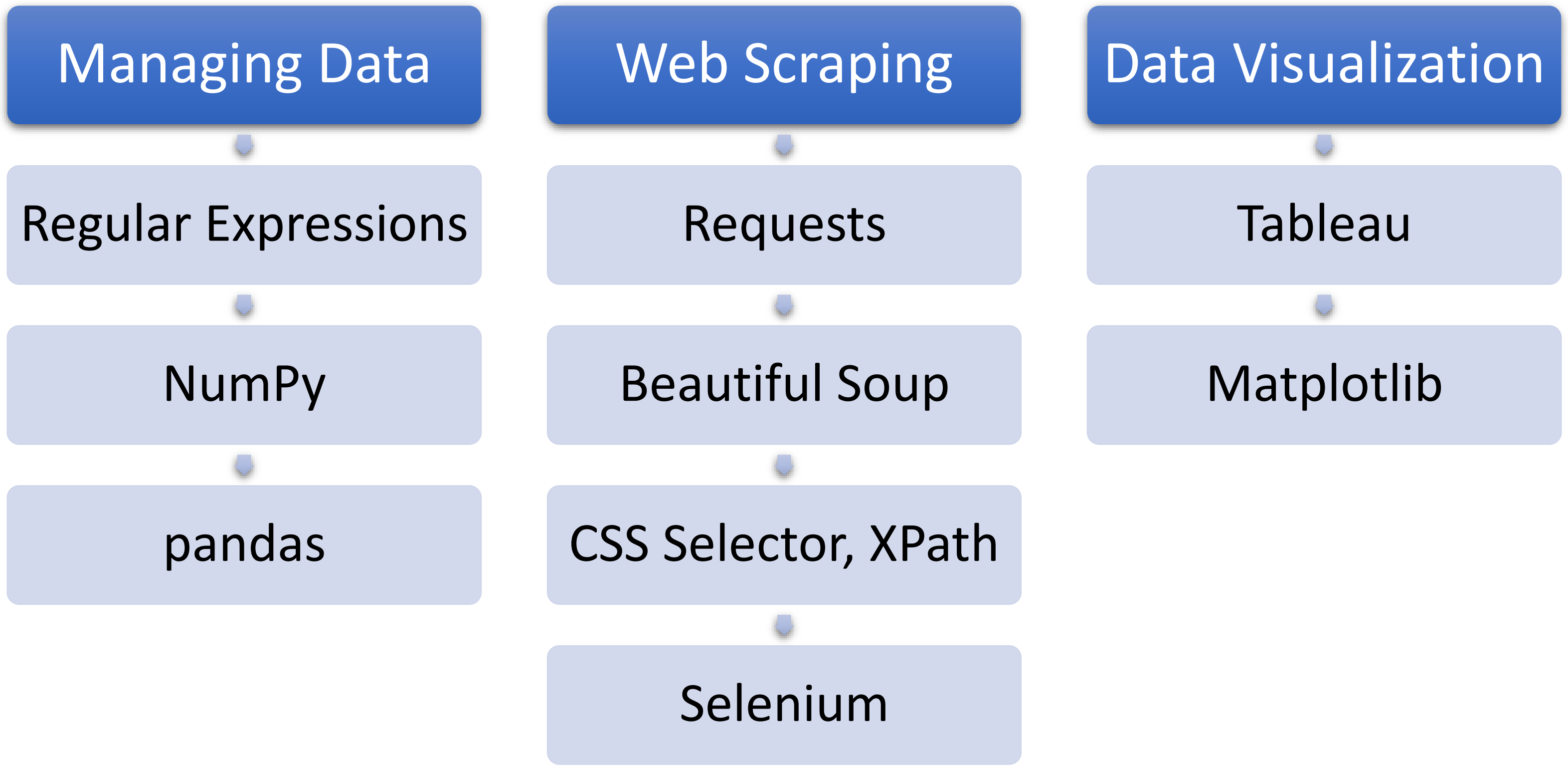
Web Scraping III

MSBA7001 Business Intelligence and Analytics
HKU Business School
The University of Hong Kong

Instructor: Dr. DING Chao

Course Roadmap



Selenium

What is Selenium?

- Selenium is an open-source automation testing framework designed for automating web browsers, allowing testers and developers to write scripts in various programming languages, including Java, C#, and Python.
- It provides tools and libraries to interact with web applications, facilitating tasks like clicking buttons, filling out forms, and verifying content.

What is Selenium?

