

1. Python Selenium 설치하기

다른 라이브러리를 설치하는 것과 동일합니다.

pip 또는 conda 명령어를 사용해서 설치해 주세요.

```
pip install selenium
```

```
conda install selenium
```

파이참에서는 툴 아래쪽 'Python Packages' 탭 클릭

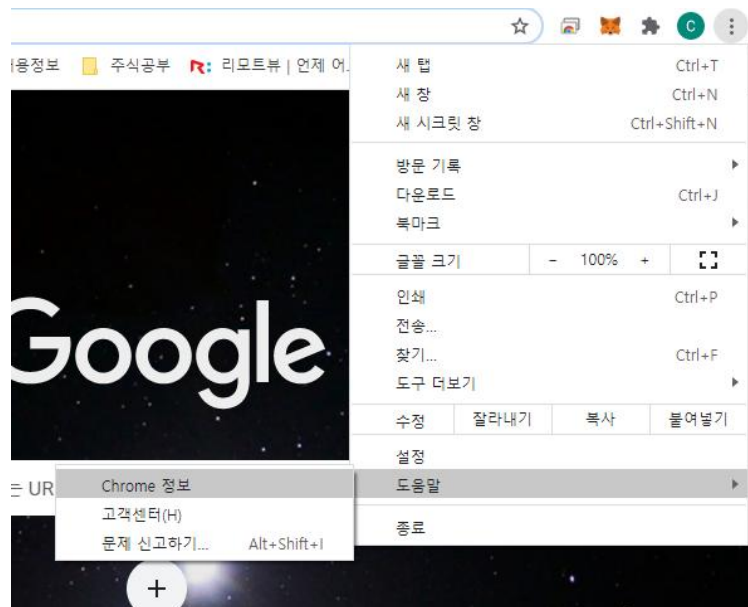
- > 왼쪽 검색영역에서 설치할 모듈명 입력 : 엔터
- > 오른쪽의 문서 부분 우측 위에 'install' 버튼 클릭하면 설치됨

또는 File > Settings > Project Interpreter > '+' 버튼 클릭 > 모듈 설치

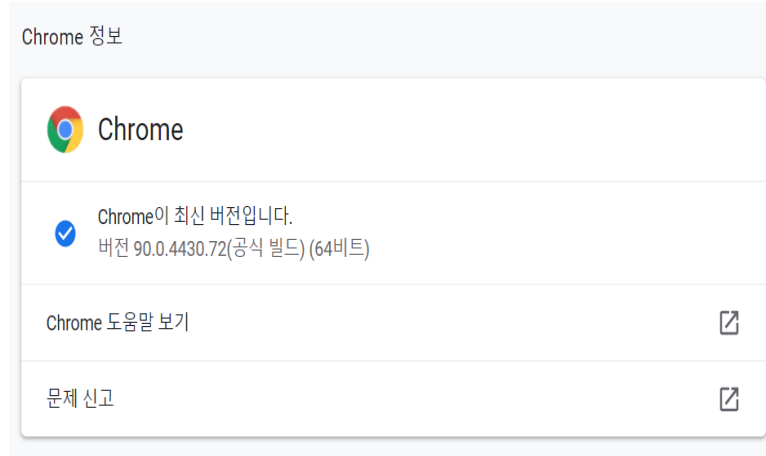
2. Chrome Driver 설치하기

Selenium 을 사용해서 웹 자동화를 위해 크롬 웹 드라이버를 설치해야 합니다.

2.1. 크롬 버전 확인하기



크롬을 실행한 후 오른쪽 위에 점 3 개 -> 도움말 -> Chrome 정버를 선택합니다.



그러면 크롬 버전을 바로 확인할 수 있습니다.

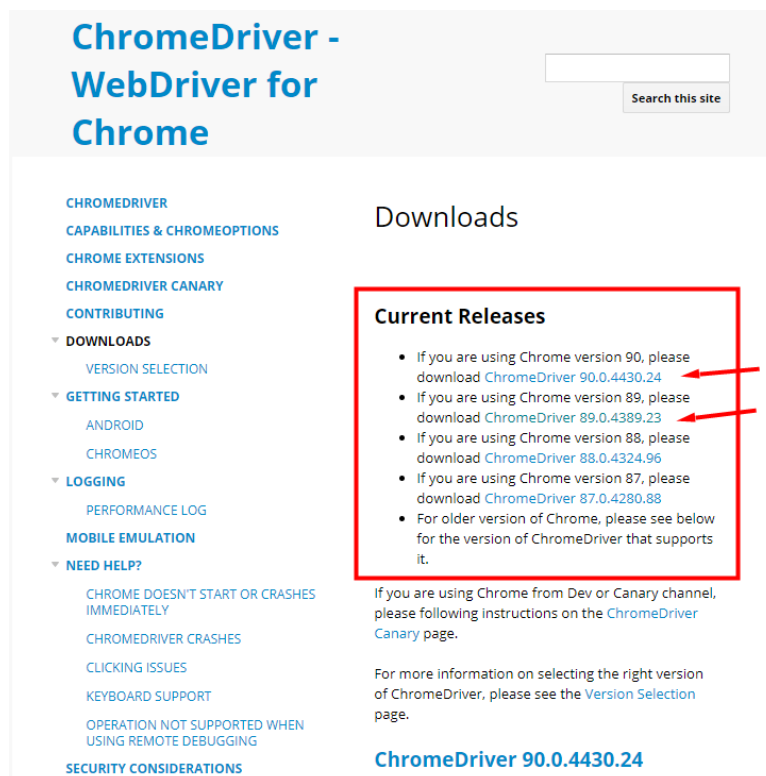
2.2. 크롬 드라이버 다운로드

웹에서 '크롬 드라이버' 로 검색하면,







ChromeDriver - WebDriver for Chrome 이라는 사이트에 들어가서 확인된 브라우저 버전에 대한 Driver 를 다운받습니다. (윈도우용은 win32.zip 으로 받으면 됩니다.)

