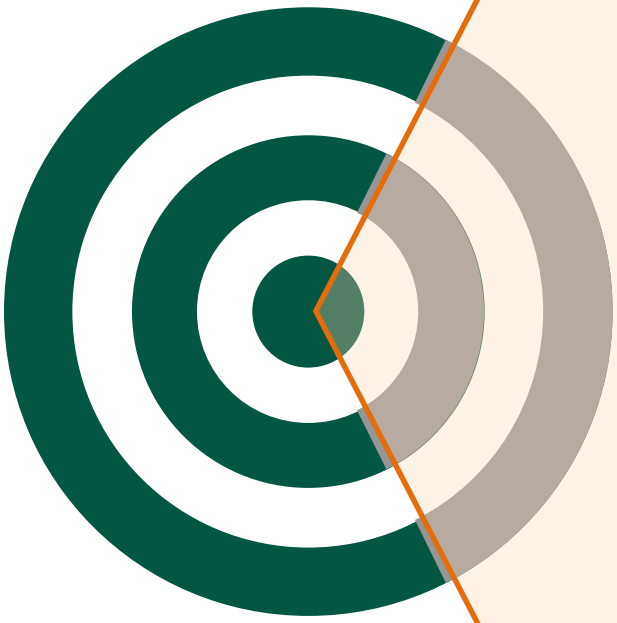
# Cucumber

## Training for State Street

IMARTICUS LEARNING

# Agenda



**In this session, you will learn about:**

- What is BDD (Behavior Driven Development)?
- Basic concepts of writing BDD tests
- What is Gherkin in BDD ?
- Overview of Cucumber
- Basics of Cucumber and its Architecture
- Benefits of BDD
- Use Selenium WebDriver and Cucumber together
- Use Internet Explorer driver to automate the operations on Internet Explorer browser

# Introduction to the Gherkin Language

## The Given-When-Then Notation

### Given

**Given** describes a pre-condition, setting the stage for the actions that will play out in the next section

### When

**When** introduces the user action. The when we're actually talking about the heart of the test

### Then

**Then** describes the results, what we think should happen if the feature executes correctly

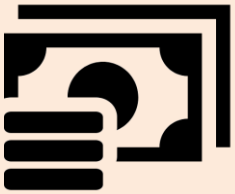
# Basic Concept of Writing BDD

IMARTICUS

## How will a Fund Transfer Work in Net Banking Application?

### MODEL OF ACCESS

### BALANCE SUFFICIENCY



### CORRECT CREDENTIALS

### VALID AMOUNT

### VALID DATE

# Basic Concept of Writing BDD

Let us understand an example of:

How will a fund transfer work in Net Banking application

Hands-on: Jot down scenarios/conditions that would occur during a fund transfer.

# Basic Concept of Writing BDD

**"What If I tell that .."**

These scenarios can be written in structured manner.

Such that the machine can understand.

And can execute as well as provide you an outcome…

**"But how ?"**

# Basic Concept of Writing BDD

In typical BDD world, you would make use of;


Steps/conditional statements

***"Let us see how our core net banking scenarios would transform in BDD approach..."***

# Basic Concept of Writing BDD

Fund transfer module in Net banking in BDD

**What was our pre-condition ?**

That the Net banking is available and user can login


**Given**

**Given** that the fund transfer module is available in net banking and user is accessing with valid credentials.

# Basic Concept of Writing BDD

Fund transfer module in Net banking in BDD

**What was the event ?**

Fund Transfer based on different conditions


| | |
|---|---|
| **When** | **When** user transfers amount with sufficient balance in account and date is current date, future date or date is a holiday and destination account details are valid and transaction credentials are valid |

# Basic Concept of Writing BDD

Fund transfer module in Net banking in BDD

**What was the post-condition ?**

Funds must be transferred

| | |
|---|---|
| **Then** | **Then** amount must be transferred and appropriate transaction logs must be maintained. |