- Selenium has four major components:
  - Selenium Integrated Development Environment (IDE)
  - Selenium Remote Control (RC)
  - **Selenium WebDriver**
  - Selenium Grid

Install Selenium 4

- To add selenium 4 to conda environment, run the following command in terminal:

```
conda install conda-forge::selenium
```

- Or, use pip (Python package management system). Run the following command in terminal:

```
pip install selenium==4.4.3
```

- See more details on the official page.

Install WebDriver

- Selenium requires a driver to interface with the chosen browser such as Chrome, Firefox, Edge and Safari.
- Make sure your browser is updated to the lasted version.
- Download the latest driver and place it in the same folder as your script file.

Browser	Where to download
Chrome	https://chromedriver.chromium.org/downloads
Edge	https://developer.microsoft.com/en-us/microsoft-edge/tools/webdriver/
Firefox	https://github.com/mozilla/geckodriver/releases
Safari	https://webkit.org/blog/6900/webdriver-support-in-safari-10/

Load a Web Page

- We start by loading a webpage in the current browser (Chrome) session.

Use relative path.
For Mac, without .exe

```
from selenium import webdriver
from selenium.webdriver.chrome.service import Service

service = Service(r'./chromedriver.exe')
driver = webdriver.Chrome(service = service)

driver.get(r'https://ug.hkubs.hku.hk/course')
```

Loads the webpage.

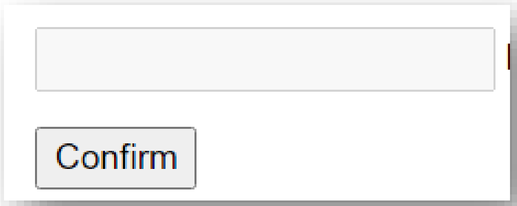
Creates a WebDriver object.
Think of it as a browser.

WebDriver Methods (partial)

Method	Description
add_cookie()	Adds a cookie to your current session.
back()	Goes one step backward in the browser history.
close()	Closes the current window.
delete_all_cookies()	Deletes all cookies in the scope of the session.
execute_script()	Executes JavaScript in the current window/frame.
forward()	Goes one step forward in the browser history.
get_screenshot_as_png()	Gets the screenshot of the current window as a binary data.
quit()	Quits the driver and closes every associated window.
refresh()	Refreshes the current page.
set_page_load_timeout()	Sets the amount of time to wait for a page load to complete before throwing an error.
current_url	Gets the URL of the current page.
page_source	Gets the source of the current page.
title	Returns the title of the current page.

Find Web Elements

- We primarily use the following two methods to locate web elements on a page:
  - **`find_element`**: returns the first matched element
  - **`find_elements`**: returns a list of matched elements
- It's possible to search by element ID, class name, CSS selector, XPath, etc. Simply implement the **`By`** class.



```
<input type = "text" id = "search"></input>
<br>
<button type = "submit" class = "confirm">Confirm</button>
```

```
from selenium.webdriver.common.by import By
inquiry = driver.find_element(By.ID, 'search')
button = driver.find_element(By.CLASS_NAME, 'confirm')
```

The `By` Class

Attribute	Description
By.ID	The first element with the id attribute value matching the location will be returned.
By.NAME	The first element with the name attribute value matching the location will be returned.
By.XPATH	The first element with the xpath syntax matching the location will be returned.
By.LINK_TEXT	The first element with the link text value matching the location will be returned.
By.PARTIAL_LINK_TEXT	The first element with the partial link text value matching the location will be returned.
By.TAG_NAME	The first element with the given tag name will be returned.
By.CLASS_NAME	The first element with the matching class attribute name will be returned.
By.CSS_SELECTOR	The first element with the matching CSS selector will be returned.

Interact with Elements

- Once an element is located, we could apply various operations on the element.
  - Click a button
  - Send text
  - Extract its text
  - Take a screenshot
  - ……
- For example, click on the “Confirm” button.

```
button = driver.find_element(By.CLASS_NAME, 'confirm')
button.click()
```

Element Methods (partial)

Method	Description
is_selected()	Checks if element is selected or not. It returns a boolean value True or False.
is_displayed()	Checks if element it visible to user or not. It returns a boolean value True or False.
get_attribute()	Gets attributes of an element, such as getting href attribute of anchor tag.
send_keys()	Sends text to any field, such as input field of a form or even to anchor tag paragraph, etc.
click()	Clicks on any element, such as an anchor tag, a link, etc.
clear()	Clears text of any field, such as input field of a form or even to anchor tag paragraph, etc.
screenshot()	Saves a screenshot of current element to a PNG file.
submit()	Submits a form after you have sent data to a form.
parent	Gets internal reference to the WebDriver instance this element was found from.
tag_name	Gets name of tag you are referring to.
text	Gets text of current element.