다운받은 zip 파일을 압축을 풀면 chromedriver.exe 파일을 사용합니다.

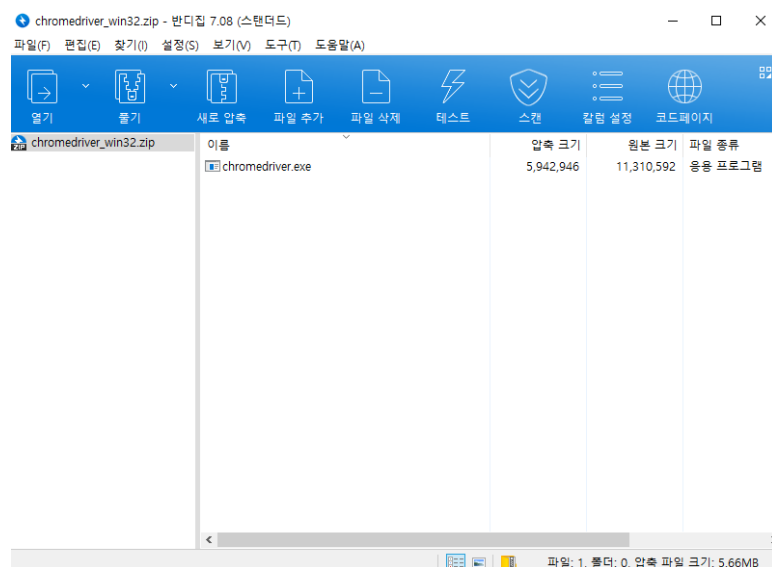
링크 : <https://chromedriver.chromium.org/downloads>



Index of /90.0.4430.24/

	Name	Last modified	Size	ETag
	Parent Directory		-	
	chromedriver_linux64.zip	2021-03-15 16:49:46	5.53MB	f132297377308392f3e5b44cf282f77a
	chromedriver_mac64.zip	2021-03-15 16:49:48	7.68MB	01378f44ca91150771859e254809fb66
	chromedriver_mac64_m1.zip	2021-03-15 16:49:50	7.01MB	9cd97b08730a9d395610d051b4aa2c05
	chromedriver_win32.zip	2021-03-15 16:49:51	5.67MB	eeb5e37fc4d4b21337a46576137a2053
	notes.txt	2021-03-15 16:49:56	0.00MB	a79b03d7895fbb145c4d3d0a63ba0d41

리눅스 vs MAC vs 윈도우 본인의 os 에 맞게 선택해주세요.



3. 드라이버 위치 변경

다운로드 받은 크롬 드라이버를 자신이 사용할 python 이 있는 위치 또는 자신이 사용하는 python 의 경로로 옮겨줍니다.

일반적으로 아나콘다를 사용하고 특정 경로를 사용하지 않으신다면 사용자 폴더 밑으로 설정되는 경우가 많으니 참고하시기 바랍니다.

```
In [7]: import os

        os.getcwd()

Out [7]: 'C:\\Users\\wbc\\py_workspace'
```

4. Selenium 사용하기

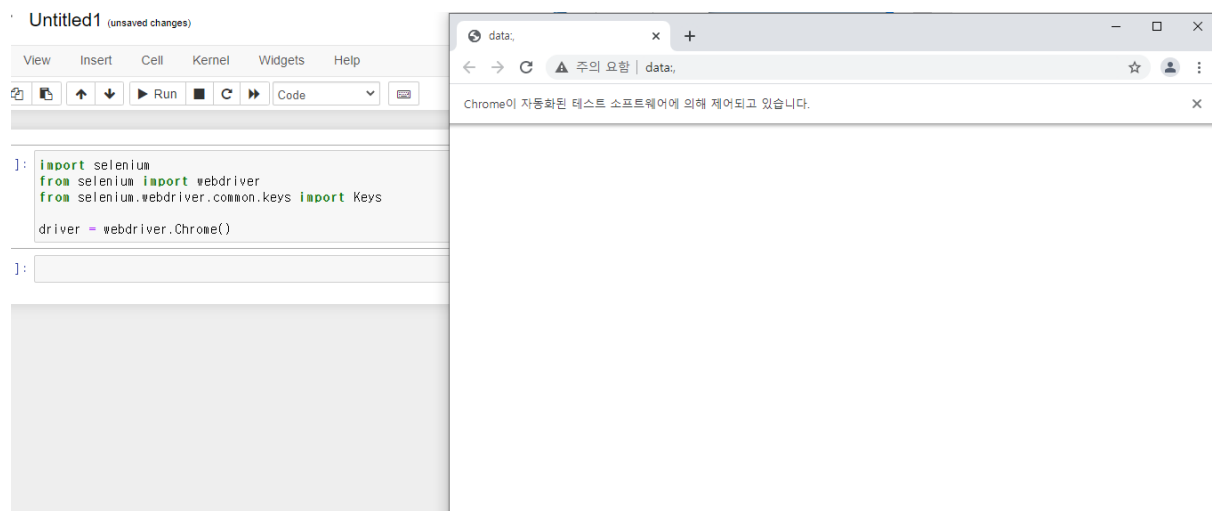
Selenium 과 Webdriver 가 잘 설치되어서 실행되는지 확인해보겠습니다.

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

driver = webdriver.Chrome()
```

selenium 을 import 해온 후 webdriver 를 실행시켜 보았습니다.

아래 사진과 같이 비어있는 크롬 창이 실행되었다면 잘 설치된 것입니다.



[Python selenium]

Import 선언

```
import selenium
from selenium import webdriver
from selenium.webdriver import ActionChains

from selenium.webdriver.common.keys import Keys
from selenium.webdriver.common.by import By

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

웹사이트 불러오기(Driver & Web Load)

```
URL = 'https://www.miraeassetdaewoo.com/hki/hki3028/r01.do'
```

```
driver = webdriver.Chrome(executable_path='./chromedriver.exe')
driver.get(url=URL)
```

먼저 webdriver.Chrome(executable_path) 함수를 사용하여 드라이버를 로드합니다.