# Basic Concept of Writing BDD

Fund transfer module in Net banking in BDD

**Lets combine the entire scenario**

| | |
|---|---|
| **Given** | **Given** that the fund transfer module is available in net banking and user is accessing with valid credentials. |
| **When** | **When** user transfers amount with sufficient balance in account and date is current date, future date or date is a holiday and destination account details are valid and transaction credentials are valid. |
| **Then** | **Then** amount must be transferred and appropriate transaction logs must be maintained. |

# What is Gherkin in BDD ?

**Gherkin is the language that BDD (or Cucumber) understands, it is:**

**Business Readable**

**Domain Specific Language**

It helps you define the behavior, without getting into technical details of how it is to be implemented

Gherkin is defined as a part of Cucumber libraries, so it would have some conventions.

- **Single Gherkin file contains description of a single feature**
- **Source files have a .feature extension**

# How Does Gherkin Syntax Work?

- Gherkin is a line-oriented language that **makes use of indentation** to define the structure
- **Line endings** would terminate a statement
- **Spaces or Tabs** can be used for indentation
- Lines **start** with a **keyword**
- **Comments** can be done anywhere in the file and begin with **Hash (#) symbol**, with an appropriate text
- The internal parser of Cucumber breaks this file **into 'Features', 'Scenarios' and 'Steps'**

**"Let us see how our example gets transformed using Gherkin"**

We would see some additional concepts about Gherkin when we dive into Cucumber details

# How Does Gherkin Syntax Work?

It has an internal parser that divides the written form of test scripts into:

| Feature | Scenario | Steps |
| --- | --- | --- |
| Some short yet descriptive explanation of what is desired starts the feature and gives it a title | Some determinable business situation starts the scenario, and contains a description of the scenario | Features consist of steps, also known as **Givens**, **Whens** and **Thens** |

**Let us see how our example gets transformed using Gherkin**

We would see some additional concepts about Gherkin when we dive into Cucumber details

