행
2. 검색창을 찾아 "Selenium 자동화 테스트" 입력
3. 검색 버튼을 클릭하여 검색 실행





## 퀴즈 타임



QUIZ 6번부터 10번을 풀어봐요!



# Selenium으로 입력 필드와 버튼 조작하기

## 입력 필드(Form Input)와 버튼(Button)의 역할

- 로그인, 회원가입, 검색 등 웹사이트의 핵심적인 인터랙션 요소
- Selenium을 사용하면 입력 필드에 자동으로 텍스트 입력하고 버튼을 클릭할 수 있음

### 입력 필드 조작 (`send_keys()`)

```
input_box = driver.find_element(By.ID, "username")
input_box.send_keys("test_user") # 입력창에 값 입력
```

- `.send_keys()` 를 사용하면 사람이 직접 키보드로 입력하는 것처럼 텍스트를 입력할 수 있음

### 버튼 클릭 (`click()`)

```
login_button = driver.find_element(By.ID, "loginBtn")
login_button.click() # 로그인 버튼 클릭
```



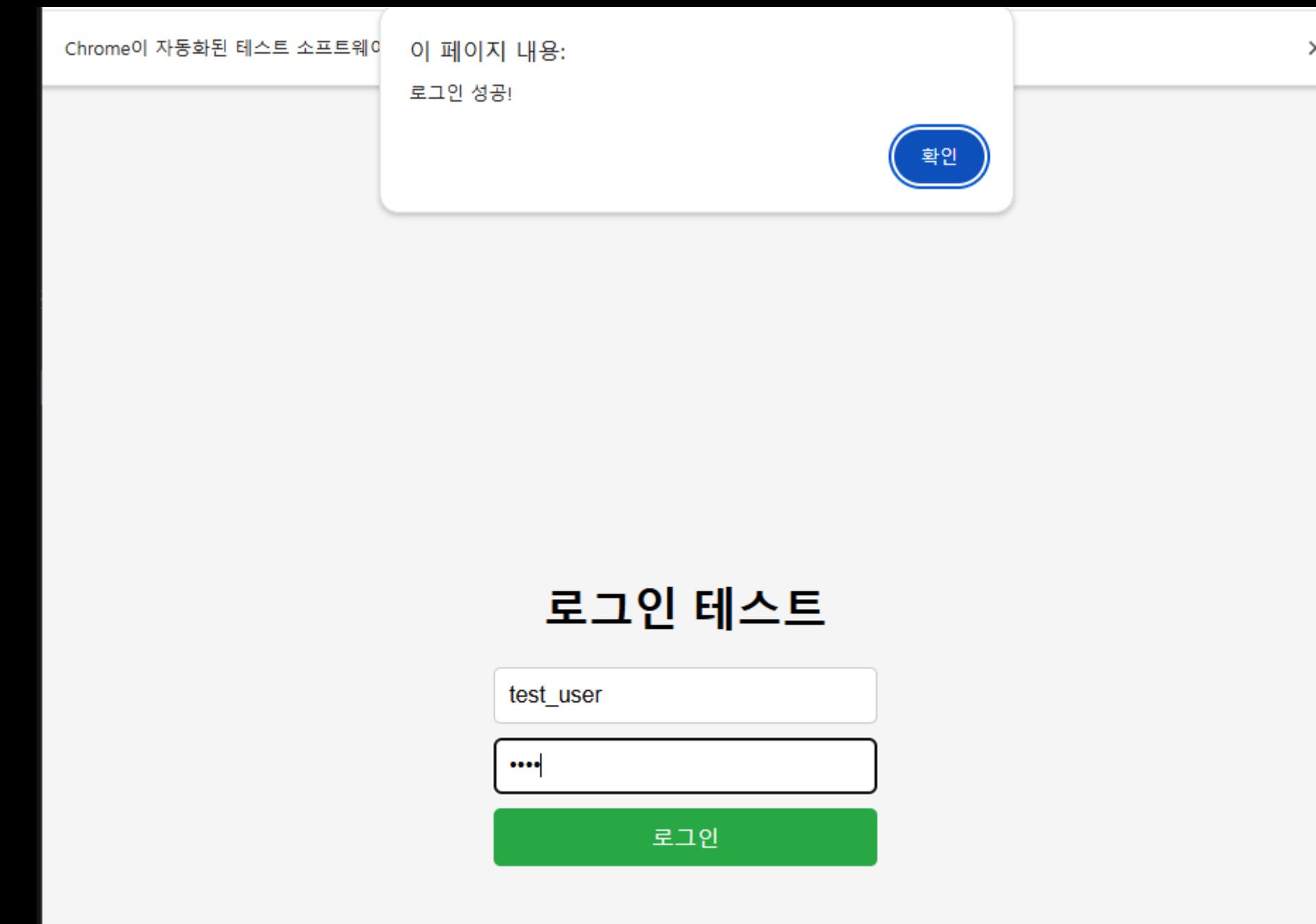