별도의 폴더에 저장한 경로를 작성해도 됩니다.

여기서는 driver 라는 변수에 저장합니다.

다운로드한 브라우저 드라이브 파일이름이 chromedriver.exe 라면 경로는 소스파일과 같은 디렉토리인 경우 그냥 파일 이름(chromedriver)만 입력하면 됩니다. 확장자는 필요없으며, 파일 경로가 다르다면 상대경로나 절대경로를 이용합니다.

그리고 get(url) 함수를 사용하면, 해당 URL 을 브라우저에서 띄웁니다.

현재 url 얻기

참고로, 현재 url 은 다음 코드를 쓰면 됩니다.

```
print(driver.current_url)
```

브라우저 닫기

```
driver.close()
```

Wait till Load Webpage(로딩 대기)

브라우저에서 해당 웹 페이지의 요소들을 로드하는 데 시간이 좀 걸립니다.

그러므로 element 가 존재하지 않는다는 error 를 보고 싶지 않다면 해당 요소가 전부 준비가 될 때까지 대기해야 합니다.

Implicit Waits(암묵적 대기)

```
driver.implicitly_wait(time_to_wait=5)
```

찾으려는 element 가 로드될 때까지 지정한 시간만큼 대기할 수 있도록 설정합니다.

이는 한 webdriver 에 영구적으로 작용합니다. 인자는 초 단위이며, Default 값은 0 이다.

위의 예시는 5 초까지 기다려 준다는 의미이다.

Explicit Waits(명시적 대기)

간단하게는 time.sleep(secs) 함수를 사용하여 무조건 몇 초간 대기하는 방법이 있습니다.

효율이 아주 좋지 않으므로 사용하지 않는 것을 권합니다.

아래 코드를 살펴봅시다.

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

driver = webdriver.Chrome('chromedriver')
driver.get(url='https://www.google.com/')
try:
    element = WebDriverWait(driver, 5).until(
        EC.presence_of_element_located((By.CLASS_NAME , 'gLfyf'))
    )
finally:
    driver.quit()
```

위의 코드는 웹페이지에서 class 가 gLFyf 인 어떤 element 를 찾을 수 있는지를 최대 5 초 동안 매 0.5 초마다 시도합니다. expected_conditions(EC)는 만약 element 를 찾을 수 있었으면 True 를, 아니라면 False 를 반환합니다.

이 예시에서는 element 가 존재하는지를 조건으로 사용했는데, 이것 말고도 아래와 같은 예시들이 있습니다

제목이 어떤 문자열인지, 어떤 문자열을 포함하는지, 특정/모든 요소가 로드되었거나/볼 수 있거나/볼 수 없거나/클릭 가능하거나 등등의 여러 조건이 가능합니다.

- title_is
- title_contains
- presence_of_element_located
- visibility_of_element_located
- visibility_of
- presence_of_all_elements_located
- text_to_be_present_in_element
- text_to_be_present_in_element_value
- frame_to_be_available_and_switch_to_it
- invisibility_of_element_located
- element_to_be_clickable
- staleness_of
- element_to_be_selected
- element_located_to_be_selected
- element_selection_state_to_be
- element_located_selection_state_to_be
- alert_is_present

Custom 으로 조건을 설정하는 것도 가능한데, __init__ 함수와 __call__ 함수를 구현한 class 를 작성하면 됩니다. 여기를 참조합니다.

참고로, until(method, message="") 함수는 method 의 반환값이 False 인 동안 계속 method 를 실행합니다.

반대로 until_not(method, message="") 함수는 True 인 동안 실행합니다.

구글에 접속해서 검색기능 사용 코드 작성해 보기

먼저 다음 코드를 실행해보자. 앞에서 설명한 대로 chromedriver.exe 를 같은 위치에 놓았다면 문제 없이 실행될 것입니다.

구글 홈페이지가 바뀌었다면 오류가 생길 수도 있습니다.

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

options = webdriver.ChromeOptions()
options.add_argument('window-size=1920,1080')

driver = webdriver.Chrome('chromedriver', options=options)
driver.implicitly_wait(5)

driver.get(url='https://www.google.com/')

search_box =
driver.find_element_by_xpath('//*[@id="tsf"]/div[2]/div[1]/div[1]/div/div[2]/input')

search_box.send_keys('greeksharifa.github.io')
search_box.send_keys(Keys.RETURN)

elements = driver.find_elements_by_xpath('//*[@id="rso"]/div[*]/div/div[1]/a/h3/span')

for element in elements:
    print(element.text)
    print(element.text, file=open('gorio.txt', 'w', encoding='utf-8'))

sleep(3)
driver.close()
```