# Basic Concept of Writing BDD

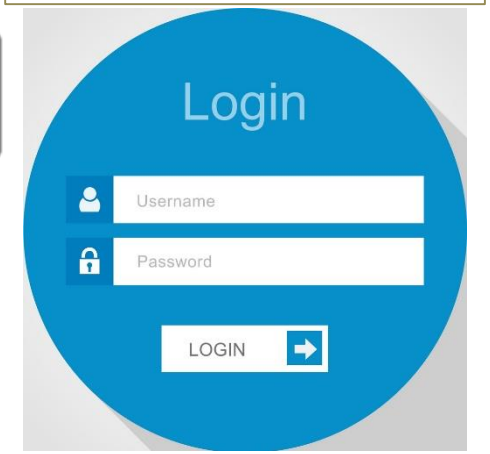
In the fund transfer module

**How will you access the application ?**

**Which browser?**

**URL?**


**Credentials?**

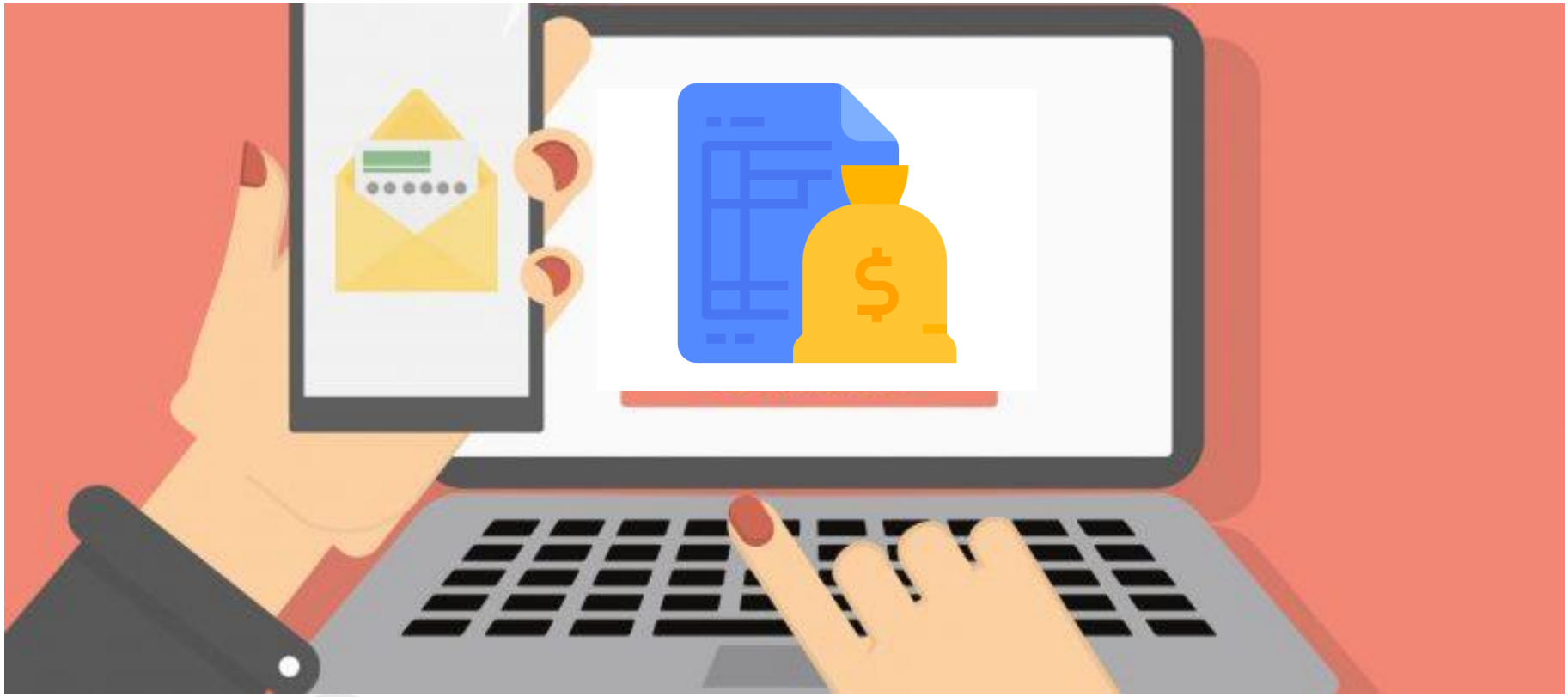



**Hands-on:**

**Jot down scenarios/conditions that would occur during a fund transfer.**

# Basic Concept of Writing BDD

**Once you have logged in, Check if your account has sufficient Balance?**



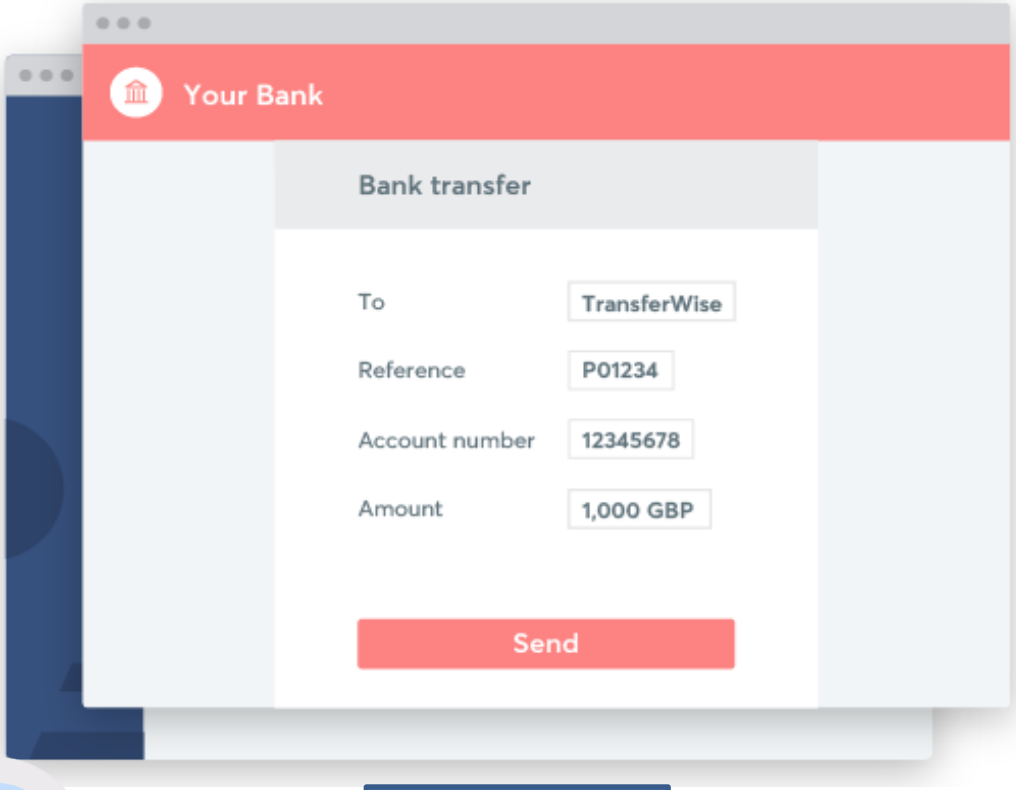
**Hands-on:**

**Jot down scenarios/conditions that would occur during a fund transfer.**

# Basic Concept of Writing BDD

After balance check

**Enter proper details of account where you need to transfer**




**Hands-on:**

**Jot down scenarios/conditions that would occur during a fund transfer.