Jenkins

Lecture -101

Alll these changes are made in **configure**

1:05---copy complete path from pycharm--to---------- general --”use custom workspace”--directory ---add path

1:15-- observe other options---- **source code management** (git or none)

1:45--Built --- execute window batch command----current path will be --code path(pycharm file)

3:30--- exact required path

**5:40-**--- where you want the **report to get created**-**-----note**---- create an empty package with name reports

cd tests

py.test --browser\_name chrome --html=%**WORKSPACE%/**reports/reports.html

6:10---- see list of avaiable **envt variable**----%**WORKSPACE%**---automatically pass the absolute path of the directory assigned to the build as a workspace

6:50--Avoid **hard coding** is avoided--------call envt variable ---%**WORKSPACE%**

**Benefit** of running tests with **jenkins---** even **non technical** person can do it

Lecture-102

Browser name was **hard coded**--

1:02--- General-- this project is parameterized--**-choice parameter**

1. Name-- browserName
2. Choice-- chrome

-firefox

-IE

1. Description-empty

1:49--new option comes--- Built with parameter--it asks browser name

2:35

**But how** this command will come and **replace browser** in execute window batch command

---browser name replaced by **”%browserName%”**

**As given below**

cd tests

py.test --browser\_name **”%browserName%”**  --html=%**WORKSPACE%/**reports/reports.html

**-----quotes**  are used to show that it comes from the **choice parameter**

-

5:25---- avoid **hard coding of URL**----

**5:25----Using addoption**

**6:27--**add 1 more **choice parameter----URL**

**Lecture-103**

**JUnitxml report**

**1:58--- in** execute window batch command

**cd tests----------------------means** below reports generated in tests folder

cd tests

py.test --browser\_name **”%browserName%”**  --html=%**WORKSPACE%/**reports/reports.html

**2:50 --Post-build Actions**--**-** publish **JUnit** test result report

tests/\*.xml

--it will automatically create a **plug in**

**5:30--jUnitxml report----in------ Latest Test Result**

**---graph** will be seen if we run it 4-5 times

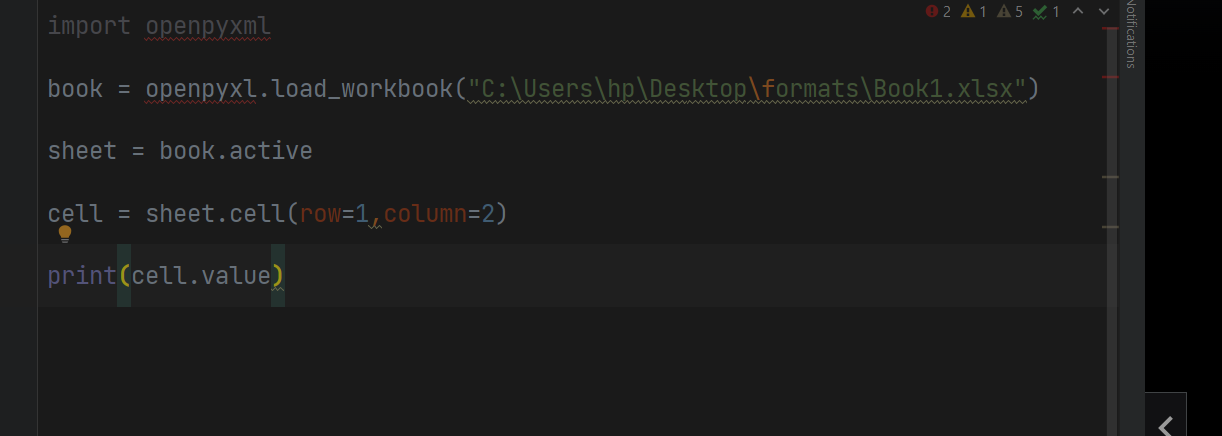
6:08--**Original report**---- in workspace folder-- report

Lecture--104

1:15------ openpyxl------ open cmd

Pip install openpyxl

Or install in pycharm interpreter



C:\Users\hp\PycharmProjects\pythonSelFramework\pageObject\data driven testing.py

Lecture-106

--6:50----- right click--evaluate expression ---- to know value

Lecture- 107

5:25-- see how to debug

---select the expression --right click -- evaluate expression--- evaluate --it will give us result

Lecture -108

Integrate it into our framework

-- in new folder --TestData --new file----HomePageData.py

Selenium features

-exclusively for web Based application

1demo—includes

Inheritance

Read\_part1

Read\_and\_write\_file

try and except

exception

webdriver

registration

With ----- has **:**

From \_\_\_\_\_\_\_\_\_\_import\_\_\_\_\_\_\_----has **no** :

Lecture -20

**OOPS**

-creating class and object with same name

**class** Calculator:

num =100

def getData(self):

print("I am a good boy")

**obj** = Calculator()

obj.getData()

print(obj.num)

--obj is created outside the class

**Lect- 21**

Constructor

Constructor name is always **init**

**Uses of default constructor**

* To create instance variable

**class** Calculator:

num =100

#default constructor

Def \_init\_(self):

def getData(self):

print("I am a good boy")

**obj** = Calculator()

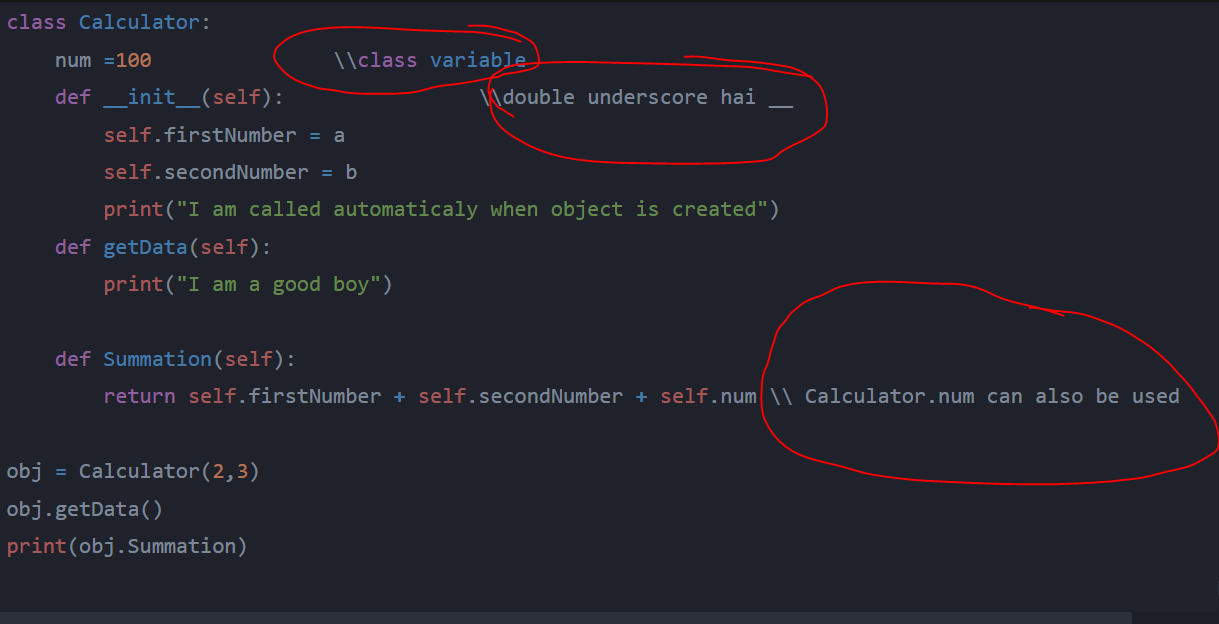
obj.getData()

print(obj.num)

