Q1: WHAT IS SELENIUM?

- Selenium is an open source project (library) to automate browsers

Q2: WHY ARE WE USING SELENIUM? WHAT ARE THE ADVANTAGES OF SELENIUM?

- Open source --> Free
- Supports multiple programming languages
- Supports multiple OS (Windows, Mac, Linux)
- Supports multiple browsers
- It has a major community support behind it

Q3: WHAT ARE SOME OF THE DISADVANTAGES OF SELENIUM?

- We cannot automate desktop applications
- Requires advance programming language experience/knowledge
- No costumer service
- There is no built-in report coming from Selenium library

Q4: WHAT IS A WEB ELEMENT?

- Everything we see on a web page from links, to images, to input boxes, to checkboxes all of them are web elements.

Q5: WHAT IS THE DIFFERENCE BETWEEN GETTEXT AND GETATTRIBUTE METHOD?

- .getText():
    - It doesn't accept any argument.
    - It will return the text of the provided WebElement as a String
    - Return type: String
    - .getText() method can only read in between the <openingTag> and </closingTag>

- .getAttribute("attributeName"):
    - It accepts a String argument as an "attributeName".
    - It will find the matching "attributeName" and return its value as a String.
    - Return type is String.
    - Gets the text from the opening tag.

-Q6: WHAT ARE LOCATORS?

- Locators are methods coming from Selenium library that help us locate WebElements.
- There are 8 locators
- id, name, linkText, partialLinkText, cssSelector, xpath, tagname, className

-Q7: WHAT IS YOUR LOCATOR APPROACH? HOW DO YOU DECIDE WHICH LOCATOR TO USE?

- If there is id, I make sure it is not dynamic and I use it.
- If there is class, I can check if it is unique or not by using "." from cssSelector
- If the web element is a LINK, I use "linkText" locator
- If none of the above is applicable, I am comfortable creating custom locators using XPATH.

Q8: HOW MANY TYPES OF XPATH ARE THERE?

#1- ABSOLUTE XPATH:
- Starts with "/" single slash
- "/" means start from the root element "html" and go 1 by 1 to the desired web element.
- This locator is not stable and will break very easily if there is any minimal change in the html page.
- Therefore it is not recommended to use.

#2- RELATIVE XPATH:
- Starts with "//" DOUBLE slash
- "//" means jump to the web element provided
- Relative xpath is more reliable because we are being very specific compared to "absolute xpath"

- //input[@id='something']

- commonly used xpath locators:
  //tagName[@attribute='value']
  //tagName[.='text']
  //tagName[text()='text']
  /following-sibling::
  /preceding-sibling::

Q9: HOW DO YOU HANDLE DYNAMIC WEB ELEMENTS?

We can use the xpath locator methods such as : contains, starts-with, and ends-with to locate web elements that has dynamic attribute value.

//tagName[contains(@attribute, 'value')]
//tagName[starts-with(@attribute, 'value')]
//tagName[ends-with(@attribute, 'value')]

- We can also locate a static (not-changing/unique) parent or child and move from there to desired web element.

Q10: How do we go from child to parent using XPATH?

- "/.." will take our locator from child to parent
- How do we go from parent to child using XPATH?
- "/" will take our locator from parent to child

Q11: What is Maven?

- Maven is a "build automation tool”

Q12: What is a "build"?

- Repeating steps when we are creating and managing our maven project, such as: creating the folder structure, adding, compiling our code, testing, deploying

Q13: What is the most important file in a Maven project?

- pom.xml file

Q14: What is pom.xml file and why do you use it?

- pom -> project object model
- xml -> extensible mark up language
- We manage (add, remove, and change versions) of dependencies and plugins etc.

Q15: What are the differences in between findElement() and findElements() methods

- findElement() method:
- Return type: WebElement type
- It returns a single WebElement

Q16: What happens if it cannot find a WebElement?

- NoSuchElementException will be thrown.
- findElements() method:
- Return type: List<WebElement>
- It returns multiple WebElements in a List of WebElement.

Q17: What happens if it cannot find a WebElement?

- It will not throw exception.
- It will return empty list.

Q18: How do we handle checkboxes and radio buttons?

- First, we locate then we can click.

Q19: How do we verify if checkbox is selected or not?

- isSelected() method:
  - if checkbox/radiobutton is selected, isSelected method will return "true"
  - if checkbox/radiobutton is NOT selected, isSelected method will return "false”

- isEnabled() method:
  - if web element is enabled, isEnabled method will return "true"
  - if web element is NOT enabled, isEnabled method will return "false"

Q20: WHAT IS TESTNG?

- UNIT TESTING FRAMEWORK.
- Originally it was created by a developer for developers.
- As testers we are using some of the annotations and methods to create certain structure for our tests.