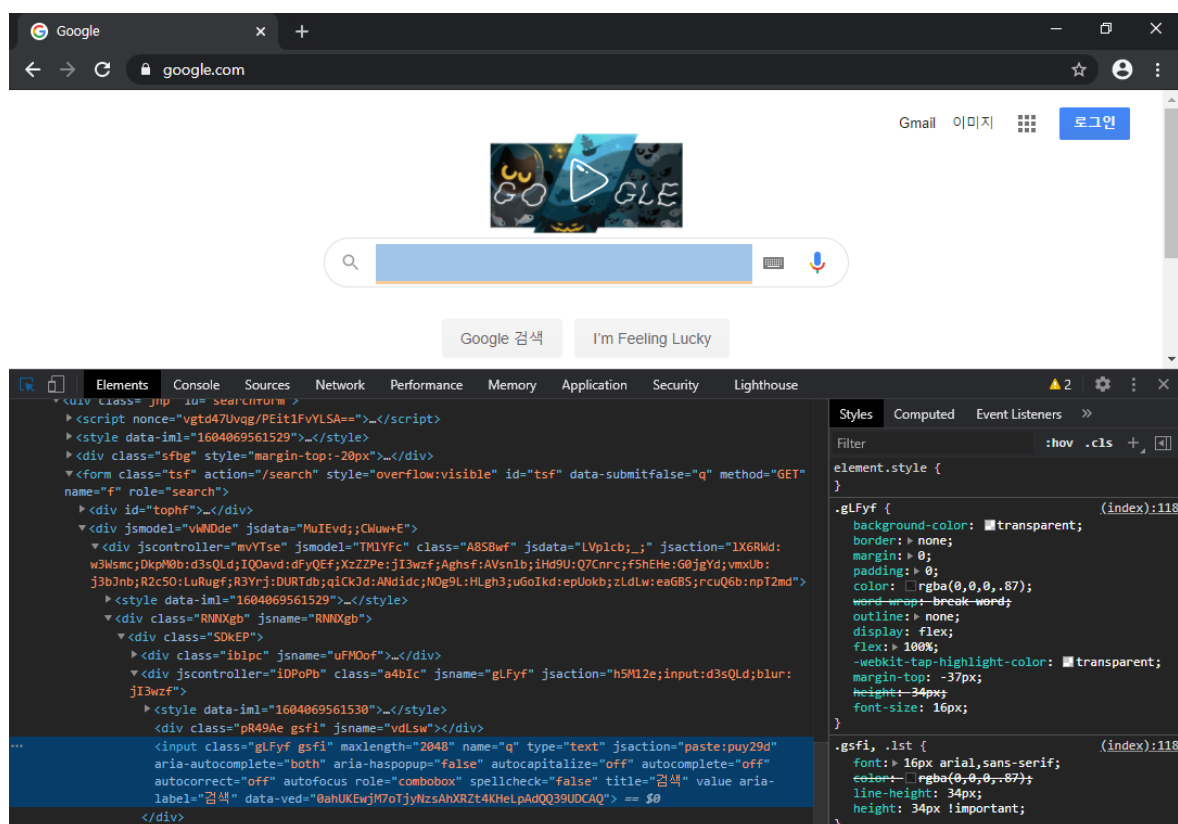

구글 검색의 결과가 출력되면서 동시에 `gorio.txt` 라는 파일이 생성되며 같은 내용이 출력되어 있음을 확인할 수 있습니다.

요소 찾기(Locating Elements)

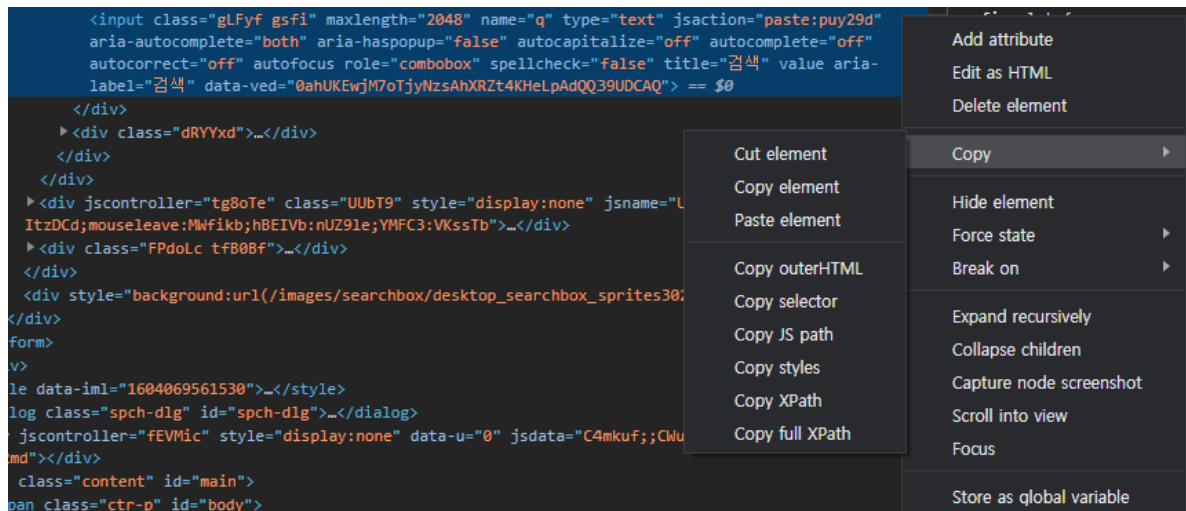
Selenium 은 다양한 요소(element)를 찾는 방법을 지원합니다.

HTML 을 조금 다뤄봤다면, class, id, parent-child 관계 등등을 알고 있을 것입니다.

먼저 어떤 요소를 찾을지부터 정해야 합니다. 그냥 크롬을 켜서, `Ctrl + Shift + C` 를 눌러 원하는 요소를 클릭합니다.



그러면 해당 요소의 정보를 볼 수 있을 것이다. 원하는 요소를 아래쪽 Elements 창에서 우클릭하여 내용을 복사하거나(Copy element 혹은 Copy XPath 등), 아니면 그냥 직접 보고 입력해도 됩니다.



```
<input class="gLFyf gsfi" maxlength="2048" name="q" type="text"
jsaction="paste:puy29d" aria-autocomplete="both" aria-haspopup="false"
autocapitalize="off" autocomplete="off" autocorrect="off" autofocus=""
role="combobox" spellcheck="false" title="검색" value="" aria-label="검색"
data-ved="0ahUKEwjM7oTjyNzsAhXRZt4KHlPAdQQ39UDCAQ">
```

각 요소에는 class 나, XPath 나, id 등의 여러 속성이 존재합니다. 이 속성이나 경로를 갖고 요소를 찾을 수 있습니다.