10:30 – instance variables keep on changing

Class variable remains the same

13:00 --- self can be used to call global variable as well



def \_\_init\_\_(self,a,b):

when def is inside the class--- it is called method

15:47

Self keyword is mandatory for calling variable names into method

#instance and class variables have whole different purpose

Constructor name should be \_\_init\_\_

Lecture –22

Inheritance

Copying characteristics from parent to child class

11:45 ---see how it will run

from demo import Calculator  
class childImpl(Calculator):  
 num2 =200  
  
 def \_\_init\_\_(self):  
 Calculator.\_\_init\_\_(self,2,10)  
 def getCompleteData(self):  
 return self.num2 +self.num +self.summation()  
obj = childImpl()  
print(obj.getCompleteData())

Lecture –25 -----

Read text file content

file = open(‘test.txt’)

print(file.read())

file.close()

file = open(‘test.txt’)

print(file.read(2))---------------------------**means** it reads 2 bits of data

file.close()

Output---------------- it reads only 1st 2 characters

**Read content line wise**

file = open(‘test.txt’)

print(file.read(5))

print(file.readline())

print(file.readline())

file.close()

**output**---file.read(5)--- means it will read 1st 5 letters

2\* file.readline------------reads next 2 lines----------------6:15

Avoiding mixing read(2)and readlines

**#** in pycharm ----makes it a comment

Lecture- 26

Reading all the content line by line

file = open(‘test.txt’)

line = file.readline()

while line!= “”:

print(line)

line =file.readline() ----if we don’t run this line then only 1st line is run indefinitely

file.close()

using for loop

for line in file.readlines():

print(line)

Lecture -27

Writing data into a file

#read the file and store all the lines in list

#write the list back to the file

with open("test.txt","r") as reader:  
 content = reader.readlines()  
 reversed(content)  
with open("text.txt","w") as writer:  
 for line in reversed(content):  
 writer.write(line)

**lecture -28**

raise an exception and assertion **are 2 methods** to fail the test

**Raise an exception**

ItemsInCart = 0

If ItemsInCart != 2:

raise Exception(“products cart count not matching”)

**assertion method**

ItemsInCart = 0

If ItemsInCart != 2: #Raise Exception(“products cart count not matching”)

Pass

assert(ItemsInCart == 2)

#output ---assertion failed

ItemsInCart = 0

If ItemsInCart != 2: #Raise Exception(“products cart count not matching”)

Pass

assert(ItemsInCart == 0)

5:30----output -------------Process finished with exit code 0--means

**Lectur -29**

Try **and except** block

Trying to open an non existant file

try:

with open(‘’filelog.txt’,’r’) as reader:

reader.read()

except:

print(”some how I reached here”)

**output** ---test will not fail

**note---** but it doesn’t let you know what exception message python is throwing

try:

with open(‘’filelog.txt’,’r’) as reader:

reader.read()

except Exception as e:

print(e)

output – [Error 2] No such file or directory: ‘filelog.txt’ ??????

Lecture -30

try:

with open(‘’filelog.txt’,’r’) as reader:

reader.read()

except Exception as e:

print(e)

finally:

print(“cleaning up resources”)

**note---finally** printed irrespective of existance of error or not

purpose of **finally**

-cleaning the data used for above tests

---avoids mess

lecture-32

invoke the chrome browser

2:20 ---webdriver.Chrome – likhte hi option aa jayega –import karne ka

4:40 – driver is used because directly browser can’t be invoked

8:40 --- these 3 lines code you will have to write everytime to open the browser

9:15—driver.close()

11:00—m—method

p---property --brackets **don’t** come

driver.title --- #gets the title

print(**driver.title**)

12:16—make sure that you landed on the correct URL

Print(Driver.current\_url)

Lecture-33

Driver has full control of the chrome browser that we invoked

driver.maximize\_window()

driver.**back()----**to go back in browser

driver.**refresh()**

driver.**forward()—**3:25

--how to keep the windows open

from selenium import webdriver  
  
from selenium.webdriver.chrome.options import Options  
options = Options()  
options.add\_experimental\_option("detach",True)  
driver = webdriver.Chrome(options=options)  
driver.maximize\_window()  
driver.get("https://youtube.com/")

Lecture -34 **for firefox**

1:20 –gecko driver (google it)---------for firefox

2:19 – change the path

&

webdriver.Firefox---------------rest remains the same

for Microsoft edge

webdriver.Edge

download Microsoft edge driver

Lecture -35

Inspecting HTML to identify attributes of element

<https://rahulshettyacademy.com/angularpractice/>

-how selenium will know that we have to put email\_id in this edit box

2:49—open inspect –element—

Driver supports class , name ,required type

4:10 ---new package gets installed

Driver.find\_element(By.)

**Note**—bulb symbol helps a lot

4:28--- it tells by how many ways it can identify the elements

driver.find\_element(By.Name,”email”).send\_keys(“ankit@gmail.com”)

5:44—for password edit box—same thing is done—inspect

6:17

To avoid confusion –take an attribute that is unique

7:30

Xpath , CSSSelector can be created for any element of the page

8:51—for check box

driver.find\_element(By.ID,”exampleCheck1”).click()

Lecture -36

**For XPATH**

1:28 ---tag name means input

2:08—syntax

-

3:25—class=”alert alert=success alert\_dismissible”-------------means 3 class

Space -----**means** another class

--anyone of the following can be used to access it

4:50-- .text------------- to grab the text

**For CSSSelector**

**--**we tried it for **name** field (although it could be done by ----By.NAME)

7:17 – syntax given for CSSSelector

7:38 –easy conversion from Xpath to CSSSelector

driver.find\_element(By.CSS\_SELECTOR, “input[name=’name’]”).send\_keys(“Rahul”)

driver.find\_element(By.XPATH,”//input[@type=’submit’]”).click()

message = driver.find\_element(By.CLASS\_NAME,"alert-success").text

----Note --.text ----**no ()** **!** , .text-- because we wanted to read and print the message

**Note** ---focus on the fact that why input are different in the 2 cases and why .click method is used

**Write a test**

**9:02**

**Assert** ”Success” in message !

10:14---assertion failed

**Lecture -37**

1:16-----------**Plug in**------selectorsHub------to validate the Xpath and CSS you wrote are correct or not

1:45—**example**--- to find what is unique for that path

2:59—using ID CSS can be created

#id--------- (example #inlineRadio 1)-can also be used as CSS

CSS can be coverted by 2 other methods

1. 3:10—using class also CSS can be created

------see how we come to know that success message is a class name

1. 3:38--------------.classname

**Using index** while writing Xpath ---- when more than 1 element is matching

5:25—3 elements were matching

5:50—[3]---------third one is matching--1 element is matching

---6:25-----(//input[@type='text']**)**[3]----- **Note**-- 3 is outside the round brackets!

As 3 text boxes were there and we wanted the third one

7:18—two way binding----

7:39 ---- .clear()-------to clear **that particula**r edit box!