Q21: Why do we use annotations?

- Annotations allows us to change the behaviors of regular java methods and allows us to create certain executable flow.

Q22: How do we handle DROPDOWNS?

#1- Non-select dropdowns (HTML):
- Just locate with any locator and click.

#2- Select dropdowns:
- If a dropdown is created using <select> tag, we can use SELECT class' object and methods coming from it.

Syntax:

```
Select yearDropdown = new Select(dropdown_as_WebElement);
Select monthDropdown = new Select(dropdown_as_WebElement);
```

Q23: How do we get currently selected option using select object?

- yearDropdown.getFirstSelectedOption() --> currently selected option as a WebElement
- return type: WebElement

Q24: How many options do we have for selecting <option> from a dropdown?

#1- selectByIndex(int) -> accepts index as int, and indexes start from 0
#2- selectByValue(String) -> accepts the value of attribute "value"
#3- selectByVisibleText(String) -> accept the text of the option as a String as it is displayed on the page

Q25: How do we get all of the options?

- yearDropdown.getOption();
- Return type: List<WebElement>

Q26: HOW MANY TYPES OF ALERTS WE HAVE AND HOW TO HANDLE IT?

#1- Non-JavaScript (HTML) Alerts:
- If it is not blocking the page, if you are able to right click and inspect, it is an HTML Alert.
- For handling, We inspect, locate and click just as any other WebElement.

#2- JavaScript Alerts
- If it is blocking the page, if you are bot able to inspect, it is a JS Alert.

#1- Information Alert:
- User can only .accept();

#2- Confirmation Alert:
- User can .accept(), and decline()

#3- Prompt Alert:
- User can .accept(), decline(), sendKeys();

- Handle JS Alerts using Alert from Selenium.

```
Alert alert = driver.switchTo().alert();
Alert alert = Driver.getDriver().switchTo().alert();
alert.accept();
alert.dismiss();
alert.sendKeys("I can send keys here");
```

Q27: What is iframe?

- HTML inside of another HTML.

Q28: How do we handle it?

- We need to locate the iframe and switch to it.

Q29: Why do we have to handle it?

- Because selenium can focus one thing at a time.
- By default, it will be looking in the main <html> code.
- If there is any inner <html> we must explicitly switch to it to be able to do any action in it.
- Otherwise, Selenium will not be able to see any web element from the inner html <iframe>

Q30: How many ways do we have to switch to an iframe?

#1- index: indexes start from 0.
driver.switchTo().frame(int index);

#2- id, name: if there is id or name attribute we can use to locate and switch to iframe.
driver.switchTo().frame(String id/name);

#3- WebElement: we can locate the iframe as a web element and switch to it.
driver.switchTo().frame(WebElement);
driver.switchTo().frame(driver.findElement(By.locator);

Q31: After switching to inner frame, how do we go back to parent frame?

- parentFrame() -> this will switch back to direct parent
- defaultContent() -> will switch back to the default <html> of the page

Q32: What is the difference between a window and a tab for selenium?

- Both are same for selenium.
- Both TABS and WINDOWS will be treated as WINDOWS.

Q33: How do we handle WINDOWS?

- We use window handles to handle windows.

Q34: What is a window handle?

- Randomly generated alphanumeric unique id for each window or tab
driver.switchTo().window(windowHandle);

Q35: What is properties file?

- It is just another type of file just like .txt, and .pdf whatever.
- But this file has .properties extension.

Q36: Why do we use properties file? What makes it different then other type of files?

- It stores value in "key=value" format
- We are trying to avoid hard coding some of the important test data in our project.

Q37: What is hard coding?

- Writing data directly inside of the source code is called hard coding.
- If I have to go inside of .java class to change the data, it means I hard coded that data.

Q38: Why did we create Driver utility class and method?

#1- We were having hard time passing the SAME(current) driver instance in different classes and different packages.
#2- We had to write too many lines just to instantiate our "driver"
#3- Now we are not only instantiating our driver in just one line, but we are also optimizing the setups.
- determine the type of browser by reading from "configuration.properties" file
- window.maximize is implemented in our Driver util class
- timeOut.implicitlyWait is implemented in our Driver util class

- we created a private constructor, and closed access to the object.
- we created a private static WebDriver.
- we created a public method which delivers the WebDriver instance in the way we want to deliver.
- the way we want to deliver:
  - if session does not exist, it will create a new session/instance
  - if session already exists, it will return existing session.

Q39: How are we able to return the same instance of our driver?

- We implemented Singleton Design Pattern.

Q40: What is Singleton Design Pattern?

- Singleton Design Pattern makes sure we are returning the same instance every time we call our method.
- In our case we are trying to return the same instance of DRIVER

- We achieved this by implementing Singleton Design Pattern.

```
if (driver == null){
    create new
}
return driver;
```

Q41: How to handle the "advanced" mouse and keyboard actions?