## 실습 4: 가짜 로그인 테스트

### 실습 목표

login.html을 브라우저에서 실행하여, Selenium이 자동으로 아이디와 비밀번호를 입력하고 로그인 버튼을 클릭하는 과정 실습

### 실습 개요

1. login.html을 브라우저에서 실행하여  
직접 로그인 테스트
2. Selenium 코드(main.py) 실행 후  
자동으로 입력 및 로그인 버튼 클릭 확인
3. 올바른 아이디/비밀번호 입력  
→ "로그인 성공!"
4. 틀린 아이디/비밀번호 입력  
→ "로그인 실패!"





# 로그인 성공/실패 메시지 확인하기

## 로그인 성공/실패 메시지를 자동으로 확인해야 하는 이유

- 로그인 자동화 이후 정상적으로 로그인되었는지 검증하는 과정 필요
- Selenium을 사용하면, 페이지에서 특정 메시지를 찾아 로그인 성공 여부 확인 가능

## 로그인 성공/실패 메시지를 확인하는 방법

### 1. 로그인 성공 메시지가 있는지 확인 (`text` 속성 활용)

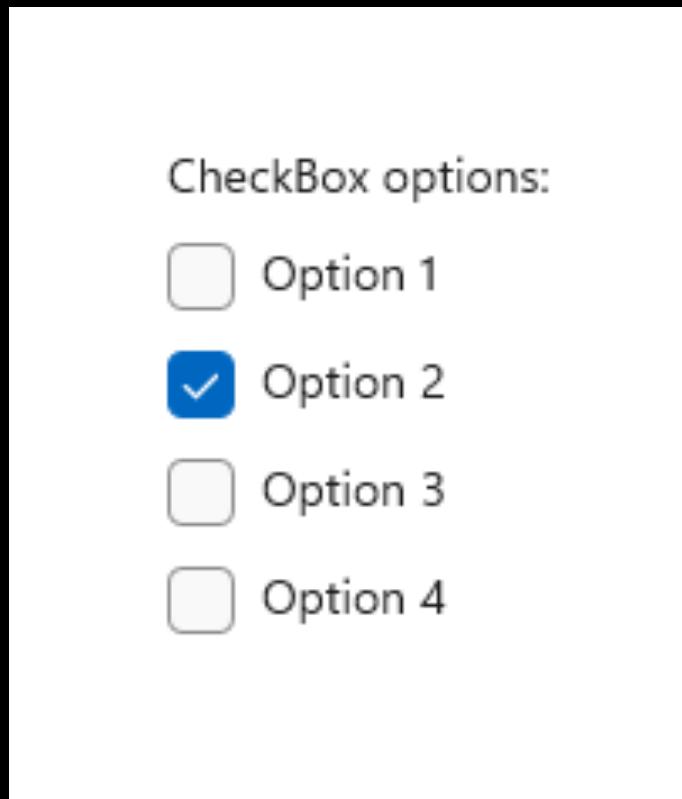
```
success_msg = driver.find_element(By.ID, "successMessage").text  
print(success_msg) # "로그인 성공!" 출력 예상
```

