Top Ten
DATE
PAGE

* Packages to be imported:

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

* Calling different browsers:

```
driver = webdriver.Chrome/Firefox/Ie(executable_
        path = "C:\.....\IEDriverServer.exe")
```

print(driver.title) → Title of the page.
print(driver.current_url) → Returns URL of page.
print(driver.page_source) → Returns HTML code for page.

Clicking the button:

```
driver.find_element_by_xpath("//*[@id='Tabbed']/
                              a/button").click()
```

driver.close() → closes only one current window, not all windows.
driver.quit() → closes all browsers
driver.maximize_window()/ → To maximize the size of
driver.fullscreen_window() opened browser
↳ Task bar and browser's menu bar will not be visible.

* Navigation Commands:

driver.back() → to go back and open previous page
driver.forward()/refresh() → to go ahead and open page

* Conditional Commands: always returns true or false (about Visibility)

↳ is_displayed(), is_enabled() can be used for any web elements whereas is_selected() can be used for checkboxes or radio buttons or dropdowns

ele = driver.find_element_by_xpath("//*[@id='pus']")
print(ele.is_displayed()) → check presence of webelement
print(ele.is_enabled()) → primarily for buttons

ele.send_keys("Venkatesh") → to enter inputs
ele.send_keys(Keys.ENTER) → to press enter button
ARROW_DOWN

* Wait commands:

Synchronization is basically balancing between code execution and response of application.
eg: While executing the code, if the element is not enabled, then it will skipped and execution will be continued and at last, error will be displayed where element was not enabled/load

Top Ten
DATE
PAGE

↳ to solve it, we will use waits.

→ Implict wait is applicable for all elements present in the page. It is time based and affects execution and it doesn't catch performance issue in application.
driver.implicitly_wait(10)

here, there is a chance of exception because if the element is not loaded within given period of time,

→ Explicit wait is based on condition not upon time. it works on expected condition. so, we will import it as

① from selenium.webdriver.common.by import By
from selenium.webdriver.support import expected_conditions as EC
wait = WebDriverWait(driver, 10)
element = wait.until(EC.element_to_be_clickable
((By.XPATH, "//*[@id='stop']")))
element.click()
from selenium.webdriver.support.ui import
WebDriverWait

↳ Explicit wait is prefered over time.sleep() / thread.sleep()

② driver.find_element(By.ID, "flightdate").clear()
driver.find_element(By.ID, "flightdate").send_keys
("15/10/2018")

Top Ten
DATE
PAGE

* Working with input boxes: → text field (visible text)
  → Password field (Encrypted text)

* Count of input boxes:

```
ib = driver.find_elements(By.CLASS_NAME, 'text-field')
print(len(ib))
```

* Providing inputs into input boxes:

① driver.find_element(By.ID, 'RESULT_TextField-2').send_keys("pavan")

```
driver.find_element_by_id("RESULT-TextField-2").send_keys("Pavan")
```

* Checking the status of input box: enabled or not

↳ It will return true or false

```
status = driver.find_element(By.ID, "Result").is_displayed() / is_enabled()
```

* Working with radio buttons & check boxes:

① driver.find_element_by_id("Result").click()

② status = driver.find_element_by_id("Result").
is_selected()

↳ States of radio button: Normal, hover, checked, disabled, Disabled & checked.

* Working with dropdown:

Select values from dropdown:

From selenium.webdriver.support.ui import Select
element = driver.find_element_by("Radio")
drop = Select(element)
print(len(drop.options)) → returns the count
① drop.select_by_visible_text('Morning')
② drop.select_by_index(2)
③ drop.select_by_value("Radio-2")

✓* Printing the elements in dropdown:

for option in drop.options:
    print(option.text)

Top Ten
DATE
PAGE

* Working with links : links have tag 'a'

① links = driver.find_elements(By.TAG_NAME, "a")
print(len(links))

② for link in links:
print(link.text)

Clicking on the link:

① driver.find_element(By.LINK_TEXT, "Register").click()

② driver.find_element(By.PARTIAL_LINK_TEXT, "REG").click()

* Working with alerts : Pop-up / Alert window
(Confirm box, alert box, prompt box, hidden division, file download, child browser pop-up)

① time.sleep(5)
driver.switch_to.alert().accept()

② driver.switch_to.alert().dismiss()

③ obj = driver.switch_to.alert
message = obj.text → To print message

Top Ten
DATE
PAGE

* Working with frames/iframe:

① driver.switch_to.frame("Packageframe")
driver.find_element_by_link_text("link").click()
driver.switch_to.default_content() → to return to default content

* Working with Browser Windows:

→ driver.current_window_handle will return handle id of of current browser whereas driver.window_handles will return the handle ids of all browsers.