Lecture -38

**change password**

**Tranversing from parent to child!**

driver.find\_element(By.LINK\_TEXT,"Forgot password?").click() !  
driver.find\_element(By.XPATH,"//form/div[1]/input").send\_keys('ankitkharb931@gmail.com')  
driver.find\_element(By.XPATH,"//form/div[2]/input").send\_keys('nayapassword')  
driver.find\_element(By.XPATH,"//form/div[3]/input").send\_keys('nayapassword')  
#driver.find\_element(By.XPATH,"//form/div[4]/button").click()  
driver.find\_element(By.XPATH,"//button[text()='Save New Password']").click() !

-using **linktext**  for forget password

**<a------------** suggests linktext

2:10—how to identify linktext –very important

----text present on that link

--partial link text can also be used if complete text is not available

4:40 -- **//input** can have multiple results so we narrow down by providing more info

5:00----**//form**---whole form is selected--- now we will go from form to to specific text box where email locator is present

//form/div[1]/input-------------------**traversing** from parent to child

--form was the parent –div was the child –we wanted 1st one

5:28-how to traverse from parent to child—form is a parent and different <div are the child items

5:59—tells which div to access

//form/div/[1]/input---------------------for 1st one **and not**[0] **!**

--this method is used when no unique attribute can be found—6:45

8:10 – how to know where to locate

**Note** -- it is better to learn parent child traversing -- either of them

9:30---- in CSS

By.CSS\_selector ----Form div:nth-child(2)input

9:50 ---.send\_keys()--- is used because input is required

10:30 – I am more comfortable with **CSS locator** –because it is **fast**

12:40----**Xpath** based upon text !

//button[text()=’save new password’]

Note –this facility is not available for CSS

driver.find\_element(By.XPATH,"//button[text()='Save New Password']").click()

Lecture- 40

**( Registration )**

In this lecture we will be required to import something--- where , I don’t know the exact location

**Static dropdown**

----options are fixed

To implement it

--**Select()**  ---------is used !

Define an variable ----dropdown()

3:14---- various options available

(**dropdown**.--- likhte hi aa jayega)

Example

dropdown.select\_by\_index(0) **!**

**or**

dropdown.select\_by\_visible\_text(“female”)  **!**

or

dropdown>select\_by\_value------------------if option available –5:35

6:40

---- suppose we need to test 3 options in a dropdown

====select

**Note**----if **<select**—is given for any dropdown –means it is **static**

dropdown.select\_by\_index(0)------means item mentioned in the dropdown at index 0

dropdown.select\_by\_index(1)

dropdown.select\_by\_index(2)

Lecture -41

**Autosuggestive dropdowns**

-

Practice page-----Flightbooking page

driver.find\_element(By\_ID,”autosuggest”).send\_keys(“ind”)!-------**-not** element**s**

**time.sleep(2)**----------to take 2 seconds pause before running the next statement

**for loop** will be required as position of “india” is not fixed in the dropdown box

---in the drop down –right click on india—**3:30**

**5:30---li** is the class name

Li[class=’ui-menu-item’] a----------------gets you **access to 3 items!**

Countries = driver.find\_element**s(By.CSS\_SLECTOR, “**Li[class=’ui-menu-item’] a”)

len(countries)---------------------return list of elements

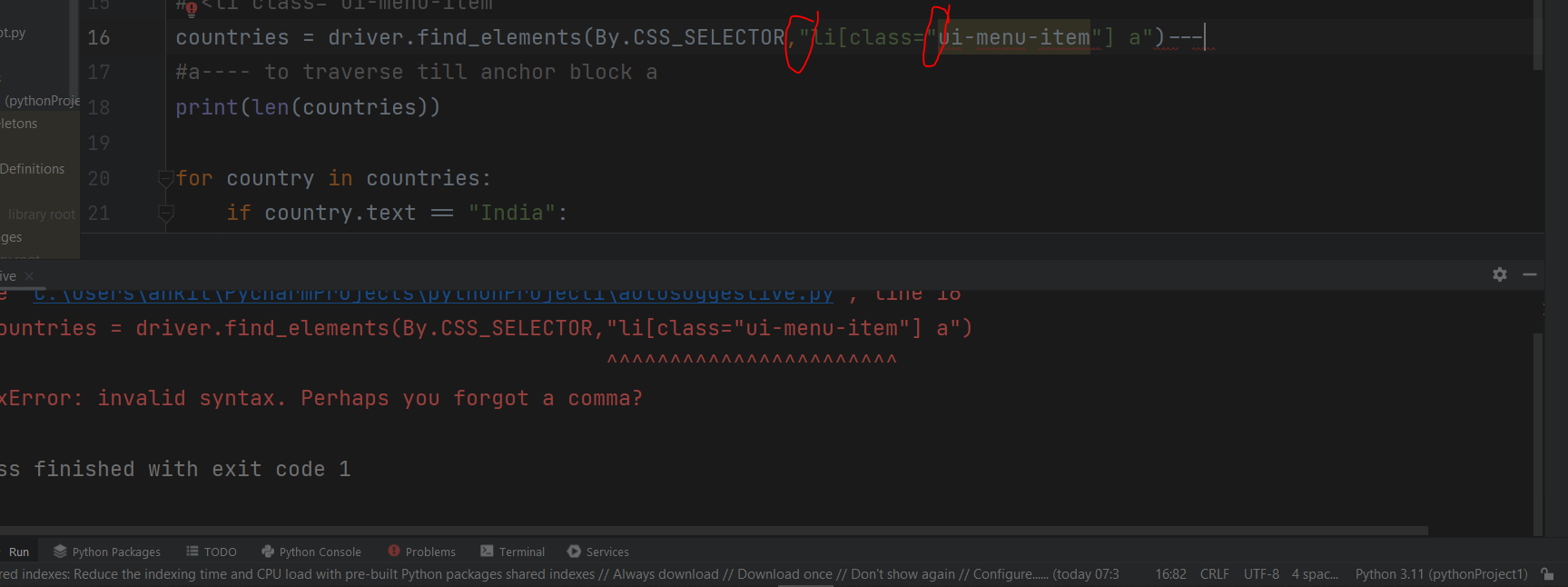
**For loop!**

for country in countries:

if country**.text** ==’India’:

country.click()---------------------click will be made

**break**---------------------when condition is executed



--same type of “ ” inverted coma should not be there.

**Lecture—42**

1:10 ---- to check what all is inside the for loop

3:00 **----.text()** method **can’t** be used initially for auto suggestive dropdowns as that text will not be initially present on that page.

So

Print(driver.find\_element(By.ID, “autosuggest”).text)------------will **not work**

**5:15—so**  -------- .get\_attribute()---------------will be used

**Write a test**

Assert driver.find\_element(By.ID, “autosuggest”). .get\_attribute(‘’value) == ‘India’

**Lecture- 43**

**Dynamic checkbox**

0:55--- to know which page to look for

**--Checkbox** location could be **variable**

//input[@type=’checkbox’]-------------will **not** work as 3 checkboxed are there

--although directly by choosing ID –below operation could have been done

But we are just showing **another option**

Checkboxes = driver.find\_elements(By.XPATH, “//input[@type=’checkbox’]”)

Print(len(checkboxes))------**to check** if we are going correctly or not

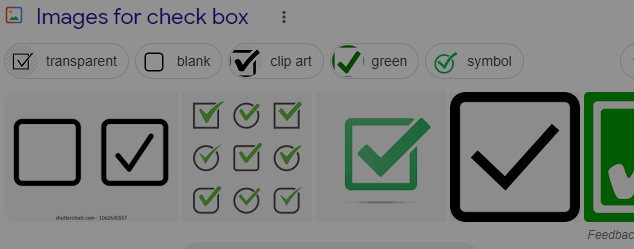
For checkbox in checkboxes:

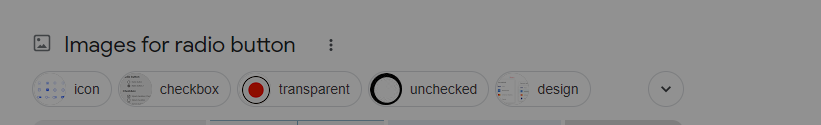
If Checkbox.get\_attribute(‘value’) ==’option2’: **!** ------Checkbox.text bhi chal jaata

Checkbox.click()

8:00--**assert** Checkbox.is\_selected() !---------------(return true if selected ------return type --boolean)

Break





Lecture—44

radiobuttons = driver.find\_elements(By.CSS\_SELECTOR,".radiobutton") !