Interact with INPUT Elements

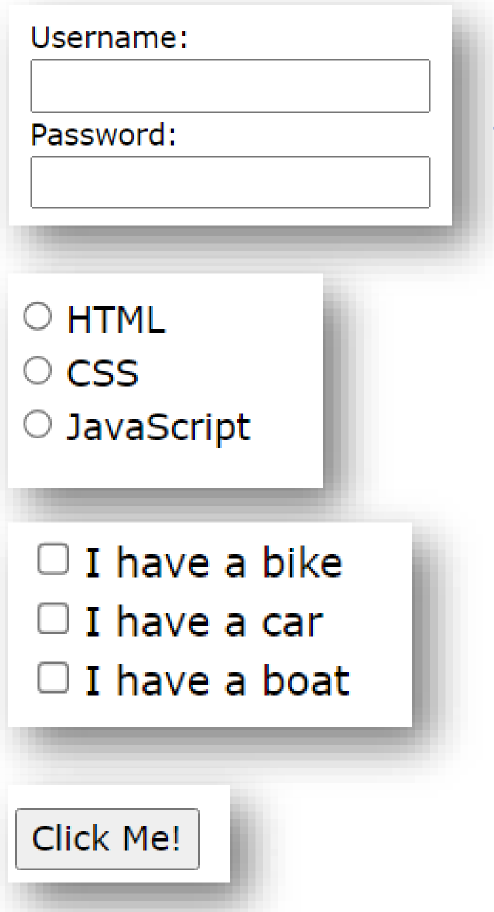
<input type="text"> → `send_keys()`

<input type="radio">

<input type="checkbox">

<input type="button">

`click()`



- See other input types here: https://www.w3schools.com/html/html_form_input_types.asp

The `Keys` Class

- We can perform all the keyboard operations with the help of the **`Keys`** class.
- For example, press ENTER after sending “IIMT” to the inquiry element.

```
from selenium.webdriver.common.keys import Keys
inquiry = driver.find_element(By.ID, 'search’)
inquiry.send_keys('IIMT', Keys.ENTER)
```

The `Keys` Class

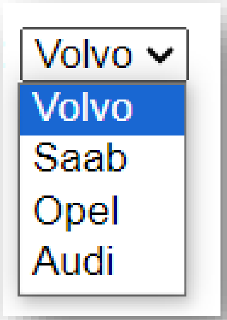
- Available keys below. To use a key: `Keys.PAGE_UP`

ADD	ALT	ARROW_DOWN	HELP
ARROW_LEFT	ARROW_RIGHT	ARROW_UP	HOME
BACKSPACE	BACK_SPACE	CANCEL	INSERT
CLEAR	COMMAND	CONTROL	LEFT
DECIMAL	DELETE	DIVIDE	LEFT_ALT
DOWN	END	ENTER	LEFT_CONTROL
EQUALS	ESCAPE	F1 ~ F9	LEFT_SHIFT
META	MULTIPLY	NULL	PAGE_DOWN
PAGE_UP	PAUSE	RETURN	RIGHT
SEMICOLON	SEPARATOR	SHIFT	SPACE
SUBTRACT	TAB	NUMPAD0 ~ NUMPAD9	

The **Select** Class

- The **Select** class works with the HTML SELECT element. We use it to select or deselect items in a dropdown menu.

Methods
select_by_index()
select _by_value()
select _by_visible_text()
deselect_by_index()
deselect _by_value()
deselect _by_visible_text()
is_multiple()
options()
first_selected_option()
all_selected_options()
deselect_all()



```
<select id = "car">
  <option value="volvo">Volvo</option>
  …
</select>
```

```
from selenium.webdriver.support.select import Select
car = Select(driver.find_element(By.ID, 'car'))
car.select_by_visible_text('Volvo')
```

ActionChains

- ActionChains are a way to automate low-level interactions such as mouse movements, mouse button actions, keypress, and context menu interactions.

```
from selenium.webdriver.common.action_chains
import ActionChains
```

- Actions can be executed in a chain pattern:

```
ActionChains(driver).send_keys_to_element(inquiry,
"MSBA").click(button).perform()
```

- Or, actions can be queued up one by one, then performed.