① print(driver.current_window_handle)

② handle = driver.window_handles

for handle in handles:
    driver.switch_to.window(handle)
    print(driver.title)
    if driver.title == "Pound Value":
        driver.close()

* Working with Web tables: table tag

```
rows = len(driver.find_elements_by_xpath("//table/tr"))
cols = len(driver.find_elements_by_xpath("//table/tr[1]/th"))
print(rows)
print(cols)
```

```
for r in range(2, rows+1):
    for c in range(1, cols+1):
        value = driver.find_element_by_xpath("
                "//html//table/tr["+str(r)+"]/
                td["+str(c)+"]").text
        print(value, end='  ')
    print()
```

* Scrolling

① Pixel approach

```
driver.execute_script("window.scrollBy(0,5000)", "")
```

② Element approach

```
flag = driver.find_element_by_xpath("//*[@id='exam']")
driver.execute_script("arguments[0].scrollIntoView();",
                      flag)
```

Top Ten
DATE
PAGE

③ till end:

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

* Mouse Actions:

① Mouse Hover:

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

electronics = driver.find_element_by_xpath("//@id")
actions = ActionChains(driver)
actions.move_to_element(electronics).click().perform()
```

② Double click():

```
copytext = driver.find_element_by_xpath("//@id")
actions = ActionChains(driver)
actions.double_click(copytext).click().perform()
```

③ actions.context_click(button).perform() → for right click

④ Drag and Drop :

```
source_element = driver.find_element_by_xpath("")
target_element = driver.find_element_by_xpath("")
actions.drag_and_drop(source_element, target_element).perform()
```

* Upload file :

```
driver.find_element_by_id("button").send_keys("C://A.../img.jpg")
```

* Download a file in chrome browser :

```
driver.find_element_by_id("button").click()
```

↳ "driver.get() is used to navigate to particular URL and wait till page loads. it doesn't maintain browser history and cookies whereas driver.navigate() does same thing but doesn't wait till page loads. it maintains browser history or cookies to navigate back or forward

Top Ten
DATE
PAGE

* Data driven testing using excel: library : openpyxl

Reading excel file:

```
import openpyxl
path = "C:\selenium\data.xlsx"
workbook = openpyxl.load_workbook(path)

sheet = workbook.active --> for active sheet
sheet = workbook.get_sheet_by_name("Sheet1")

rows = sheet.max_row
cols = sheet.max_column

for r in range(1, rows+1):
    for c in range(1, cols+1):
        print(sheet.cell(row=r, column=c).
              value, end = "  ")
    print()
```

DATE Top Ten
PAGE

* Writing to excel file:

```
for i in range(1,6):
    for c in range(1,4):
        sheet.cell(row=r, column=c).value = "welcome"
workbook.save(path)
```

*** We can input text in the text box without the method sendkeys with help of javascript executor. selenium executes javascript commands with help of the execute_script method.

* All pairs testing: testing all possible combinations (similar to cross-join) of parameters involved.

* Grey box testing: testing is carried out based on the partial knowledge of underlying design and implementation of system.

* Compliance testing: Meeting predefined standards

* Vulnerability testing: finding loopholes to get into system.
eg: Password should be encrypted.

Top Ten
DATE
PAGE

* Handling cookies:

→ Cookie is a piece of information from website and saved by your browser.
eg: It stores the login info. like username & password

→ It is stored in the form of key, value pair.

Capture all cookies created by browser:

```
cookies = driver.get_cookies()
print(len(cookies))
print(cookies)
```
→ It is stored in form of key-value pair

Adding new cookie to browser:

```
cookie = {'name': 'Mycookie', 'value': '12345678'}
driver.add_cookie(cookie)
```

Deleting the cookie:

① `driver.delete_cookie('MyCookie')`
② `driver.delete_all_cookies()`

Top Ten
DATE
PAGE

* Screenshot capture!

① driver.save_screenshot("C:\screenshot\page.png")
→ it can save/work with any extension
② driver.get_screenshot_as_file("C:\...\page.png")
→ it can save/work with png extension

* Logging: To discover scenarios which we might not have thought of while developing.

↳ log can have following levels: Debug, info, Warning, Error and Critical

→ info, warning can be ignored and usually debug, info messages won't be printed.

Printing to console:

```
import logging
logging.error("this is error message")
logging.critical("this is error message")
```

→ To print logs to file, we should mention the file path and to print even debug, info message, we should mention level.

Top Ten
DATE
PAGE

```
import logging

logging.basicConfig (filename = "C://---//test.log",
        format = '%(asctime)s : %(levelname)s :
         %(message)s',
         datefmt = '%m/%d/%Y %I:%M:%S %p',
         level = logging.DEBUG)

logging.info (" log msg")
logging.warning/error/critical("critical msg")
```

→ we can use logger object instead of using logging every time.