위에서 예시로 든 것은 구글의 검색창을 나타냅니다. 이 검색창은 id 가 없고 class 가 특징적인 것 같으니, class 로 찾아봅니다.

class 로 찾는 방법은 다음과 같습니다.

```
search_box = driver.find_element_by_class_name('gLFyf')
# 아래는 키보드 입력을 해 주는 코드이다. 나중에 설명하겠지만 한번 해 보자.
search_box.send_keys('gorio')
```

그러면 search_box 라는 변수에는 구글의 검색창 요소가 담겨 있는 것입니다. 선택한 요소에 키보드 입력을 보내거나, 클릭하는 등의 동작을 할 수 있습니다.

class 말고도 선택하는 방법은 많습니다. 위에서 class_name 로 끝나는 함수를 쓰는 대신, id, name, xpath, css_selector, 등으로 끝나는 함수를 사용할 수 있습니다.

```

driver.find
    m find_element(self, by, value)
    m find_element_by_class_name(self, name)
    m find_element_by_css_selector(self, css_selector)
    m find_element_by_id(self, id_)
    m find_element_by_link_text(self, link_text)
    m find_element_by_name(self, name)
    m find_element_by_partial_link_text(self, link_text)
    m find_element_by_tag_name(self, name)
    m find_element_by_xpath(self, xpath)
    m find_elements(self, by, value)
    m find_elements_by_class_name(self, name)
    m find_elements_by_css_selector(self, css_selector)
    m find_elements_by_id(self, id_)
    m find_elements_by_link_text(self, text)
    m find_elements_by_name(self, name)
    m find_elements_by_partial_link_text(self, link_text)
    m find_elements_by_tag_name(self, name)
    m find_elements_by_xpath(self, xpath)

```

총 18 개의 함수를 지원합니다. 9 개의 쌍이 있는데, `find_element` 로 시작하는 함수는 조건에 맞는 요소를 하나만 반환하고, `find_elements` 로 시작하는 함수는 해당 조건을 만족하는 모든 요소를 반복가능한(iterable) 형태로 반환합니다.

위에서 볼 수 있듯이 class 나, css selector, id, name, tag_name, xpath, link_text, partial_link_text 등으로 선택 가능합니다.

맨 위의 함수인 그냥 `find_element` 함수의 경우만 인자를 3 개 받습니다. `by` 는 조금 전 `explicit_waits` 에서 보았던 그 `selenium.webdriver.common.by` 입니다. `by` 도 `CLASS_NAME` 등으로 속성을 지정 가능합니다.

- ID = "id"
- XPATH = "xpath"
- LINK_TEXT = "link text"
- PARTIAL_LINK_TEXT = "partial link text"
- NAME = "name"
- TAG_NAME = "tag name"
- CLASS_NAME = "class name"
- CSS_SELECTOR = "css selector"

link_text 는 <a> tag 의 링크를 대상으로 찾습니다. 비슷하게 partial_link_text 는 요소의 링크가 전달한 링크를 일부분으로 포함되어 있으면 해당 요소가 선택됩니다.

```
<html>
<body>
  <p>Are you sure you want to do this?</p>
  <a href="continue.html">Continue</a>
  <a href="cancel.html">Cancel</a>
  <p class="content">Site content goes here.</p>
</body>
</html>
```

여기서 다음 코드로 찾을 수 있습니다.

```
continue_link = driver.find_element_by_link_text('Continue')
continue_link = driver.find_element_by_partial_link_text('Conti')
```

CSS Selector 는 다음 코드를 쓰면 됩니다.

```
content = driver.find_element_by_css_selector('p.content')
```

여기서 find_element_by_xpath 는 매우 강력한 찾기 기능을 제공합니다.

웹페이지 상에서 해당 요소의 전체경로(혹은 상대경로)를 갖고 찾기 기능을 수행할 수 있는데, 원하는 요소에서 Copy XPath 만 한 다음 그대로 갖다 쓰면 해당 요소를 정확히 찾아줍니다.

```
search_box =
driver.find_element_by_xpath('//*[@id="tsf"]/div[2]/div[1]/div[1]/div/div[2]/input')
```

똑같은 결과를 갖는 코드는 다음과 같습니다.

```
from selenium.webdriver.common.by import By

search_box = driver.find_element(By.XPATH,
'//*[@id="tsf"]/div[2]/div[1]/div[1]/div/div[2]/input')
```

XPath 로 요소 찾기

표현식	설명
nodename	nodename을 name으로 갖는 모든 요소 선택
/	root 요소에서 선택
//	현재 요소의 자손 요소를 선택
.	현재 요소를 선택
..	현재 요소의 부모 요소를 선택
@	속성(attributes)를 선택
*	모든 요소에 매치됨
@*	모든 속성 요소에 매치됨
node()	모든 종류의 모든 요소에 매치됨
	OR 조건의 기능

예시는 다음과 같습니다.

표현식	설명
/div	root 요소의 div 요소
./div	현재 요소의 자식 요소 중 div 요소
/*	name에 상관없이 root 요소를 선택
./* 또는 *	context 요소의 모든 자식 요소를 선택
//div	현재 웹페이지에서 모든 div 요소를 선택
./div	현재 요소의 모든 자손 div 요소를 선택
//*	현재 웹페이지의 모든 요소를 선택
.//*	현재 요소의 모든 자손 요소를 선택
/div/p[0]	root > div > p 요소 중 첫 번째 p 요소를 선택
/div/p[position()<3]	root > div > p 요소 중 첫 두 p 요소를 선택
/div/p[last()]	root > div > p 요소 중 마지막 p 요소를 선택
/bookstore/book[price>35.00]	root > bookstore > book 요소 중 price 속성이 35.00 초과인 요소들을 선택

표현식	설명
<code>//*[@id="tsf"]/div[2]/</code>	id가 tsf인 모든 요소의 자식 div 요소 중 3번째 요소를 선택
<code>//title //price</code>	title 또는 price 요소를 선택

- 참고

텍스트 입력(키보드 입력)

어떤 요소를 `find_element...` 함수를 통해 선택했다고 가정합니다.

```
search_box =
```

```
driver.find_element_by_xpath('//*[@id="tsf"]/div[2]/div[1]/div[1]/div/div[2]/input')
```

선택한 요소에 키보드 입력을 명령으로 주어 텍스트 입력 등을 수행할 수 있습니다.