ActionChains Methods (partial)

Method	Description
click()	Clicks an element.
click_and_hold()	Holds down the left mouse button on an element.
context_click()	Performs a context-click (right click) on an element.
double_click()	Double-clicks an element.
drag_and_drop()	Holds down the left mouse button on the source element, then moves to the target element and releases the mouse button.
key_down()	Sends a key press only, without releasing it.
key_up()	Releases a modifier key.
move_to_element()	Moves the mouse to the middle of an element.
perform()	Performs all stored actions.
pause()	Pause all inputs for the specified duration in seconds
release()	Releases a held mouse button on an element.
reset_actions()	Clears actions that are already stored locally and on the remote end
send_keys()	Sends keys to current focused element.
send_keys_to_element()	Sends keys to an element.

Waits

- When a page is loaded by the browser, the elements within that page may load at different time intervals, leading to ElementNotVisibleException when locating an element.
- Waiting provides some slack between actions performed – mostly locating an element or any other operation with the element.
- We may use time.sleep() to simulate waiting, but wait for how long?
- Selenium WebDriver provides two types of waits – implicit & explicit.

Explicit Waits

- An explicit wait is a code you define to wait for a certain condition to occur before proceeding further in the code.
  - When a certain element shows up
  - When a certain element is selected
  - When a new window is opened
  - When the title become XXX
  - ……
- This condition can be specified by using the WebDriverWait class in combination with expected_conditions:

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

Explicit Waits

- For example: wait up to 10 seconds unless it finds the element to return within 10 seconds.

until
until_not

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

title_is
title_contains
presence_of_element_located
visibility_of_element_located
visibility_of
presence_of_all_elements_located
element_located_to_be_selected
element_selection_state_to_be
element_located_selection_state_to_be
alert_is_present
……

Implicit Waits

- An implicit wait tells WebDriver to wait a certain amount of time when trying to find any element (or elements) not immediately available. This is a global setting that applies to every element location call for the entire session. The default value is 0.

```
driver.implicitly_wait(10)
```

- As soon as the element is located, the driver will return the element and the code will continue executing, so a larger implicit wait value won't necessarily increase the duration of the session.
- *Warning*: Do not mix implicit and explicit waits. Doing so can cause unpredictable wait times. For example, setting an implicit wait of 10 seconds and an explicit wait of 15 seconds could cause a timeout to occur after 20 seconds.

Common Errors

- InvalidSelectorException: The CSS or XPath selector you are trying to use has invalid characters or an invalid query.
  - Possible solution: copy + paste CSS or XPath from source
- NoSuchElementException: The element can not be found at the exact moment you attempted to locate it.
  - Possible solution: add explicit or implicit wait
- StaleElementReferenceException: An element goes stale when it was previously located, but can not be currently accessed. This often happens when you have refreshed the page, or the page has dynamically changed.
  - Possible solution: Always relocate the element every time you go to use it. Try not to refresh a page.

Selenium Best Practices

- Test actions one by one, then create a chain.
- Choose the best element locator: ID, class name, CSS, XPath.
- Take screenshots when a test fails.
- Clear cookies to remove previous visit history:
    - User preference
    - Login
    - Authentication
    - Shopping carts
    - Recently viewed

```
driver.delete_all_cookies()
```

Selenium Best Practices

- **Add a user agent and other options:**

```
from selenium.webdriver.chrome.options import Options

options = Options()

custom_user_agent = "MyUserAgent"

options.add_argument(f'user-agent={custom_user_agent}')

driver = webdriver.Chrome(service=service, options=options)
```

accept_insecure_certs
browser_version
page_load_strategy
timeouts
……

Selenium Best Practices

- **Handle Exceptions**: When your script encounters an error and crashes, it could lose all the data that it has scraped. To avoid this, you should handle exceptions and errors properly in your code.