**

# Basic Concept of Writing BDD

**Then**

**Enter a Valid Amount**

**Should be less than your account balance**

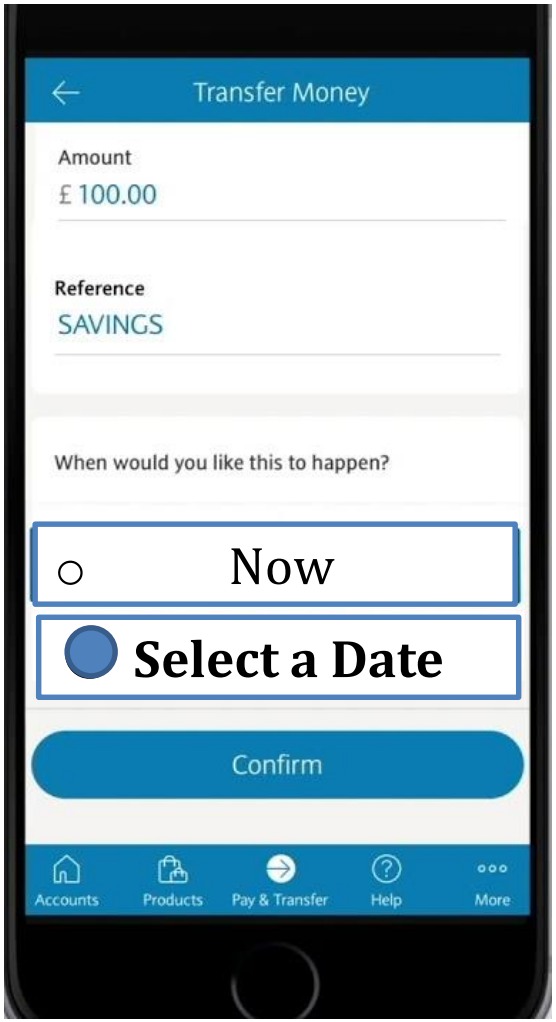

**Hands-on:**

**Jot down scenarios/conditions that would occur during a fund transfer.**

# Basic Concept of Writing BDD

**Then**

**Provide a valid date for the transfer to take place (in case you want to schedule later)**



# Basic Concept of Writing BDD

**Based on the previous pointers, there can be 5-6 core scenarios that can be painted for this module**

**Hands-on:**

**Jot down scenarios/conditions that would occur during a fund transfer.**

# Basic Concept of Writing BDD

## Typical Scenarios

### ENOUGH BALANCE

Transfer must take place if there is **enough balance** in source account.