**radiobuttons[2].click() !** -----if we know that position of radio button will not change

assert radiobuttons[2]**.**is\_selected() **!**

3:40----------------example Of hiding/showing the elements

**4:20**------**assert** Driver.find\_by\_element(By.ID,”displayed-text”).is.displayed()—-------------**assert type** is bollean

Driver.find\_by\_element(By.ID,”hide-textbook”).click()

assert Driver.find\_by\_element(By.ID,”displayed-text”).is.displayed()--------false

**assert not**---too exists

lecture—45

--**0:20**=------------- how a java pop up looks like

Handling java pop up using selenium

1st write a code to pop up a java message

**4:15**--- alert = driver.switch\_to.alert **!**

alertText = alert.text--------------**no** round brackets!

assert name in alertText--------------------means name comes in alertText!

alert.accept() **---to click on okay button**

**alert.dismiss() -------to click on negative button**

Process finished with exit code 0----**-means** assertion passed

-print ho jayega--- means we will get to know that the required will get pop up when conditions are triggered

Lecture -47

Greenkart page

3:00

Couponcode—rahulshettyacademy

**6:28**-- how to identify given 3 items

//div[@class="products"]---1 element match

//div[@class="products"]/div--- all the elements matched

--- use find element**s ------6:45**

Lecture-48

**Chaining** of web elements

Chaining of parent web element to child web element to construct dynamically

2:20

Instead of driver.find\_element

**We used**

**Result.find\_element**------------------as result is the parent web element

Results = driver.find\_element**s()**

for result in results :

result.find\_element().click() !

--to r**educe the scop**e of search we used result.find\_element instead of driver.find\_element

--3 result the toh isiley 3 baar loop chalega

4:50—see it clicked on all the 3 elements

5:40----click on the cart button

Lectur -49

Implicit wait

**Driver.implicit\_wait(5) !**

-1:45- sabse uppar kar dena –**means** for global

--means **maximum time out** will be 5 seconds – agar kam mein kaam chal gaya toh kam mein ho jayega

4:10 --- time.sleep(2)----**-not** removed for driver.find\_elements(means before this statement)

--because it will **not** work for that!

Lecture- 50

Wait = WebDriverWait(driver,10)  
wait.untill(expected\_conditions.presence\_of\_element\_located(By.CSS\_SELECTOR,"promoInfo"))

**!**

**Explicit wait**

--suppose a **particular scenario** – requires waiting period of 15 seconds

But if we increase waiting period every where – unneccary wastage of time for other case because we know that if items not displayed in 5 seconds –there is a bug

4:00---2nd last line

Wait = WebDriverWait(driver,10)

5:20 – .promoinfo----0 elements matching

----wait until promocode is applied & “code applied ” appears

wait.untill(expected\_conditions.presence\_of\_element\_located(By.CSS\_SELECTOR,"promoInfo"))



--1 more round bracket needed

-generally time.sleep(2)----------works

Lecture – 52

Functional Automation

0:30 ---find a locator to sum all the 3 elements in **total column**

**2:55 – CSS** while tranversing from parent to child !

**Note---**it is important to give heading using #

**Sum validation**

**9:20**---- note int is used to convert string to integer

Sum is equated with totalAmount

Sum = 0!

For price in prices:

Sum = sum + int(price.text)

Print(sum)

Totalamount = **int**(driver.find\_element())

Assert sum == totalamount

Lecture -54

Implementing

-create an empty list ---actualList

-append elements in it from from text

Assert expectedList == actualist

-- **note** -- int can’t accept decimal value

Actuallist.append(result.find\_element().text)

Lecture -56

Note--- .perform is required for all actions given below

.click\_and\_hold()---------------kind of a long press

.context\_click()---- means right\_click()

.double\_click()

-drag\_and\_drop

5:10----.move\_to\_element(driver.find\_element()).

------it will just move to that element (means hover over tha place)

Lecture -57

childWindow

<https://the-internet.herokuapp.com/windows>

5:50 --- switch to new window

driver.window\_handles---to grab all the windows that are opened

Provide us in the form of a list

driver.switch\_to.window(windowsOpened[1]) !