키보드 입력은 `send_keys(*value)` 함수를 통해 할 수 있습니다.

```
search_box.send_keys('greeksharifa.github.io')
```

기본적으로 `send_keys(*value)` 함수는 문자열을 받는다. 그런데, `enter` 같은 특수 키 입력의 경우도 문자열로 처리할 수 있지만(`RETURN = '\ue006'`), 다음과 같이 입력할 수 있습니다.

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

```
search_box.send_keys(Keys.RETURN)
```

경우에 따라 조금씩 다르긴 하지만, 일반적으로 `enter` 키의

경우 `Keys.ENTER` 또는 `Keys.RETURN` 으로 입력할 수 있습니다.

아래는 입력할 수 있는 `Keys` 의 목록입니다.

```
class Keys(object):
```

```
    """
```

```
    Set of special keys codes.
```

```
    """
```

```
    NULL = '\ue000'
```

```
    CANCEL = '\ue001' # ^break
```

```
    HELP = '\ue002'
```

BACKSPACE = '\u0003'
BACK_SPACE = BACKSPACE
TAB = '\u0004'
CLEAR = '\u0005'
RETURN = '\u0006'
ENTER = '\u0007'
SHIFT = '\u0008'
LEFT_SHIFT = SHIFT
CONTROL = '\u0009'
LEFT_CONTROL = CONTROL
ALT = '\u000a'
LEFT_ALT = ALT
PAUSE = '\u000b'
ESCAPE = '\u000c'
SPACE = '\u000d'
PAGE_UP = '\u000e'
PAGE_DOWN = '\u000f'
END = '\u0010'
HOME = '\u0011'
LEFT = '\u0012'
ARROW_LEFT = LEFT
UP = '\u0013'
ARROW_UP = UP
RIGHT = '\u0014'
ARROW_RIGHT = RIGHT
DOWN = '\u0015'
ARROW_DOWN = DOWN
INSERT = '\u0016'
DELETE = '\u0017'
SEMICOLON = '\u0018'
EQUALS = '\u0019'

NUMPAD0 = 'Wue01a' # number pad keys

NUMPAD1 = 'Wue01b'

NUMPAD2 = 'Wue01c'

NUMPAD3 = 'Wue01d'

NUMPAD4 = 'Wue01e'

NUMPAD5 = 'Wue01f'

NUMPAD6 = 'Wue020'

NUMPAD7 = 'Wue021'

NUMPAD8 = 'Wue022'

NUMPAD9 = 'Wue023'

MULTIPLY = 'Wue024'

ADD = 'Wue025'

SEPARATOR = 'Wue026'

SUBTRACT = 'Wue027'

DECIMAL = 'Wue028'

DIVIDE = 'Wue029'

F1 = 'Wue031' # function keys

F2 = 'Wue032'

F3 = 'Wue033'

F4 = 'Wue034'

F5 = 'Wue035'

F6 = 'Wue036'

F7 = 'Wue037'

F8 = 'Wue038'

F9 = 'Wue039'

F10 = 'Wue03a'

F11 = 'Wue03b'

F12 = 'Wue03c'


```
META = 'Wue03d'
```

```
COMMAND = 'Wue03d'
```

텍스트 입력 지우기

위에 나와 있는 것처럼, 입력한 텍스트를 지우는

방법은 `Keys.BACKSPACE` 또는 `Keys.BACK_SPACE` 를 사용하는 것입니다.

만약 전체를 지우고 싶다면 `Keys` 가 아니라, 선택한 요소에서 `clear()` 함수를 호출하면 됩니다.

```
search_box.clear()
```

파일 업로드

파일을 받는 `<input>` 을 선택한 뒤, `send_keys(file_path)` 를 호출하면 됩니다.

```
upload = driver.find_element_by_tag('input')
```

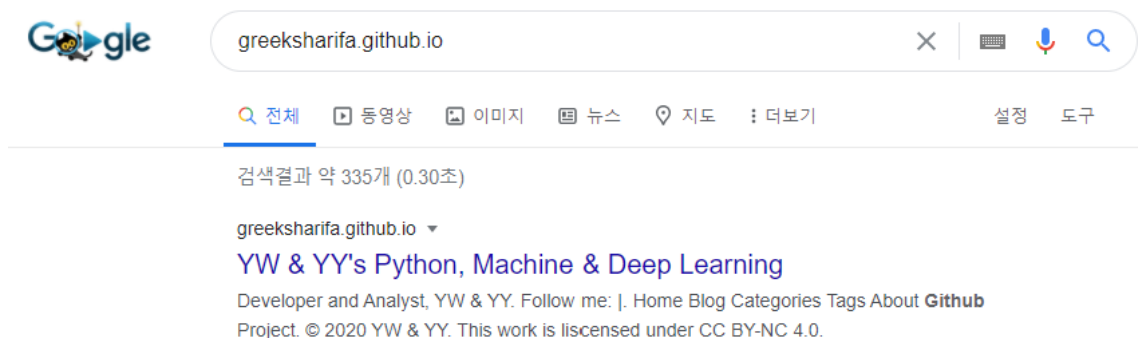
```
upload.send_keys(file_path)
```

상호작용

클릭하기(click)

`find_element` 함수로 요소를 선택한 다음에, `click()` 함수를 호출하면 됩니다.

`search_box.send_keys(Keys.RETURN)` 까지 입력했다면, 검색창은 다음과 같습니다.



이제 첫 번째 검색 결과를 클릭해 봅니다.