### VALID DETAILS

Transfer must take place if the destination account **details are valid**.

### CORRECT CREDENTIALS

Transfer must take place if transaction security **credentials** entered by user is **correct**.

### BANK HOLIDAY

Transfer can take place even if it's a **Bank Holiday** (Okay consider IMPS too ;)

### FUTURE DATE

Transfer must take place on a **future date** as set by the sender.

# Basic Concept of Writing BDD

There can be more complex scenarios wherein we can include Limit of Transfer, Daily Limit of Transfer, No. of future days.

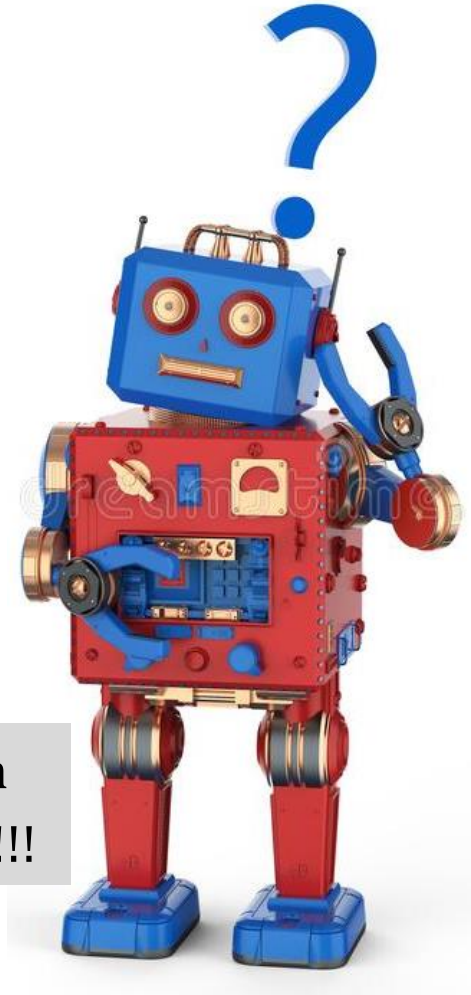

Normally developers would start developing the feature and write some unit tests at the end (maybe in a hasty way), leading to gaps and inefficiency in the testing process.

***"Okay, so tell me how would BDD help ?"***

# Basic Concept of Writing BDD

"But..."

- These scenarios are English like sentences!!!
- How would a machine understand this?
- Is there a syntactical way to write these?
- How will these transform into Scripts?
- Where can I run them?



And a lot of more questions would arise when you talk of Test Automation and Execution ... !!!

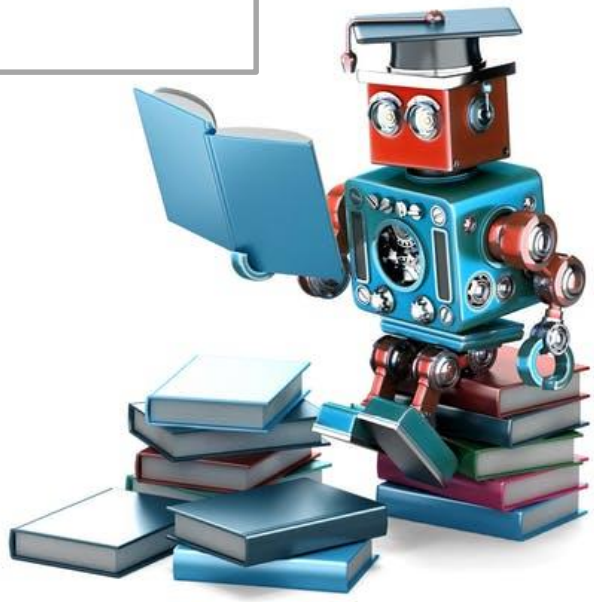
# Basic Concept of Writing BDD

**"But, What If I tell that ..."**

These scenarios can be written in structured manner.