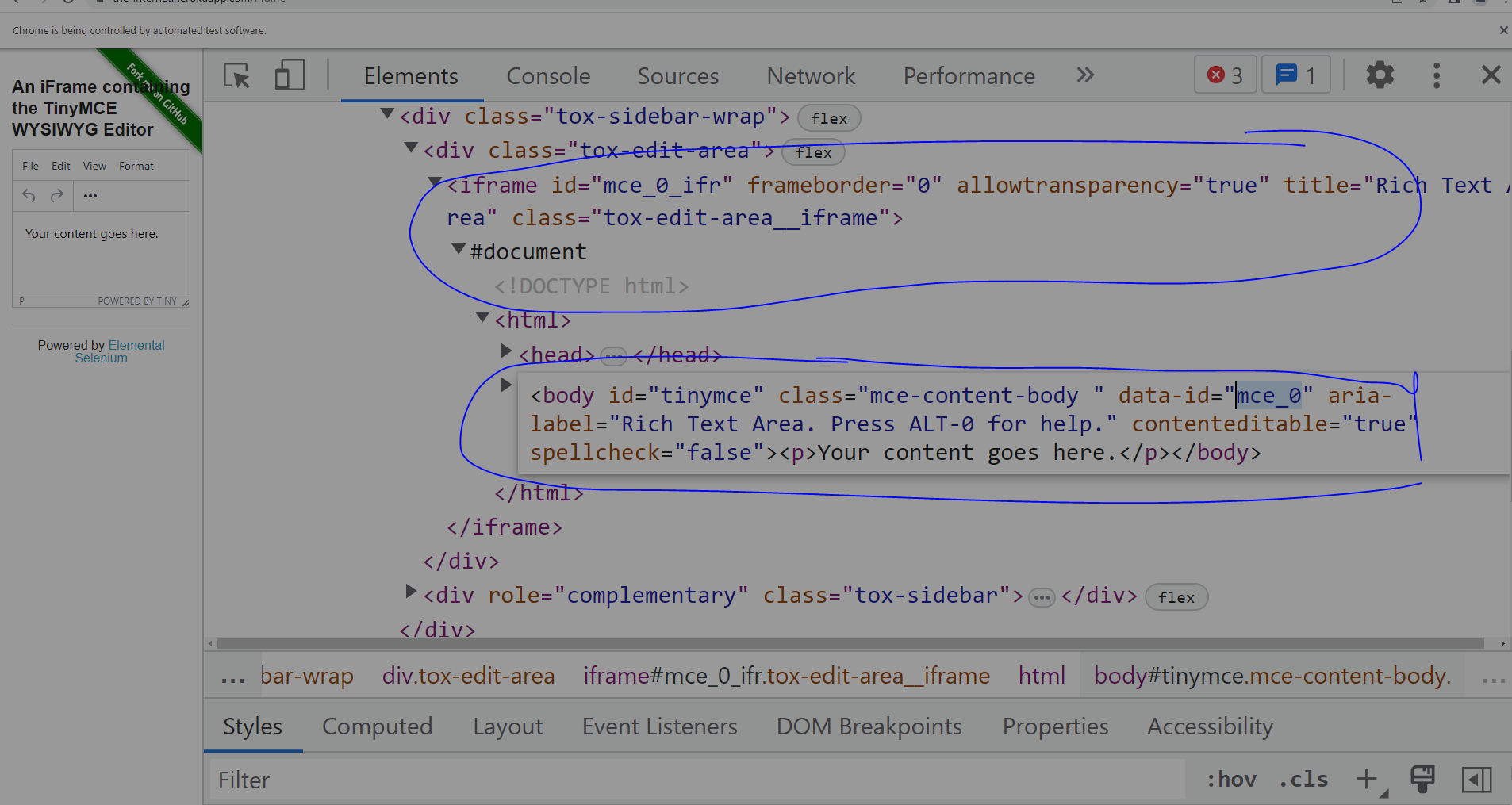
Lecture- 58

How to identify that frames were used

1. Ask the developer --if frames were used
2. Normally do your automation and if it fails --means -- frames were used

5:00-- error shown if we **treated frames as normal** items

“no such element error”



Note both were pointing towards the same place

-- both were not shown in chrome unless automation window opened

-7:20---- h3 as unique tag

9:25

-switch to default content

10:24 -- for more practice of frame

driver.switch\_to.frame("mce\_0\_ifr")

driver.switch\_to.default\_content()

Lecture -59

**Miscellaneous**

JavaScript executor

2:30-- see how we are typing in the console -- to scroll

**Write Window.scroll in console ---==**

3:55--- to scroll to the bottom

Window.scrollBy(0,document.body.scrollHeight) !

Driver.execute\_script(window.scrollBy(0,document.body.scrollHeight) )**!**

**Execute script** is used for executing javascript

driver.get\_screenshot\_as\_file("screen.png") !

8:00

head mode--- browser invokation can be seen

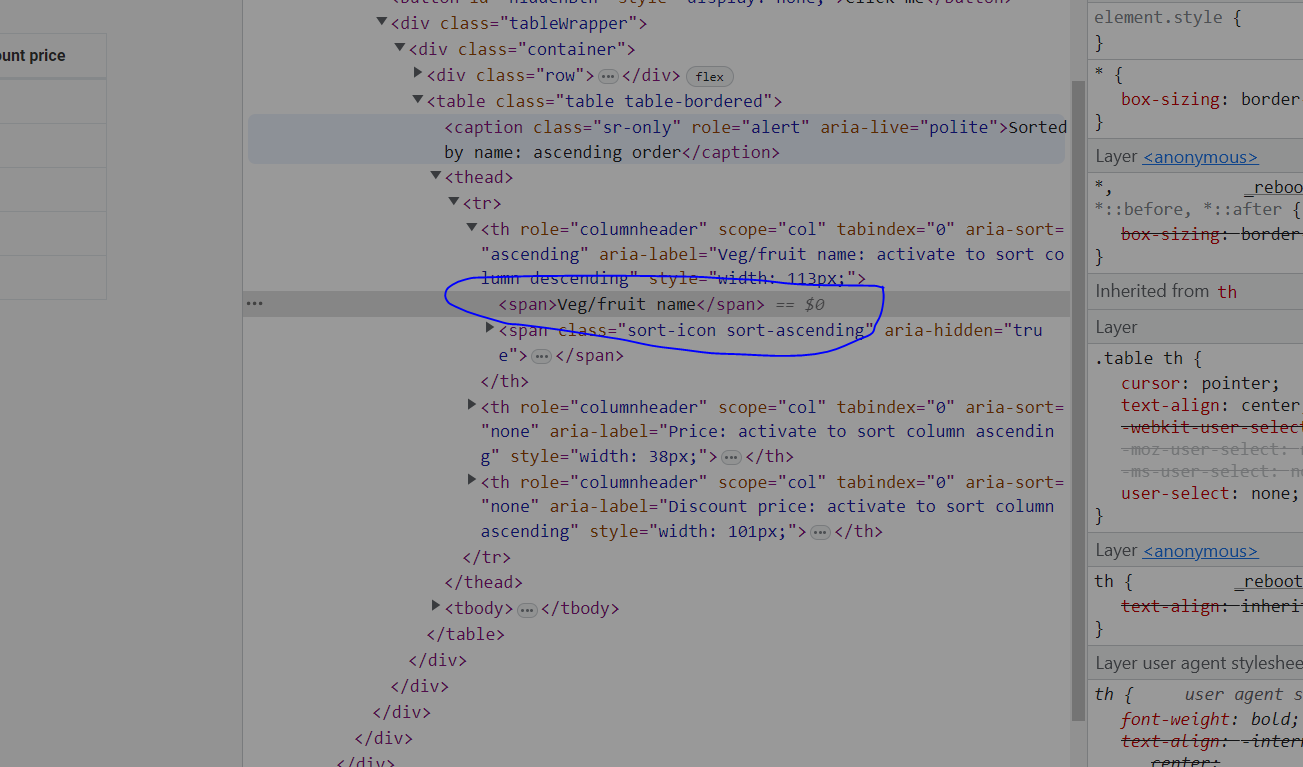
headless mode ---un necessary works are avoided

**11:30** -- connection is private error---also called **ignore certification error**

chrome\_options.add\_argument("--ignore-certificate-errors") !