### 2. 로그인 실패 메시지를 찾기 (`text` 비교)

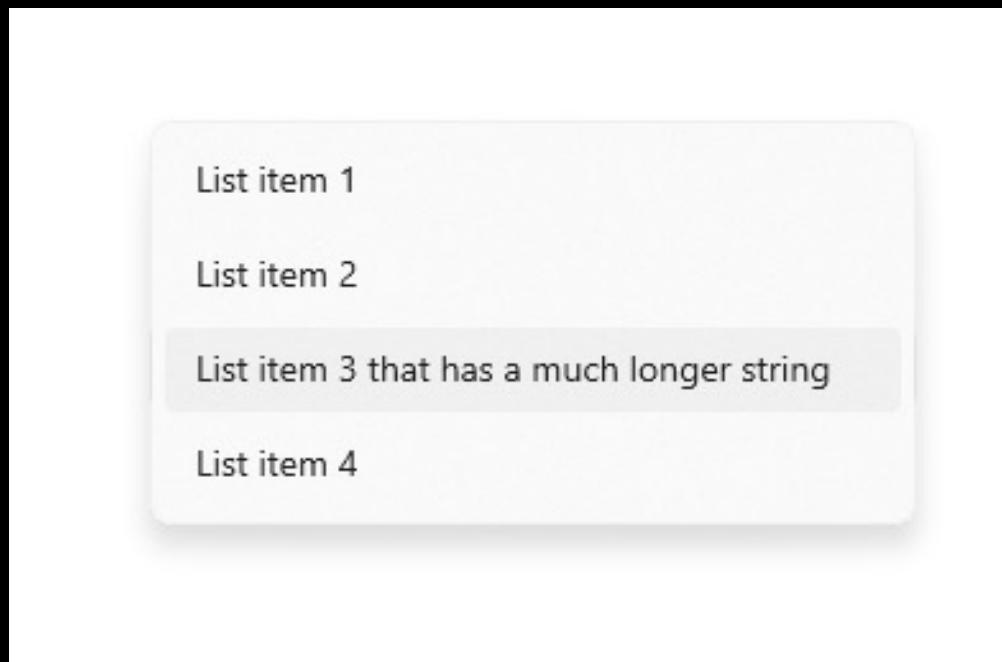
```
error_msg = driver.find_element(By.ID, "errorMessage").text  
if "로그인 실패" in error_msg:  
    print("로그인 실패 감지됨!")
```

# 체크박스와 드롭다운을 선택하는 방법

## 체크박스(Checkbox)와 드롭다운(Dropdown)의 역할



체크박스:  
다중 선택이 필요한 경우 사용



드롭다운:  
여러 개의 옵션 중 하나를  
선택하는 경우 사용

Selenium을 사용하면 체크박스를 자동으로 체크하거나  
드롭다운에서 특정 값을 선택할 수 있음

### 체크박스 자동 선택 방법

```
checkbox = driver.find_element(By.ID, "agreeCheckbox")
checkbox.click() # 체크박스 클릭 (선택/해제)
```

◆ 체크박스가 이미 선택된 상태인지 확인하려면?

```
if not checkbox.is_selected():
    checkbox.click() # 체크 안 되어 있으면 클릭
```

### 드롭다운에서 특정 옵션 선택하는 방법 (`Select` 클래스 활용)

```
from selenium.webdriver.support.ui import Select

dropdown = Select(driver.find_element(By.ID, "country"))
dropdown.select_by_visible_text("한국") # "한국" 옵션 선택
```



# 다양한 HTML 폼 요소 자동화하기

## HTML 폼 요소

- 웹사이트에서 사용자 입력을 받는 모든 요소를 의미
- input, select, textarea, radio button, checkbox, button 등이 있음