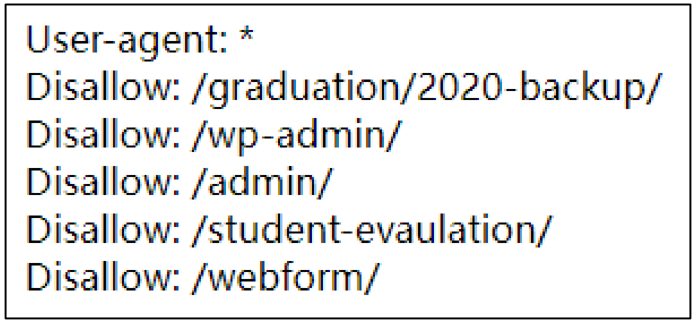
```
from selenium.common.exceptions import NoSuchElementException

try:
    element = driver.find_element(By.ID, 'abccba')
except NoSuchElementException:
    print('Element does not exist')
```

ERROR_URL
ElementNotVisibleException
InvalidSelectorException
NoSuchWindowException
StaleElementReferenceException
TimeoutException
……

Selenium Best Practices

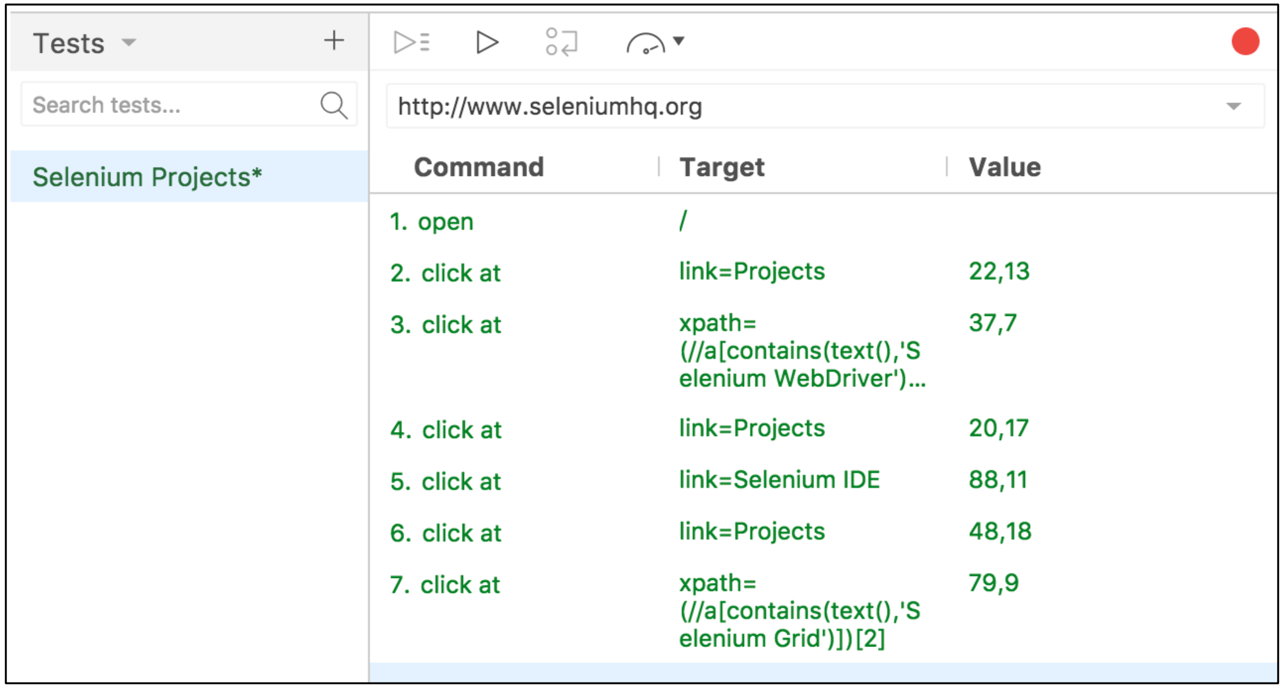
- **Respect Robots.txt**: Always check the website's robots.txt file before scraping.
- It's a file that tells search engine crawlers which URLs the crawler can access on your site. Some sites explicitly disallow certain actions which you should respect.
- The file lives at the root of a site. So, the robots.txt file for Google lives at www.google.com/robots.txt



```
User-agent: *
Disallow: /graduation/2020-backup/
Disallow: /wp-admin/
Disallow: /admin/
Disallow: /student-evaulation/
Disallow: /webform/
```