`Ctrl + Shift + C` 을 누른 뒤 제목 부분을 클릭하고, 위에서 설명한 `Copy XPath` 를 이용하여 `XPath` 를 얻습니다. `XPath` 는 다음과 같습니다.

```
//*[@id="rso"]/div[1]/div/div[1]/a/h3/span
```

다음 코드를 써 봅니다.

```
posting = driver.find_element_by_xpath('//*[@id="rso"]/div[1]/div/div[1]/a/h3/span')
posting.click()
```

실행하면 첫 번째 검색 결과가 클릭되어 다음과 같은 화면을 볼 수 있을 것입니다.



Andre Derain, Fishing Boats Collioure, 1905. France.

옵션 선택 및 제출(submit)

XPath 등으로 select 요소를 선택한 다음에 각 옵션을 선택할 수 있지만, 그것보다 더 좋은 방법이 있습니다.

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

```
select = Select(driver.find_element_by_name('select_name'))
```

```
select.select_by_index(index=2)
```

```
select.select_by_visible_text(text="option_text")
```

```
select.select_by_value(value='고리오')
```

selenium.webdriver.support.ui.Select 는 select 요소를 선택하여 쉽게 다룰 수 있도록 합니다.

위 코드에서 볼 수 있듯이 select 내에서 인덱스로 선택하거나, 옵션의 텍스트, 혹은 어떤 값을 통해 선택이 가능합니다.

특정 선택을 해제하려면 다음 코드를 사용합니다.

```
select.deselect_by_index(index=2)
select.deselect_by_visible_text(text="option_text")
select.deselect_by_value(value='고리오')
```

전부 해제

```
select.deselect_all()
```

선택된 옵션 리스트를 얻으려면 select.all_selected_options 으로 얻을 수 있고, 첫 번째 선택된 옵션은 select.first_selected_option, 가능한 옵션을 모두 보려면 select.options 를 사용하면 됩니다.

제출(submit)하려면 요소를 찾은 뒤 click()을 수행해도 되지만, 다음과 같이 써도 됩니다.

```
submit_btn.submit()
```

만약 선택한 요소가 form 형식이 아니라면 NoSuchElementException 오류를 볼 수 있을 것입니다.

Drag and Drop

어떤 일련의 동작을 수행하기 위해서는 ActionChains 를 사용하면 됩니다.

source 요소에서 target 요소로 Drag & Drop 을 수행한다고 합니다.

```
from selenium.webdriver import ActionChains
```

```
action_chains = ActionChains(driver)
action_chains.drag_and_drop(source, target).perform()
```

Window / Frame 이동

최신 웹 페이지에서는 frame 같은 것을 잘 사용하지 않지만, 예전에 만들어진 사이트라면 frame 을 사용한 경우가 있습니다.

이렇게 frame 안에 들어 있는 요소는 find_element 함수를 써도 그냥 찾아지지 않습니다.

find_element 함수는 frame 내에 있는 요소를 찾아주지 못합니다.

그래서 특정 frame 으로 이동해야 할 때가 있습니다.

```
driver.switch_to_frame("frameName")
driver.switch_to_window("windowName")
```

frame 내 subframe 으로도 접근이 가능하다. 점(.)을 쓰자.

```
driver.switch_to_frame("frameName.0.child")
```

windowName 을 알고 싶다면 다음과 같은 링크가 있는지 살펴봅니다.

```
<a href="somewhere.html" target="windowName">Click here to open a new
window</a>
```

혹은 webdriver 는 window 목록에 접근할 수 있기 때문에, 다음과 같이 모든 window 를 순회하는 것도 가능합니다.

```
for handle in driver.window_handles:
    driver.switch_to_window(handle)
```

frame 밖으로 나가려면 다음과 같이 쓰면 기본 frame 으로 되돌아갑니다.

```
driver.switch_to_default_content()
```

경고창으로 이동할 수도 있습니다.

```
alert = driver.switch_to.alert
```

경고창은 여기서 다룹니다.

JavaScript 코드 실행

driver.execute_script() 함수를 실행할 수 있습니다.

아래는 Name 이 search_box 인 요소의 값을 query 의 값으로 변경하는 코드입니다.

```
driver.execute_script("document.getElementsByName('id')[0].value='"+query+"'")
```

브라우저 창 다루기

뒤로가기, 앞으로 가기

브라우저는 뒤로가기(back)와 앞으로 가기(forward) 기능을 제공합니다.

이를 selenium 으로 구현이 가능합니다.

```
driver.forward()
driver.back()
```

.

화면 이동(맨 밑으로 내려가기 등)

크롤링을 하다 보면 화면의 끝으로 내려가야 내용이 동적으로 추가되는 경우를 자주 볼 수 있습니다.

이런 경우에는 웹페이지의 최하단으로 내려가는 코드를 실행할 필요가 있습니다.

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

물론 전체를 내려가야 할 필요가 없다면 document.body.scrollHeight 대신 지정된 값만큼 이동해도 됩니다.

특정 요소까지 계속 찾으려면 ActionChain 을 써도 됩니다.

```
from selenium.webdriver import ActionChains

some_tag = driver.find_element_by_id('gorio')

ActionChains(driver).move_to_element(some_tag).perform()
```

브라우저 최소화/최대화

```
driver.minimize_window()
driver.maximize_window()
```

Headless 설정

브라우저 창을 띄우지 않고 수행하는 방법입니다.