### Selenium을 활용하여 HTML 폼 요소 자동화

#### 1. 텍스트 입력 필드 (`input[type="text"]`)

```
input_field = driver.find_element(By.ID, "nameInput")
input_field.send_keys("홍길동")
```

- ◆ `send_keys()` 를 활용하여 사용자 입력을 대신 입력 가능

#### 2. 다중 입력 필드 (`textarea`)

```
textarea = driver.find_element(By.ID, "messageBox")
textarea.send_keys("이것은 자동 입력된 메시지입니다.")
```

- ◆ 다른 텍스트 입력 필드와 동일하게 `send_keys()` 사용



# 다양한 HTML 폼 요소 자동화하기

## Selenium을 활용하여 HTML 폼 요소 자동화

### 3. 라디오 버튼 ( `radio button` ) 선택

```
radio_button = driver.find_element(By.ID, "maleRadio")
radio_button.click()
```

- ◆ 라디오 버튼은 `click()` 을 이용하여 선택 가능

### 4. 드롭다운 ( `select` 태그)에서 옵션 선택

```
from selenium.webdriver.support.ui import Select

dropdown = Select(driver.find_element(By.ID, "city"))
dropdown.select_by_value("seoul") # value 기준 선택
```

- ◆ `select_by_visible_text()` 또는 `select_by_index()` 로도 선택 가능

### 5. 파일 업로드 ( `input[type="file"]` )

```
file_input = driver.find_element(By.ID, "fileUpload")
file_input.send_keys("D:/필요/python/241121/sample.txt")
```

- ◆ 파일 업로드는 `send_keys()` 를 사용하여 직접 파일 경로를 입력하면 됨



# 실습 5: 체크박스 & 드롭다운 자동 선택하기

## 실습 목표

- 체크박스 선택 여부를 확인하고 자동으로 체크하기
- 드롭다운에서 특정 옵션을 자동으로 선택하기
- Selenium을 활용하여 폼 입력 자동화 연습

## 실습 개요

- Selenium 코드를 실행하여 체크박스 선택 및 드롭다운 값 변경 자동화
- 사용자가 입력한 값이 정상적으로 선택되었는지 확인

체크박스 & 드롭다운 테스트

이용약관에 동의합니다.

국가 선택:

한국

제출



# Selenium으로 웹페이지의 텍스트 가져오기

- Selenium을 사용하면 웹페이지에서 텍스트를 자동으로 추출할 수 있음
- .text는 보이는 텍스트만 가져올 수 있음
- get\_attribute("textContent")는 숨겨진 텍스트까지 가져올 수 있음

## 'element.text' 사용 (보이는 텍스트 가져오기)

```
element = driver.find_element(By.TAG_NAME, "h1")
print("페이지 제목:", element.text)
```

## 'get\_attribute("textContent")' 사용 (숨겨진 텍스트 가져오기)

```
hidden_text = driver.find_element(By.ID, "hidden").get_attribute("textContent")
print("숨겨진 텍스트:", hidden_text)
```