Lecture-60

**Sorting the table**

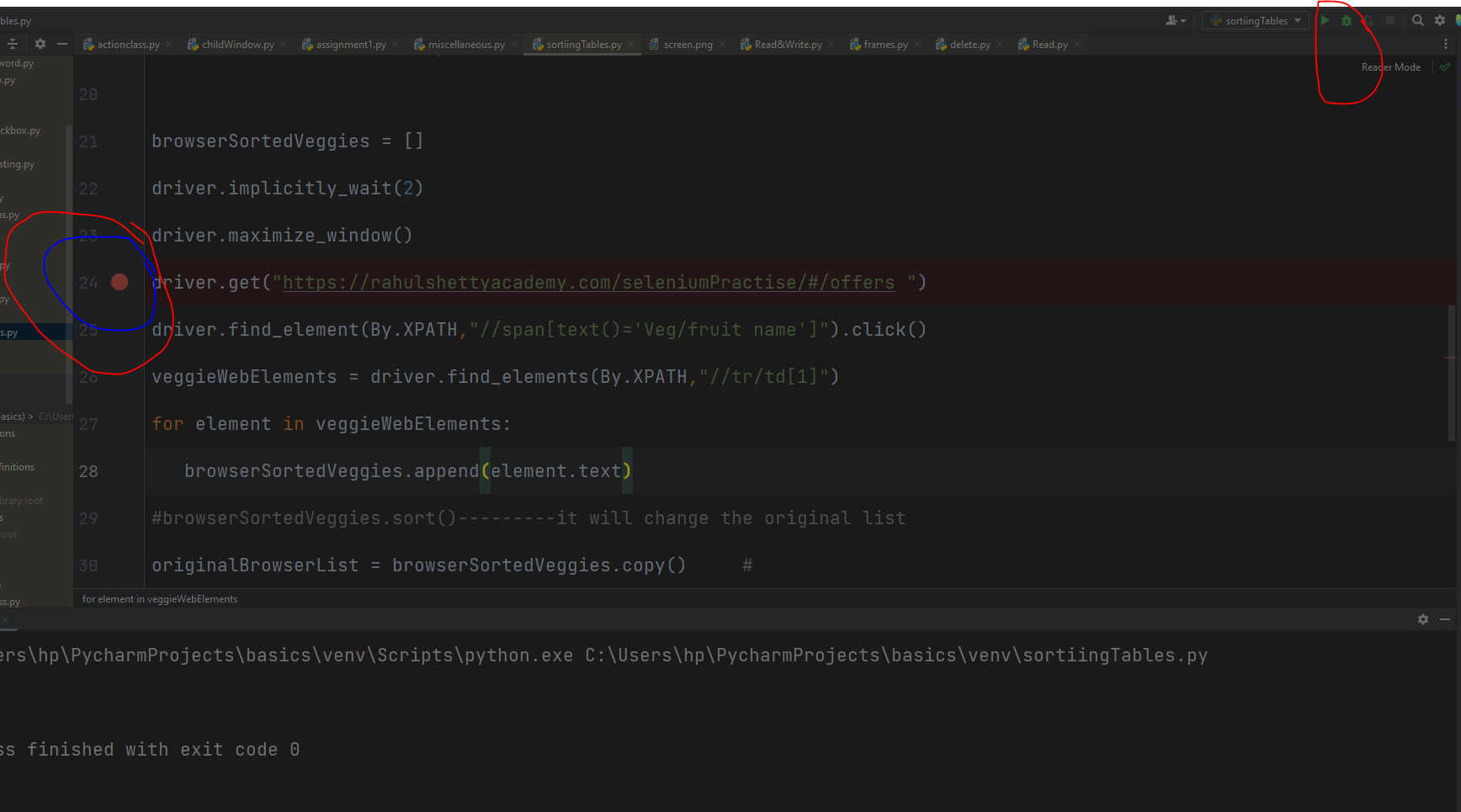


See how we handled it

originalBrowserList = browserSortedVeggies.copy() !

7:20-- in every tr , we want the 1st td

15:00---see how to **debug**



-so that script stops at that point

15:55-- stepover-- to move one step at a time

Lecture -61

Read a little bit more about chrome options

**Section -16 end to end practice (unchecked)**

**2:25 ----** using href attribute

Lecture- 63

1:55-- see how it reached --iphone

5:00 --see button tag is only present in the 2nd div

Lectur-64

0:50 ---selectHub plugin might not be there

So come to console -- for xpath do

$x(provide xpath )---------------------check if it is valid or not

Section -18

Part1 Pytest -**Unit Testing Framework**

Lecture -68

4:20 --- always start file name with **test\_**

8:00 -- important--- how to initialise the pytest

8:25---again important --- pass the path in pytest

--w**atch it many times**

Lecture--69

0:10 --- cmd use for running the tests

Enter the path-- by right clicking in pycharm

-- enter **py.test**

**py.test -v**  (v stands for verbose)

-s for **logs**

6:00 --- assertion

6:55--- test various test cases in different file together

8:30

def test\_firstProgram():  
 print("Hello")  
  
  
def test\_firstProgram():-----oberve colon!  
 print("Good morning")

-- 2nd method will overwrite the 1st one

Lecture -71

**2:20**  Run the **selected file** only in the cmd --- **give the** complete file name

py.test **file\_name** -v -s

**Note** -- you should be on the directory where the file is present

----meaning of -v -s ???

4:45-- name should be meaningful

6:00-- running only those testcases that have creditcard name

-k creditcard

py.test **-k CreditCard** -v -s

# any pytest file should inititate it's name with test\_  
#pytest method names should start with test  
# every code should be wrapped in the method  
# every method is treated as 1 test case  
# method name should make sense  
# -k stands for method name execution , -s logsin output , -v stands for metadata  
# you can run specific file with py.test <filename>

Lecture -72

Grouping tests

1:25 -- import pytest

2:00-- @pytest.mark .smoke -----here smoke is the name of the group

2:25 -- use keyword **-m** ----for (mark)------ in cmd

py.test **-m smoke** -v -s

**Note** -- mark used for indetification here is smoke

4:20 --- always use -k **unless** you want to run for smoke or regression testing for them use -m

6:00 --- skip a test [-----------@pytest.](mailto:-----------@pytest.mark.skip)**[mark.skip](mailto:-----------@pytest.mark.skip)**

**--------------------above** that particular test

8:45--- @pytest.**mark.xfail**---- test will **run --- but** it’s result will **not** be shown in the report

Lecture -73

**Fixtures**

**Uses** ----- pre request

Used as setup and tear down(**yield**) method for test cases

**3:00**

Can be **declared anywhere** -- **beginning**  or in the end

@pytest.fixture()------------------**no** colon

3:30 --

def test\_fixtureDemo(**setup**): ----- mentioning setup in it links this method with the fixture

-- so now setup method will be **executed 1st**

5:55 -- made it in a new filepy

7:25-- **yield**---

In this example see the **order of execution**

1. I will be executed first
2. I will execute steps in the fixtureDemo method
3. I will executed last

**Lecture-74**

**Conftest**

**Note** --- import pytest in conftest file

**Avoiding duplicating** fixture code for every file

1:55 -- create a file with **name conftest** -- and save the fixture file in it

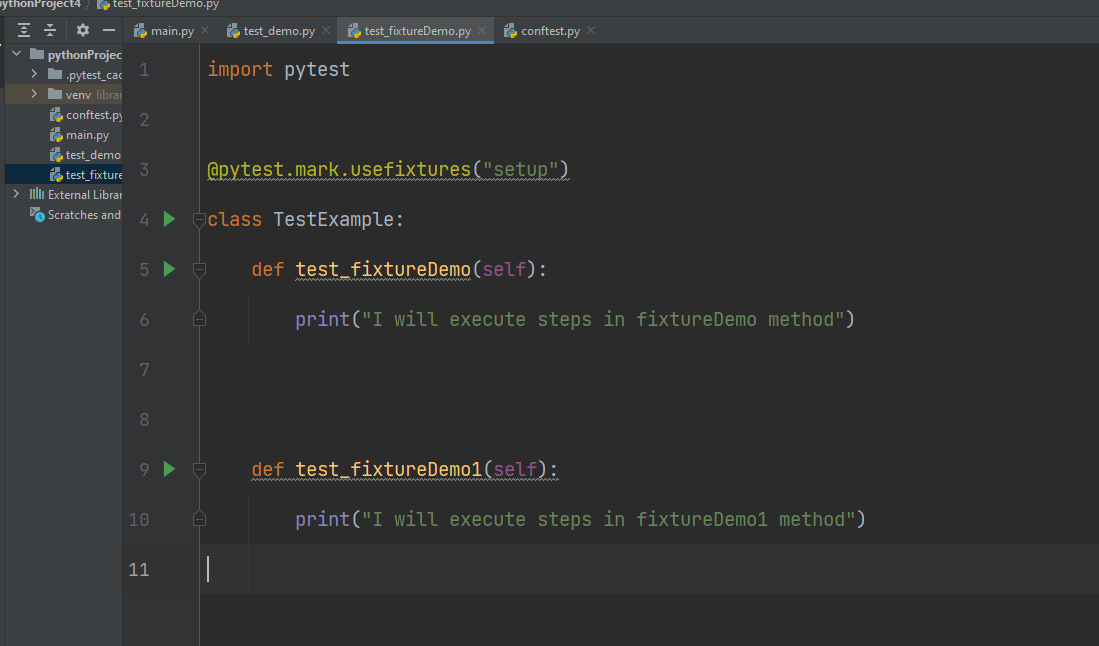
3:20 -- full **mechanism explained**

----if fixture is mentioned -- 1st fixture will be checked in the local file if not present --- then it will check for the conftest file