```
logger = logging.getLogger()
logger.setLevel(logging.DEBUG)
logger.debug (" log msg")
logger.error(" critical msg")
```

~~****~~ gettext() method simply returns visible text present betⁿ start & end tags (which is not hidden by CSS).
getattribute() method identifies & fetches key-value pairs of attributes within HTML tags.

↳ getvalue() returns the value held by formatted text field component while gettext() returns value's string representation.

DATE
PAGE
Top Ten

* Python Unittest framework:

→ We can use unittest testing framework to organize our automation code and to extract reports.

→ To use the methods present in the UnitTest framework we have to extend the testcase class present in the unittest module i.e. unittest.TestCase.

```
import unittest
from selenium import webdriver

class SearchEngineTest(unittest.TestCase):
    def test_Google(self):
        self.driver = webdriver.Chrome()
        self.driver.get("https://gogle.com")
        print("title is" + self.driver.title)
        self.driver.close()
    def test_Bing(self):
        self.driver = webdriver.Chrome()
        self.driver.get("https://bing.com")
        print("title is:" + self.driver.title)
        self.driver.close()
if __name__ == "__main__":
    unittest.main()
```

Top Ten
DATE
PAGE

* Keywords in Unittest framework:

→ setUp method will be executed before every method in the class whereas tearDown method will be executed after every method in class.

```
import unittest

class AppTest (unittest.Testcase):

    @classmethod
    def setUp(self):
        print("Login Test")

    @classmethod
    def tearDown(self):
        print("Logout Test")

    def test_search(self):
        print("Test 1")
    def test_advancesearch(self):
        print("Test 2")

if __name__ == "__main__":
    unittest.main()
```

→ setUpClass will be executed only once before all test methods are executed whereas tearDownClass will be executed only once after all test methods are executed

```
@classmethod
def setUpClass(cls):
    print("Open Application")

@classmethod
def tearDownClass(cls):
    print("closing Application")
```

→ setUpModule will be executed only once before execution any class or any method in test class whereas tearDownModule will be executed only once.

```
def setUpModule():
    print("Setting up the module")
```

* Skipping the test : to skip specific method while executing

①
```
@unittest.SkipTest
def test1(self):
    print("Skip 1")
```

Top Ten
DATE
PAGE

② @unittest.skip("Method is not ready")
def test2(self):
    print("Skip 2")

③ @unittest.skipIf(1==1, "Members are not equal")
def test3(self):
    print("skip 3")

* Assertion : it is nothing but checkpoint

① driver.get("https://google.com")
titleofpage = driver.title
assertEqual("Google", titleofpage, "titles are not same")
assertNotEqual("Google123", titleofpage)
assertTrue(titleofpage == "Google")
assertFalse(titleofpage == "Google")

② driver = webdriver.Chrome()
assertIsNone(driver)
driver = None
assertIsNotNone(driver)

→ assertIn verifies whether the first element is present in second element. it is used to verify presence of a value in a list, tuple, set and dictionary.

Top Ten
DATE
PAGE

① list = ["python", "ruby", "java"]
self.assertIn("python", list)
assertIn("rule", list)

② assertNotIn("Ram", list)

* Relational Comparison:

① assertLess(10, 100) → compares if first element is lesser than second element
② assertLessEqual(100, 100)
③ assertGreater(100, 10)
④ assertGreaterEqual(100, 100)

Top Ten
DATE
PAGE

* Pytest:

It is a library that is implemented on the top of unittest framework.

→ filename should start with test.

```
import pytest

def test1:
    print("Test Method 1")
def test2:
    print("Test Method 2")
```

→ To execute it in terminal (alt+F12), type pytest -v -s and enter and to execute specific module, type:
pytest -v -s test1.py

* Pytest fixtures:

→ They provide a fixed baseline upon which tests can reliably and repeatedly execute.

① @pytest.fixture() → executes specific method before every test method

② @pytest.yield_fixture() → executes specific method before & after every test method.

DATE
PAGE
Top Ten

① import pytest