Such that the machine can understand, and can execute as well as provide you an outcome…

**"But How ?"**



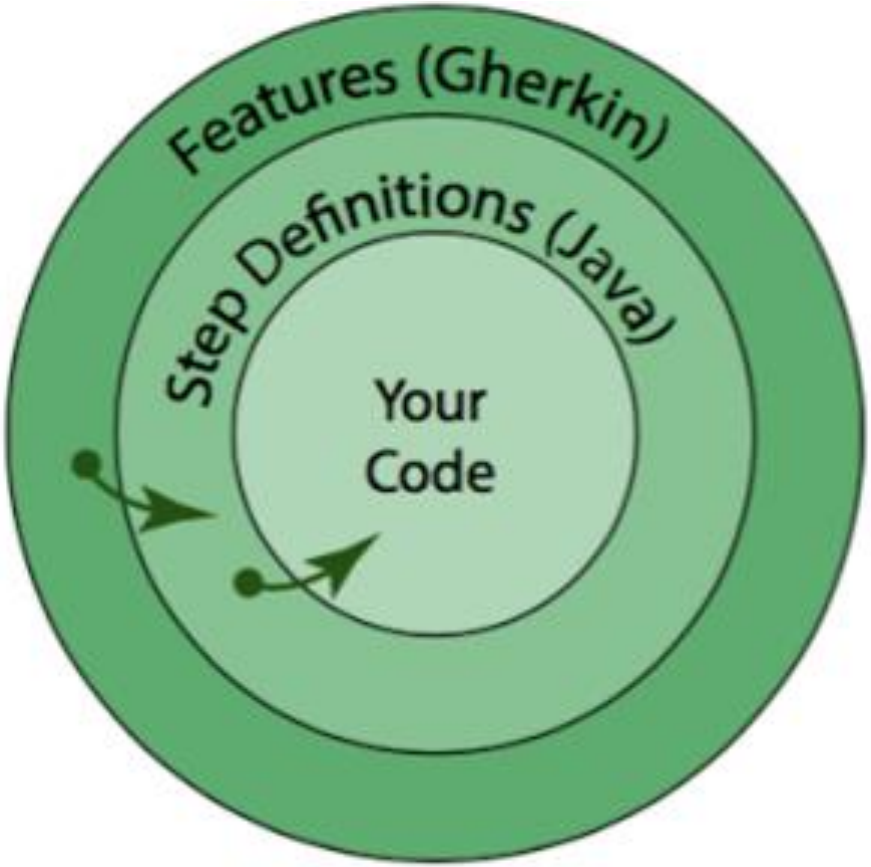
# Cucumber

# What is Cucumber?

Let us now see, how exactly Cucumber bridges the gap of using the Gherkin Feature file and converts it into a format that's technical, i.e. into Java Code.

Cucumber consists of a layer of:

- Features
- Step Definitions
- Runner Class



# Setting the Development Environment

- IDE Plugins
- Maven Build Tool
- Dependencies and Project Structure
- Cucumber -JVM
- AssertJ
- Junit4
- Selenium Client bindings
- Internet Explorer Driver setup

# What is Cucumber?

## Features:

- A Feature can be defined as a **standalone** unit or functionality of a project.

- Each independent functionality of the product under test can be termed as a feature when we talk about Cucumber. It is a **best practice** later when you start testing, that before deriving the test scripts, we should **determine the features to be tested**.

- A feature usually contains a **list of scenarios** to be tested for that feature. A file in which we store features, description about the features and scenarios to be tested is known as **Feature File**.

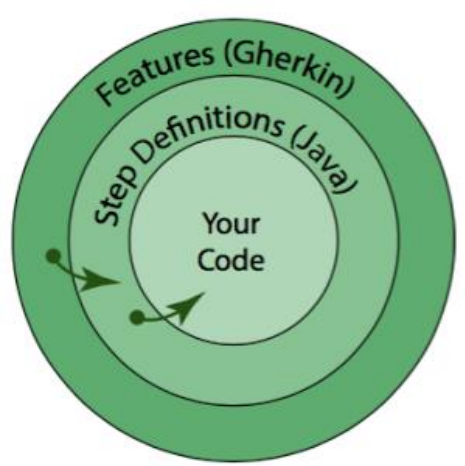
# Step Definitions

As you see from this cross-section of Cucumber; ***Step definitions are related to Java (or Code).***

Cucumber would not simply understand the Step written in Gherkin and start executing.