Selenium Best Practices

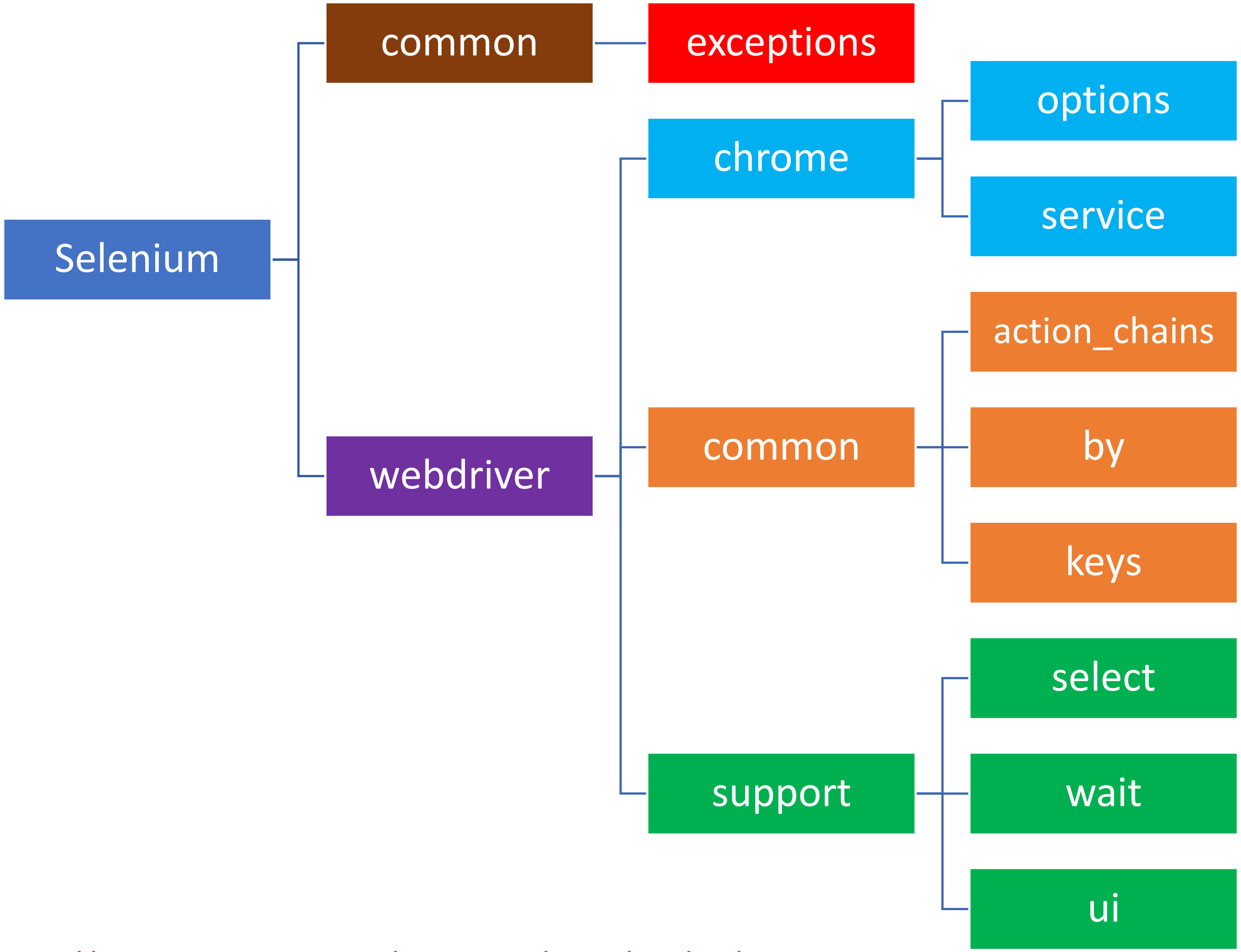
- Use **Selenium IDE** to record and playback test automation.
- Automation can be further exported as scripts, which can be used as a reference.
- To use it, add Selenium IDE as an extension to your browser.



http://www.seleniumhq.org

Command	Target	Value
1. open	/	
2. click at	link=Projects	22,13
3. click at	xpath=(//a[contains(text(),'Selenium WebDriver')...	37,7
4. click at	link=Projects	20,17
5. click at	link=Selenium IDE	88,11
6. click at	link=Projects	48,18
7. click at	xpath=(//a[contains(text(),'Selenium Grid')])[2]	79,9

Commonly Used Selenium Modules



See more: https://www.selenium.dev/selenium/docs/api/py/api.html

Install Tableau

- Go to the following page, select "Download Tableau Desktop". You need to enter your name and HKU email.

  https://www.tableau.com/tft/activation

- Install and activate with product key:

  TCTC-276A-2600-03F1-66ED

- This key is specifically licensed to the students in this course with limited activations. DO NOT share the key with others.
- The key is valid until the end of February, 2025
- You may also apply for a one-year student license for free:

  https://www.tableau.com/academic/students