```
@pytest.fixture()
def setup():
    print ("once before every method")

def test1(setup):
    print ("Test1 method")

def test2():
    print ("Test2 Method")
```

② import pytest

```
@pytest.yield_fixture()
def setup():
    print ("Before every method")
    yield
    print ("After every method")

def test1(setup):
    print ("Test1 Method")

def test2():
    print ("Test2 Method")
```

Pytest (Similar to TestNG)

↳ pytest framework makes it easy to write small tests, set scales to support complex functional testing for applications and libraries.

↳ pytest files should have the format test_first.py / first_test.py.

↳ test methods/functions should start with keyword 'test'.

eg:
```
def testLogin():
    print("Login Successful")
    assert 2+2 == 6
```

↳ to know about pytest commands in details, please use pytest -h for help

↳ Markers can be used to group test cases and run them.
pytest --markers is used to get details about markers.

eg:
```
@pytest.mark.skip/xfail/sanity/regression
def testLogin():
    print("Login Successful")
```

* Fixture : used for arrange & cleanup

↳ A test case can be broken into 4 steps:

① Arrange : Setup eg: launch browser & browsing product
② Act : Action eg: Add item to cart
③ Assert : Verification eg: verify item is added or not
④ cleanup : Reset the state of application to original state.

eg: @pytest.fixtuse()
def setUp():
print("Launch browser") } setUp
print("Login")
print("Browse Products")
yield
print("Logoff") } TearDown
print("Close Browser")

def testAddItemtoCast (setUp):
print("Add item successful")

* conftest.py : test configuration file have common methods used frequently in our test cases

eg: setUp() method will be moved to conftest.py file and scope is defined so that it can be used wherever required.

↳ the default scope of fixtuse will be in method. it could be class, module, package and session as defined.

* Parameterizing fixtures and test functions:

↳ it allows us to define multiple sets of arguments and those arguments can be taken iteratively and executed.

eg: @pytest.mark.parametrize("a, b, final", [(2,6,8), (5,8,15), (5,10,15)])
def testAdd(a,b,final):
assert a+b == final

Git & Github

↳ 'Git' is a free and open source distributed version control system designed to handle projects with speed & efficiency.

* Local version control system: Manually updating version

* Centralized version control: version control with central VCS server (Nobody can collaborate if server is down)

* Distributed version control: Git (Local computer)

↳ version will be available in server computer as well as in local computer which removes complete dependency on server computer.

↳ 'Github' is Git repository hosting service. it is a cloud service.
Github (Remote Repository) → similar to bitbucket.

* Basic workflow of Git & Github:

Working Directory —add→ Staging Area —commit→ Git Repository (Local Repository) —push↓→ Github (Remote Repository)

↳ git init → To initialize add procedure/repository

① git status → to check if any modification is done.

② git add demofile.py/ → to add file to local git staging area i.e. repository
git add */-A/.

③ git config --global user.name "Manish Verma"
git config --global user.email "vsk6895@gmail.com"
git config --global --list → configured username & password will be listed

④ git commit -m "first commit" → committing to push code to git repository from staging area

⑤ git branch -M master/main → to change branch to master/main

⑥ git remote add/origin https://github.com/manishverma/github-demo.git
set-url
→ to push to remote repository

⑦ git branch -M ~~bra~~main

⑧ git push -u origin main

* Creating personal access token :

↳ Settings → Developer settings → Personal access token → Generate new token → click on repo → Generate token

⑨ git reset → to reset all files in a repository i.e. remove all files from repository (staging area)
git reset --hard

⑩ touch .gitignore → to create a gitignore file

⑪ Vi

* H

1

2

3

4

*

↳

L

(11) vi .gitignore → to edit .gitignore file in order to add files which needs to be ignored i.e file not to be added into staging area.

↓

--INSERT--

↓

type filename to be ignored → Esc

↓

:wq

* How to clone Github repository to local repository:

(1) create a folder where you want to have all code in your computer

(2) git clone https://github.com/mverma/github-demo.git

(3) git remote set-url origin https://mverma@github.com/mverma/github-demo.git → if access is denied for username & password.

(4) git pull → to fetch latest changes from remote repository to local repository

* Importance of branching:

↳ Branch is created in order to work in isolation and avoid conflicts.

↳ git branch → to get list of branches

① git branch newbranch/
git checkout -b new_branch → create a new branch & switch to a branch

↓ git status

↓ git add .

↓ git commit -m "commit branch change"

↓ git push origin new_branch

↓ pull requests

↓ compare & pull request

↓ create pull request

↓ Merge pull request

↓ confirm merge

↳ git merge → for merging purpose

② git diff → changes to file not yet staged
git diff --cached/staged → show changes to staged file
git diff head → show all staged & unstaged file changes
git diff commit1 commit2 → difference bet^n commit1&2

③ git blame filename → list the change dates and authors for a file

④ git show [commit]: [file] → show file changes for a commit id / file

git log → show full change history

git log -p filename → show change history including differences

⑤ git branch → lists all local branches

git branch -av → lists all local & remote branches

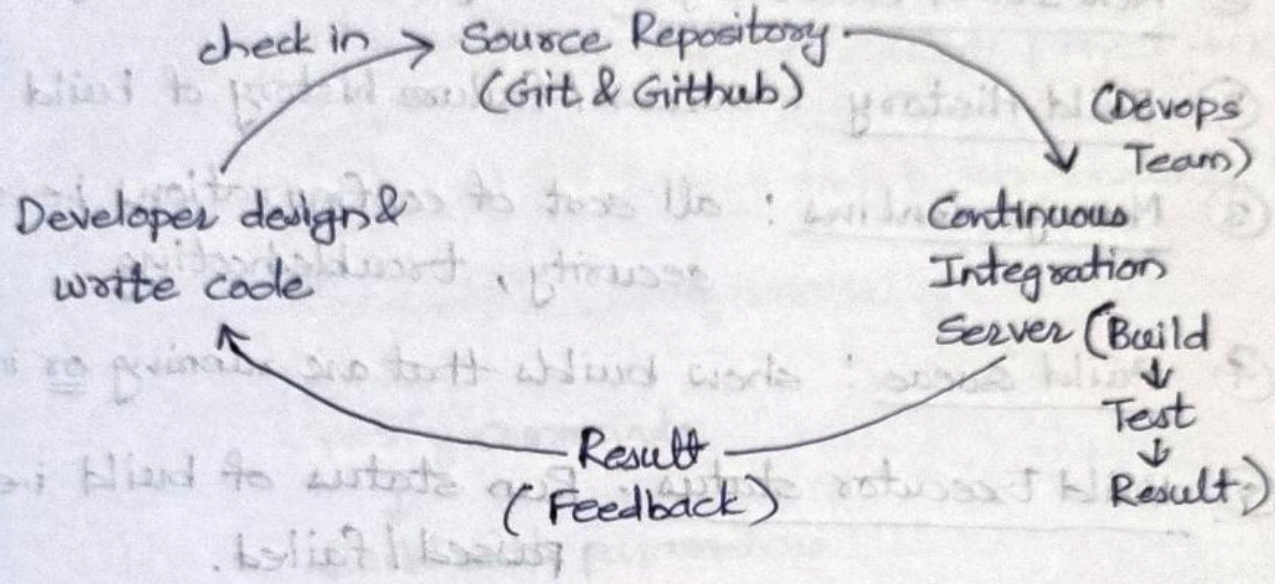
git checkout branchname → switch to branch and update working directory

git branch -d branchname → delete the branch

⑥ git fetch → get latest changes from origin

⑦ git restore → to restore the accidental changes done.

Jenkins



↳ Jenkins is a continuous integration (CI) server. it helps to automate the process of Build → Test → Result i.e. a series of commands.

* Installation of Jenkins: Must have compatible java version

① Download the latest stable Jenkins web application ARchive (WAR) file to an appropriate directory on your machine.

② Open up cmd window to download directory

③ Run the command:

```
java -jar jenkins.war --httpPort = 9090
```

④ Browse the server and wait until Unlock Jenkins page appears.

⑤ Continue on with post-installation setup wizard.

* Jenkins Dashboard overview:

① **New Item / Create a Job** : any process / activity performed

② **Build History** : success / failure history of build

③ **Manage Jenkins** : all sort of configurations i.e. system, security, troubleshooting

④ **Build Queue** : show builds that are running or in queue

⑤ **Build Executor status** : Run status of build i.e. passed / failed.

↳ **Jenkins is used to automate a series of commands:**

eg: cd c:\Jenkins-tutorial
SET directory = Jenkins1 → parameterizing
mkdir %directory%
dir > output.txt
echo %browser% → to print output

↳ **To provide values of parameters dynamically:**

Click on checkbox before this project is parameterized
↓
Add Parameter
↓
choice / string Parameter
↓
define parameters
↓
Save & Apply

* How to run automation tests in Jenkins: manually specifying project path (Custom workspace)

New item

↓

Enter an item name → Freestyle project → OK

↓

Build → Execute windows batch command

↓

Add parameters dynamically

↓

Specify commands

↓

Build with parameters

* How to Run test cases from Github using Jenkins:

New item

↓

Enter an item name → Freestyle project → OK

↓

Source Code Management → Git

↓

Enter github project URL

↓

Add credentials → Choose branch (Main/ Master)

↓

Add parameters dynamically

↓

Build → Execute windows batch command

↓

eg: pytest --browser %.browser --url %.url% → apply & save

↓

Build with parameter