8:15 -- **wrapping** methods under a class so that we can **avoid** mentioning fixture in each of them

9:15-- thus (**self**)--- will be used

9:50---- fixture will be **mentioned** just above the class



11:50 --in conftest file

@pytest.fixture(**scope = “class”)**

**If** you only want to execute it once

LEcture- 75- **unchecked**

**Using fixtures to load the data in the test**

4:35 -- create a new fixture file---**-dataLoad()**---------in conftest

6:05-- 1 more python file in created---------to **write test**

@pytest.mark.usefixtures(“dataLoad”)

Class TestExample2:

Def test\_editProfile(self,**dataLoad**):----------- Fixture **name is mentioned** because we want to **return the data from fixture**

Print(dataLoad)

---------7:20--- explains the above process

9:10

Print the return data **individually**

Def test\_editProfile(self,**dataLoad**):-

Print(dataLoad[0])

Print(dataLoad[1])

Lecture -76

3:20 --- see how **fixture** **invokes browsers** chrome , firefox ,IE

In request.params------ various browsers are the parameters

These changes are made **in the conftest**

@pytest.fixture(params=[“chrome”,”Firefox”,”IE”])

def crossBrowser(**request**):

return request.param------------------**not** paramas

3:20 ---- **request** is used when fixtures have some values there

4:10 -- writing a test

----- def test\_crossBrowser(crossBrowser):

print(crossBrowser)

- self parameter is **not** required-----as we are not the method into a class

7:35

----- if different things required to be sent in different instance---**use square brackets**

@pytest.fixture(params=[(“chrome”,”Rahul”),”Firefox”,”IE”])

-- 8:20 -- whatever belongs to **1st round bracket-**-- it belongs to **1st run only**

9:10 -- in the 1st run it calls 3 values , in 2nd run 2 values , in th 3rd run 1 value only

@pytest.fixture(params=[(“chrome”,”1”),(”Firefox”,”2”),(”IE”,”3”)])

9:35 ---- everything will reach test\_demo1 file ------but if you want only to execute things at

1. , mention it ------ means [0] index of every element of the tuple.

#data driven and parameterization can be done with return statements **in tuple format**

**Means if below test is run**

def test\_crossBrowser(crossBrowser):

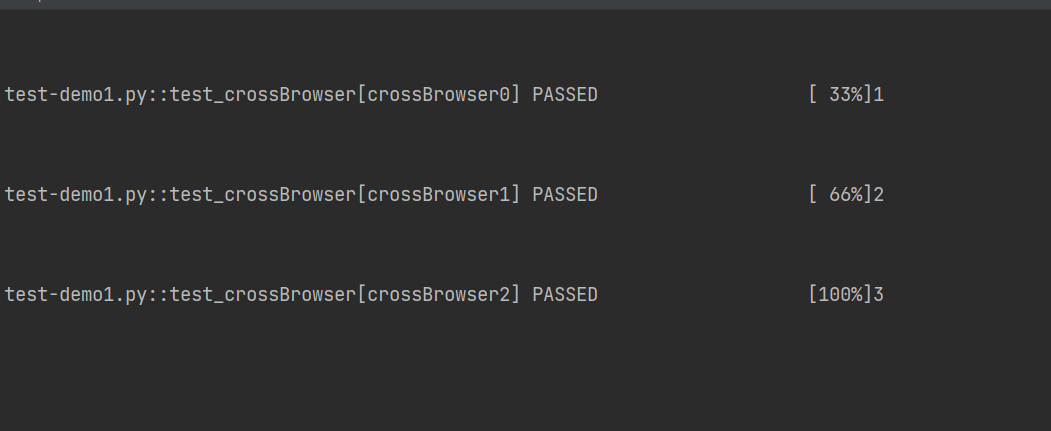
print(crossBrowser[1])

-so in the 1st run

1 will be printed

2nd run-- 2

3rd run -3



**Lecture -77**

2:20

HTML report

py.test --html\_report.html

-s statement is used for logs **2:35**

3:15 -- copy its path from new file created in Pycharm ---- copy it is browser

**Lecture -78**

Log helps to figure out the following

**Note the sequence**

1. Debug
2. Info
3. Warning
4. Error
5. Critical

Lecture -79(**unchecked**)

Text.logging

2:15 -- as testcase name was required

5:40- debug is just like a print statement -debug()

.debug()

.info()

.warning()

.error()

.critical()

9:15

**what to print**

Logger.addHandler(FileHandler) -------tells **which file** and what format required to send

**Where to print**

FileHandler = Logg**ing**.FileHandler(‘logfile.log’)------- **where** file should be printed

**LEcture -80** -----**unchecked**

**What format to print**

Formatter = Logging.Formatter(“%(asctime)s” :%(levelname)s : %(name)s :%(message)s”)

**Levelname ---------**-- to know under what heading should we keep this---error , warning ,…..

1. ------------ to treat it as a string

fileHandler.setFormatter(formatter)

7:00---- below given order (hierarchy) should be maintained

**# dive critical**

.debug()

.info()

.warning()

.error()

.critical()

**8:15** -----logger.setLevel(logging.Info)

-- set level to error --- to just see the error logs

9:35

Wrap all these test cases in a method

Def test\_logging:

--so do code indentation accordingly

12:20 -- now set

Logger.setLevel(logging.DEBUG)

13:05-- now set the level to critical

import logging  
  
def test\_loggingDemo():  
  
 logger = logging.getLogger(\_\_name\_\_)  
 fileHandler =logging.FileHandler('logfile.log')  
 formatter = logging.Formatter("%(asctime)s :%(levelname)s : %(name)s :%(message)s")  
 fileHandler.setFormatter(formatter)  
 logger.addHandler(fileHandler) #filehandling object  
 logger.setLevel(logging.INFO)  
 logger.debug("A debug statement is executed")  
 logger.info("Information statement")  
 logger.warning("Something is in the warning mode")  
 logger.error("A major error has happened")  
 logger.critical("critical issue")

**Lecture -81**

Placing **common reusable function** in 1 class **and inheriting** it

**Integrating Pytest Logs into html Reporting**

1:20 -- create a file with **BaseClass**------ and use it in all the files

**Note** -- change logger.setLevel(logging.DEBUG)----7:40

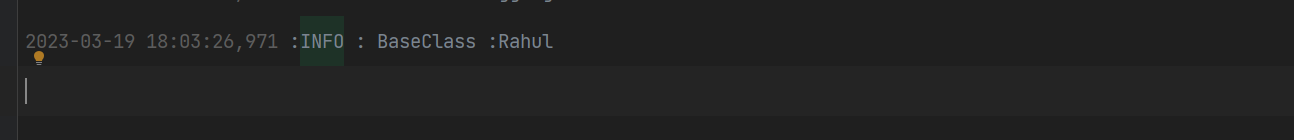
2:15- 2:20 -- see what all to write on this page