It needs to **bind the Step** with its **definition**. A **Step Definition** is a piece of **code** with a **pattern** attached to it.

The **pattern** is used to link the step definition to all the **matching Steps**, and the code is what Cucumber will execute when it sees a **Gherkin Step**.



# What is Runner Class?

IMARTICUS LEARNING

Before getting into details of the **Runner class**, it is important to understand what is **Junit**

**JUnit** is an **open source unit testing framework** for the Java programming language. JUnit has been important in the development of test-driven development, and is one of a family of unit testing frameworks. JUnit allows the developer to **incrementally build test suites** to measure progress and detect unintended side effects.

**Cucumber** is providing a way for **non-technical person** to define test cases for a product, and on the other hand, our expectation is for smooth and timely execution of such test cases.

JUnit acts as a bridge between these two

# What is Runner Class ?

So, the flow of execution will look like the following;

- Business Teams will **write down the feature file**.
- **Step definition file** will be created accordingly.
- Specify the **JUnit runner class** to run the series of test cases.

Once we run the JUnit runner class

- It will parse the **Gherkin feature file**.
- It will execute the functions written in the **step definition** file according to feature file statements.
- JUnit will combine the **test case result**.
- It will build the **test report** in the specified format (which can be html/JSON).

# Runner Class

## But who will actually drive the steps ?

Now that you have configured a feature test and also understood how to link the step with the step definitions.

But what would drive these step definitions in the correct order.

Here comes the Runner Class, which actually does not contain any logic regarding the features.

It only has some annotations that cucumber features would run and the feature files to be picked along with the step definition package.

There are quite some other parameters that can be used, that will be discussed at a later stage.

# Runner Class

Here is a sample code for the Runner Class, we have named the class as TestRunner

Location of Feature

Package of Step Definition

```
1  package master;
2
3⊕ import org.junit.runner.RunWith;
7
8  @RunWith(Cucumber.class)
9  @CucumberOptions(
10          features = "Feature"
11          ,glue={"stepDefinition"}
12          )
13 public class TestRunner {
14
15 }
16
```

# Benefits of BDD

- It is helpful to involve business users/stakeholders who are basically non-technical and can't easily read code
- It focuses on end-user experience
- The basic style of writing scenarios/tests allow for easier reuse of code in the tests
- Cucumber has a quick and easy set-up and execution as well
- Its FREE and is compatible with Ruby, Java, Scala, Groovy etc.