- With using Actions class
#1- Create object of Actions class, we pass the Driver instance into constructor
#2- we use the object
#3- we use .perform() method

```
Actions actions = new Actions();
actions.doubleClick(link).perform();
```

Q42: How many ways do you know how to scroll?

#1- window.scroll() method with JSExecutor to scroll certain number of pixels.
#2- js.executeScript("arguments[0].scrollIntoView(true)",element) --> JavaScript method which scrolls until element is visible.
#3- PageUp, and PageDown keys using keyboard actions.
#4- We can use the moveToElement() method from Actions class to scroll to certain web element

Q43: What is JavascriptExecutor?

- A simple interface with 2 methods coming from Selenium library.

Q44: How do we use JavascriptExecutor?

- We downcast our driver type to JavascriptExecutor, to be able to reach methods in it.

```
JavascriptExecutor js = (JavascriptExecutor) Driver.getDriver();

js.executeScript("scrollIntoView")
js.executeScript("scrollBy")
js.executeScript("open new tab")
js.executeScript("sendkeys")
```

Q45: WHAT IS POM DESIGN PATTERN?

- Creating .java class for each page of our web application.
- And store the relevant web elements and methods into their related classes.

Q46: How do we implement POM Design Pattern?

#1- Create a constructor and initialize the object and driver instance using **PageFactory.initElements**() method.

```
public LoginPage(){
    PageFactory.initElements(Driver.getDriver(), this);
}
```

Driver.getDriver() --> provides the current instance of our driver.
this --> provides the current class' object.

- We can think of this as if we are loading our driver instance INTO our class object, so that our class object is able to call Selenium methods.

#2- Use **@FindBy** annotation to locate web elements, instead of findElement();
--> StaleElementReferenceException is solved by POM Design pattern. Because every time we try to use the WebELement the reference of the Web Element will be refreshed. Therefore, no more StaleElementReferenceException

Q47: WHY DO WE USE PAGE OBJECT MODEL DESIGN PATTERN?

- We create centralized repository for our WebElements.
- REUSABILITY
- EASY TO MAINTAIN
- LESS CODE
- CLEANER CODE
- EASY TO COLLABORATE IN BETWEEN TEAM MEMBERS

Q48: What is synchronization? Why do we need it?

- Definition: Multiple things working at the same time.
- We need synchronization because we need to make sure our driver and our browser are on the same page at all times.

Q49: How to handle Synchronization issue?

#1- **Thread.sleep():**
- This is not coming from Selenium library.
- This method comes from JAVA library
- It does not wait for ANY CONDITION TO HAPPEN.
- It will wait for the given duration no matter what.
- Therefore this is not a good practice to use.

#2- **implicitlyWait(10);**
- This is coming from Selenium library.
- Makes driver continue looking for the WebElement for the given duration
- Stops polling as soon as the element is found
- If element is not found it will throw exception after the given time -- > NoSuchElementException
- It will apply to every single line where findElement() or findElements() is used

#3- **ExplicitWait: (WebDriverWait)**

- Comes from Selenium library.
- It can wait for different conditions to happen on the page.
- By default it does POLLING every 500ms.
- If given timer runs out, TimeOutException.

Q50: What is POLLING?

- POLLING is how many times the driver checks the DOM (HTML page) to see if the web element is there (or condition happened) or not.
- By default, polling happens every 250 ms (4 times in second) in WebDriverWait

Q51: How to use explicit waits?

#1- Create object of WebDriverWait class.

```
WebDriverWait wait = new WebDriverWait(Driver.getDriver(), 10);
```

#2- Use the object we just created to create our condition.

```
wait.until(ExpectedConditions.visibilityOf());
wait.until(ExpectedConditions.invisibilityOf());
wait.until(ExpectedConditions.titleIs());
wait.until(ExpectedConditions.titleContains());
wait.until(ExpectedConditions...);
```

#4- **FluentWait:**

- Similar to WebDriverWait
- It waits UP TO given duration and continues if condition happens earlier.
- We have the ability to change POLLING TIME.

- By default, POLLING TIME is 2 times a sec (every 500 millisecond)
- But we can change this to once a second, once every 5 second ETC.
- Waiting 30 seconds for an element to be present on the page, checking for its presence once every 5 seconds.

```
Wait<WebDriver> wait = new FluentWait<WebDriver>(driver)
.withTimeout(Duration.ofSeconds(30)) .pollingEvery(Duration.ofSeconds(5))
.ignoring(NoSuchElementException.class);
WebElement foo = wait.until(new Function<WebDriver, WebElement>() { public
WebElement apply(WebDriver driver) {
return driver.findElement(By.id("foo"));
} });
```