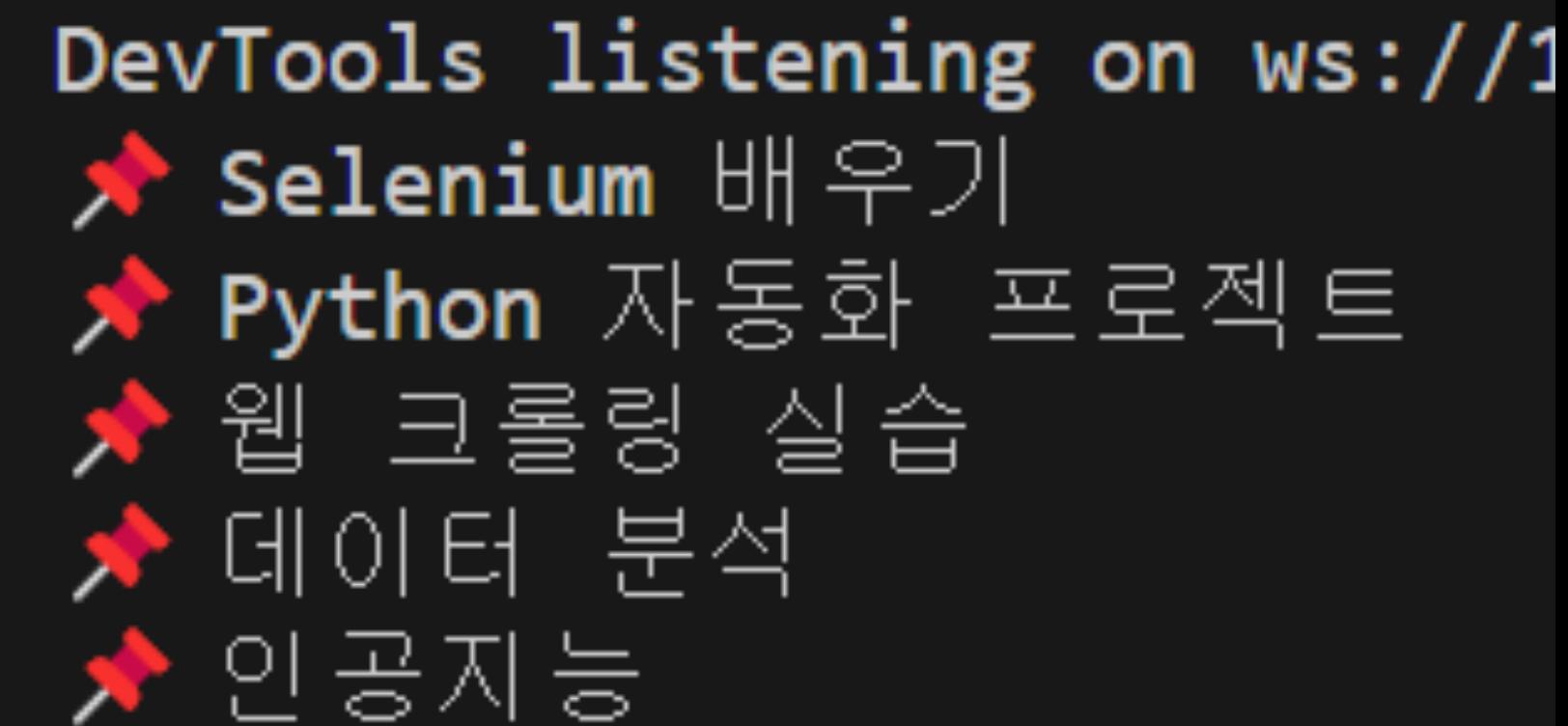
# 실습6 : 웹페이지의 텍스트 추출

## 실습 목표

- Selenium을 사용하여 웹페이지에서 여러 개의 텍스트를 가져오는 방법을 익힌다.
- find\_elements()를 사용하여 여러 개의 요소를 리스트 형태로 가져오는 원리를 학습한다.
- .text 속성을 활용하여 요소 내부의 텍스트를 추출하는 방법을 실습한다.

## 실습 진행 방법

1. index.html 파일을 열어 여러 개의 텍스트 데이터를 가져오기
2. find\_elements()를 사용하여 클래스가 "item"인 요소들을 모두 찾기
3. 찾은 요소에서 .text 속성을 사용하여 텍스트를 출력하기



# Selenium 탐정! 숨겨진 링크를 추적하라

## 웹페이지에서 링크 가져오기

- Selenium을 사용하면 웹페이지에 있는 모든 링크(URL)를 가져올 수 있음
- `get_attribute("href")`를 사용하여 `<a>` 태그의 href 속성을 추출
- `find_elements(By.TAG_NAME, "a")`를 활용하여 페이지 내 모든 링크를 한 번에 가져오기 가능

```
# 페이지 내 모든 링크 가져오기
links = driver.find_elements(By.TAG_NAME, "a")

# 각 링크의 URL 출력
for link in links:
    print(link.get_attribute("href"))
```