브라우저 크기 설정

기본적인 사용법은 다음과 같다. 브라우저가 실행될 때 창 크기를 설정할 수 있다.

```
options = webdriver.ChromeOptions()
options.add_argument('window-size=1920,1080')
```

```
driver = webdriver.Chrome('chromedriver', options=options)
```

다른 기능들은 여기에 적어 두었다. 코드를 보면 역할을 짐작할 수 있을 것이다.

```
options.add_argument('headless')
options.add_argument('window-size=1920x1080')
options.add_argument('disable-gpu')

options.add_argument('start-maximized')
options.add_argument('disable-infobars')
options.add_argument('--disable-extensions')

options.add_experimental_option('excludeSwitches', ['enable-automation'])
options.add_experimental_option('useAutomationExtension', False)
options.add_argument('--disable-blink-features=AutomationControlled')

options.add_experimental_option('debuggerAddress', '127.0.0.1:9222')
.
```

스크린샷 저장

```
driver.save_screenshot('screenshot.png')
```

Option(ChromeOption)

여러 옵션을 설정할 수 있습니다. 브라우저의 창 크기, 해당 기기의 정보 등을 설정할 수 있습니다.

기본적인 사용법은 다음과 같습니다. 브라우저가 실행될 때 창 크기를 설정할 수 있습니다.

```
options = webdriver.ChromeOptions()
options.add_argument('window-size=1920,1080')

driver = webdriver.Chrome('chromedriver', options=options)
```

다른 기능들은 코드를 보면 역할을 짐작할 수 있을 것입니다.

```
options.add_argument('headless')
options.add_argument('window-size=1920x1080')
options.add_argument('disable-gpu')

options.add_argument('start-maximized')
options.add_argument('disable-infobars')
options.add_argument('--disable-extensions')
```

```
options.add_experimental_option('excludeSwitches', ['enable-automation'])
options.add_experimental_option('useAutomationExtension', False)
options.add_argument('--disable-blink-features=AutomationControlled')

options.add_experimental_option('debuggerAddress', '127.0.0.1:9222')
```

ActionChains (마우스, 키보드 입력 등 연속 동작 실행)

```
from selenium.webdriver import ActionChains

menu = driver.find_element_by_css_selector('.nav')
hidden_submenu = driver.find_element_by_css_selector('.nav #submenu1')

ActionChains(driver).move_to_element(menu).click(hidden_submenu).perform()

# 위 한 줄은 아래와 같은 동작을 수행합니다.
actions = ActionChains(driver)
actions.move_to_element(menu)
actions.click(hidden_submenu)
actions.perform()
```

마우스 클릭, Drag & Drop, 키보드 입력 등을 연속적으로 수행할 수 있습니다.

- `on_element` 인자를 받는 함수는, 해당 인자가 주어지지 않으면 현재 마우스 위치를 기준으로 합니다.
- `element` 인자를 받는 함수는, 해당 인자가 주어지지 않으면 현재 선택이 되어 있는 요소를 기준으로 합니다.
- `key_down`, `key_up` 함수는 Ctrl 등의 키를 누를 때 쓰면 됩니다.

```
# Ctrl + C 를 누른다.
ActionChains(driver).key_down(Keys.CONTROL).send_keys('c').key_up(Keys.CONTROL).perform()
```

동작 수행 함수	설명
click(on_element=None)	인자로 주어진 요소를 왼쪽 클릭합니다.
click_and_hold(on_element=None)	인자로 주어진 요소를 왼쪽 클릭하고 누르고 있는다.
release(on_element=None)	마우스를 주어진 요소에서 떼낸다.
context_click(on_element=None)	인자로 주어진 요소를 오른쪽 클릭합니다.
double_click(on_element=None)	인자로 주어진 요소를 왼쪽 더블클릭합니다.
drag_and_drop(source, target)	source 요소에서 마우스 왼쪽 클릭하여 계속 누른 채로 target까지 이동한 뒤 마우스를 놓는다.
drag_and_drop_by_offset(source, xoffset, yoffset)	위와 비슷하지만 offset만큼 이동하여 마우스를 놓는다.
key_down(value, element=None)	value로 주어진 키를 누르고 떼지 않는다.
key_up(value, element=None)	value로 주어진 키를 떼낸다.
move_to_element(to_element)	마우스를 해당 요소의 중간 위치로 이동합니다.
move_to_element_with_offset(to_element, xoffset, yoffset)	마우스를 해당 요소에서 offset만큼 이동한 위치로 이동합니다.
pause(seconds)	주어진 시간(초 단위)만큼 입력을 중지합니다.
perform()	이미 쌓여 있는(stored) 모든 행동을 수행한다(chaining).
reset_actions()	이미 쌓여 있는(stored) 모든 행동을 제거합니다.
send_keys(*keys_to_send)	키보드 입력을 현재 focused된 요소에 보낸다.
send_keys_to_element(element, *keys_to_send)	키보드 입력을 주어진 요소에 보낸다.

경고 창 다루기(alerts)

브라우저를 사용하다보면 상단에 경고창이 뜰 때가 있습니다. (확인/취소 등)

이 경고창을 무시하는 등의 처리를 할 수 있는 기능을 제공합니다.

아래 코드는 경고창에서 수락/거절을 누르거나, 경고창의 내용을 출력, 혹은 경고창에 특정 키 입력을 보낼 수 있습니다.


```
from selenium.webdriver.common.alert import Alert
```

```
Alert(driver).accept()
```

```
Alert(driver).dismiss()
```

```
print(Alert(driver).text)
```

```
Alert(driver).send_keys(keysToSend=Keys.ESCAPE)
```

기타 기능

- Touch Actions: 마우스/키보드 입력과 비슷하게 chaining 이 가능하다. 터치와 관련한 여러 기능을 제공합니다.
- `selenium.webdriver.common.touch_actions.TouchActions`
- Proxy: Proxy 기능을 사용할 수 있습니다. `selenium.webdriver.common.proxy.Proxy`
- 쿠키(Cookies): 쿠키를 추가하거나 가져올 수 있습니다.

```
# Go to the correct domain
```

```
driver.get('http://www.example.com')
```

```
# Now set the cookie. This one's valid for the entire domain
```

```
cookie = {'name' : 'foo', 'value' : 'bar'}
```

```
driver.add_cookie(cookie)
```

```
# And now output all the available cookies for the current URL
```

```
driver.get_cookies()
```