Change made as 4:30 --in method

6:40 -- inherit the BaseClass in this file

--- take care of the code indentation

8:30 -- see filename in logfile-- is ----BaseClass -- because we have called it from there



8:45 --add step to give logFile a name from where the method is being called

**loggerName** = inspect.stack()[1][3]

--- test\_editProfile --- will be printed in the logFile

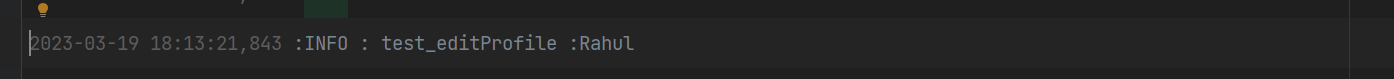
Complete info can be seen in browser --11:55------ **not done**

**Note** -- we are able to successfully connect logs to our report file by using the logger object--log.info

----- print --- **will not** give us the same info

Placing common reusable function in 1 class and how to call it in another class

import inspect  
import logging  
class BaseClass:  
 def getlogger(self):  
 loggerName = inspect.stack()[1][3]  
 logger = logging.getLogger(loggerName)  
 fileHandler =logging.FileHandler('logfile.log')  
 formatter = logging.Formatter("%(asctime)s :%(levelname)s : %(name)s :%(message)s")  
 fileHandler.setFormatter(formatter)  
 logger.addHandler(fileHandler)  
 logger.setLevel(logging.DEBUG)  
 return logger



Part -3

Lecture -85

Create a new project-----pythonSelFramerwork

Create new python package---tests

New python file --- test\_e2e

Writing code **as per py standards**-- means -- everything should be wrapped under 1 class

Things to learn---4:00

1. Standards of writing selenium tests in the framwork
2. Creating **browser invocation Fixtures** in conftest.py

7:00-- check if your current interpreter has required packages or not

Lecture- 86

0:20-- **driver, webdrive**r -- should **not** be used in the test cases

-----------instead placed somewhere else-- to make theme generic(#postman -- globals)

**Using fixtures**

**1:13**

**-- see what to write**

**https://rahulshettyacademy.com/angularpractice/**

5:10 -- writing fixture

@pytest,fixture(scope =”class”)

Import packages too

5:30 -- replace all drivers with self.driver

@pytest.mark.**usefixtures**(“”setup)

7:50 -- create a new python package --utilities

8:50 New python file --BaseClass

@pytest.mark.usefixtures(“setup”)

9:20 ---remove calling fixture in test instead inherit BaseClass

11:30 --- def test\_e2e(self,setup):-------setup is removed

Self.driver

Lecture--88

1:35 --- to pass the tests in the chrome browser

py.test --browser\_name chrome

4:55

**conftest**

def pytest\_a**dd**option(parser):

Parser.a**dd**option(

“--browser\_name”, action=”store”, default=”chrome”

)

**11:10**-- see the full code

@pytest.fixture(scope =”class”)

Def setup(request):

Browser\_name = request.config.getOption(browser\_name”)

If browser\_name == “chrome”:

Driver = webdriver.chrome(executivable\_path=”c:\\chromedriver.exe”)

Driver.get\*(website URL)

Elif browser\_name ==””

Browser\_name == firefox----Gecko Driver

IE -----which browser

11:45--- go to their official documentation -- docs.pytest.org

----------------if any doubt

12:55 py.test --browser\_name firefox

But we must have firefox browser in system

Lecture- 89

Page object design pattern

We will create a checkout page --- and save all the objects on that page

New package -- pageObject

3 python pages for class

1. HomePage
2. CheckoutPage
3. ConfirmPage

3:30 -- for shop option on this page we will create an object

7:05

Note

Driver.find\_element(HomePage.shop)  **is same as**

Driver.find\_element (By.CSS\_SELECTOR,”a[hred\*=’shop’]”)

Class HomePage:

Shop = (By.CSS\_SELECTOR,”a[hred\*=’shop’]”)

def shopItems(self):

**return**  driver.find\_element(\***HomePage.shop**)

Homepage -- here is object name **and not** class name

**9:05**

**Note -- \***  is used so that shop ---- is treated as a **tuple**

Note -- being a **class variable**

It could also have been called as **self.shop ???**

10:00 --- return --- is used so that it can be catched into our test cases

10:25--- send driver actually responsible from test\_e2e.py file to HomePage.py

11:35 ---- def \_\_init\_\_(self, driver)

Self.driver = driver

---- used for above work

12:00 --- homepage = HomePage(driver)------ in test\_e2e.py

Because we created constructor above which accepts driver as an argument

13:00--- must see

-- we created object for class in test\_e2e.py

Now driver in this object will be passed as an argument to our constructor--in HomePage.py

14:00 --

Def shopItem(self):

return **self**.driver---------as for instance variable

14:30 --

Homepage =HomePage(self.driver)

Driver.find\_element (By.CSS\_SELECTOR,”a[hred\*=’shop’]”).click()

**step replaced by**

**Homepage.shopItems().click()**

15:00

So what we did is ----- operation(like .click())--we gave it in the test case but the page object instead of hard coding it in the test case--- we placed it in the specific class (HomePage) --- and we called that by calling object for that class

Lecture -- 90

Similar operation we will do for all the page objects

7:10 --- how to handle XPATH----7:55

--Note--- it was find\_element--- **and not** elements

Lecture- 91

1:10 -- shows interlinking of different pages

2:05 ---- so **click()**  is done in the **class** page only

2:45-- checkout page ka bhi yahi daal diya

------------means creating object for checkout page

&

return checkout page

So now

shopItems().click ----- **not required** on test\_e2e case

3:50 --- change made--test\_e2e

Object **not required** to be created for the next page

6:25 -- similarly for cinform page -- things are handled on the checkoutPage only

7:10 -- redundant things deleted from test\_e2e page

All what we have done is to increase readability

Lecture -92

Creating custom utilities

5:45---- called it by class.shopItems()

**Instead** it should be called as ------object.shopItems()

Lecture -93

Lecture-95to 98 -unchecked

Jenkins

Lecture-101

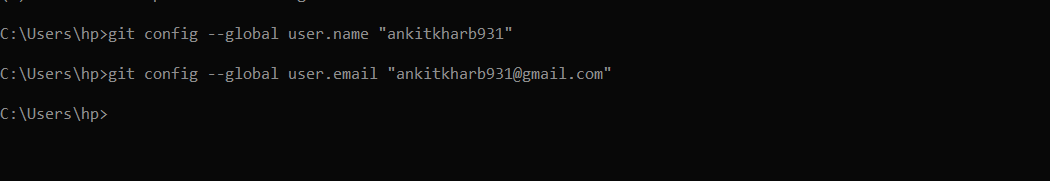
Lecture -114

Basic git commands

--we talk to github through Git command

-below link has all the basic git command

<https://confluence.atlassian.com/bitbucketserver/basic-git-commands-776639767.html>



Reach to the location where you want to work

C:\Users\hp> cd C:\Users\hp\Desktop\ankit\automated testing

C:\Users\hp\Desktop\ankit\automated testing>  **git init**

Initialized empty Git repository

Lecture -115

6:15

Stash -----Commit---GITHUB

-- **GITHUB** only takes the code that is **committed**

----to committ a code you have to add it to **stash**

-