### **get\_attribute("href")**

- HTML에서 `<a>` 태그는 href 속성을 포함하고 있음
- `get_attribute("href")`를 사용하면 해당 링크의 URL을 추출 가능



# 데이터 수집 완료! 이제 깔끔하게 저장하자

## CSV 파일 저장의 필요성

- 크롤링한 데이터를 프로그램에서 활용 가능하도록 저장
- CSV 파일은 엑셀, 데이터 분석, 머신러닝에 활용하기 용이
- csv 라이브러리를 사용하여 Python에서 쉽게 파일 저장 가능

```
# 페이지 내 모든 'item' 클래스 요소 가져오기
items = driver.find_elements(By.CLASS_NAME, "item")

# CSV 파일 저장
with open(csv_file_path, "w", newline="", encoding="utf-8") as file:
    writer = csv.writer(file)
    writer.writerow(["텍스트"]) # CSV 헤더 작성
    for item in items:
        writer.writerow([item.text]) # 데이터 저장

driver.quit()
print(f"데이터가 {csv_file_path} 파일로 저장되었습니다.")
```

scraped_data.csv	
1	텍스트
2	→ Selenium 배우기
3	→ Python 자동화 프로젝트
4	→ 웹 크롤링 실습
5	→ 데이터 분석
6	→ 인공지능



# 자동 스크롤링으로 웹페이지 끝까지 탐색하기

## 스크롤 자동화의 필요성

- 일부 웹페이지는 스크롤을 내려야 추가 콘텐츠가 로드됨
- Selenium의 `execute_script()`를 사용하면 자동으로 스크롤을 조작할 수 있음
- 무한 스크롤 페이지(예: 인스타그램, 뉴스 사이트)에서 유용하게 활용 가능

## execute\_script()를 활용한 다양한 스크롤 방법



### 1) 페이지 맨 아래까지 스크롤 이동

```
driver.execute_script("window.scrollTo(0, document.body.scrollHeight);")
```

### 2) 특정 위치로 이동 (예: 500px 아래로 스크롤)

```
driver.execute_script("window.scrollBy(0, 500);")
```

### 3) 특정 요소까지 스크롤 이동 (`scrollIntoView`)

```
element = driver.find_element(By.ID, "targetElement")
driver.execute_script("arguments[0].scrollIntoView();", element)
```

# 웹사이트의 팝업을 자동으로 닫아라

## Selenium에서 팝업 창 처리하기

- Selenium은 자바스크립트가 생성하는 팝업(alert, confirm, prompt)도 조작 가능
- 특정 웹사이트는 팝업을 닫지 않으면 크롤링이 어려움
- switch\_to.alert을 사용하여 자동으로 팝업을 닫거나 입력 가능

```
# 팝업이 나타날 때까지 대기
try:
    WebDriverWait(driver, 5).until(EC.alert_is_present())
    alert = driver.switch_to.alert
    print("팝업 텍스트:", alert.text) # 팝업 텍스트 출력

    time.sleep(3) # 팝업이 닫히기 전에 3초 동안 유지 (사용자가 볼 수 있도록)

    alert.accept() # 팝업 닫기
    print("팝업이 정상적으로 닫혔습니다.")
except Exception as e:
    print("팝업이 나타나지 않았습니다:", e)
```

팝업 종류	설명	Selenium 처리 방법
alert	확인 버튼만 있는 경고창	alert.accept()
confirm	확인/취소 버튼 제공	alert.accept() or alert.dismiss()
prompt	사용자 입력을 요구하는 팝업	alert.send_keys("입력값") 후 accept()



## 퀴즈 타임



QUIZ 11번부터 15번을 풀어봐요!