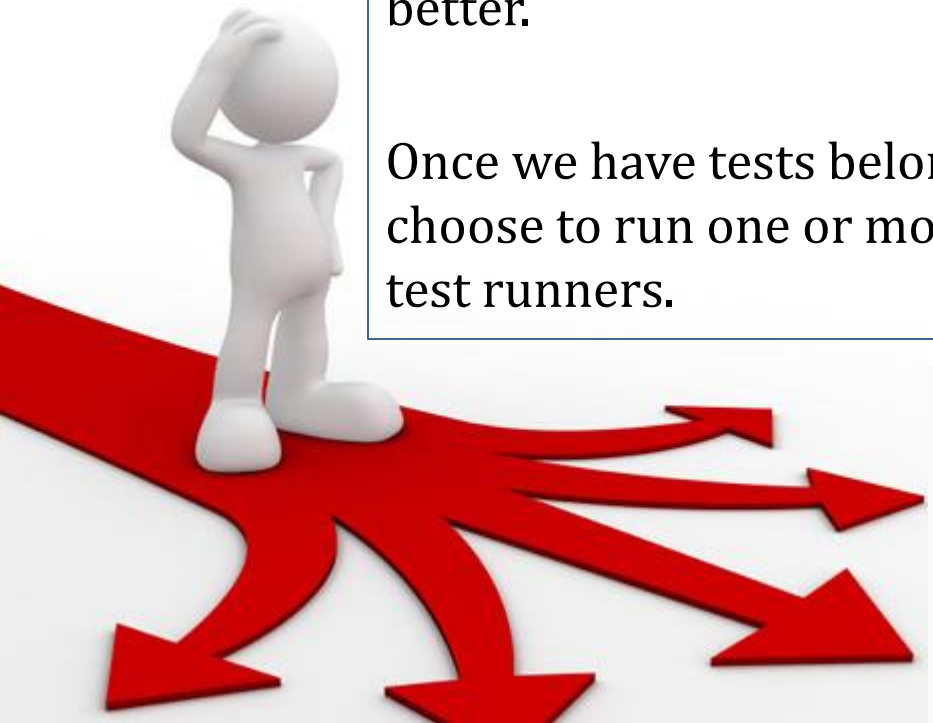
# Tagging Feature Files

**What if you have several feature files with several scenarios and eventually several steps ?**

It becomes difficult to categorize feature files based on modules or some specific functionality.

**Tags** are a feature in cucumber that allows you to **categorize tests**. That way you can organize your feature and scenarios better.

Once we have tests belonging to a certain category you can choose to run one or more combinations of these tags with our test runners.



# Tagging Feature Files

**Several feature files with several scenarios and eventually several steps**

**PROBLEM**

Difficult to categorize feature files based on modules or some specific functionality

**SOLUTION**

**Tags** are a feature in cucumber that allows you to **categorize tests**.

**This can be done in Cucumber Feature Files...**

# Tagging Feature Files

It's not only for feature files, but even **Scenarios can be tagged** as well.

You can also do **multiple tagging to the feature files**.

For e.g: Having Sub-category for tests.

```
@NetBanking @Login
Feature: Login Action

  @Sanity
  Scenario: Successful Login with Valid Credentials
    Given User is on Net Banking Page
    When User Navigates to LogIn Page
    And User enters Valid credentials to LogIn
    Then Message is displayed for Successful Login
```

# Inheritance in Tagging...

**Feature tag is inherited by all the scenarios of the feature file**

```
@NetBanking @Login
Feature: Login Action

  @Sanity
  Scenario: Successful Login with Valid Credentials
    Given User is on Net Banking Page
    When User Navigates to LogIn Page
    And User enters Valid credentials to LogIn
    Then Message is displayed for Successful Login
```

- In the feature file, if we choose to run all tests which are tagged **@NetBanking**, all the scenarios would run.

- This should happen if we choose to run all tests that are tagged **@Login**.

# Running Tests With Tags..

## Tags can be used when specifying what tests to run through the Runner Class

### Demonstration using the Runner Class

```
package master;

import org.junit.runner.RunWith;

@RunWith(Cucumber.class)
@CucumberOptions(
        features = "Feature"
        ,glue={"stepDefinition"}
        ,tags="@NetBanking"
        )
public class TestRunner {

}
```

Specified to use '**@NetBanking**'.

- Tags is just another **parameter** in the cucumber options annotation.
- If required, pass multiple tags as comma separated values.

# Parameterization in Steps..

**Complex scenarios in testing need to test a given scenario with multiple sets of data**

**PROBLEM**

Difficult to always edit the code/methods and run with different data sets.

**SOLUTION**

Essential that tests are parameterized

**This can be done in Cucumber Feature Files...**

# Parameterization in Steps..

**Update step used to specify the login action**

```
@NetBanking @Login
Feature: Login Action

  @Sanity
  Scenario: Successful Login with Valid Credentials
    Given User is on Net Banking Page
    When User Navigates to LogIn Page
    And User enters "Username" and "Password" to LogIn
    Then Message is displayed for Successful Login
```

**AND STATEMENT**

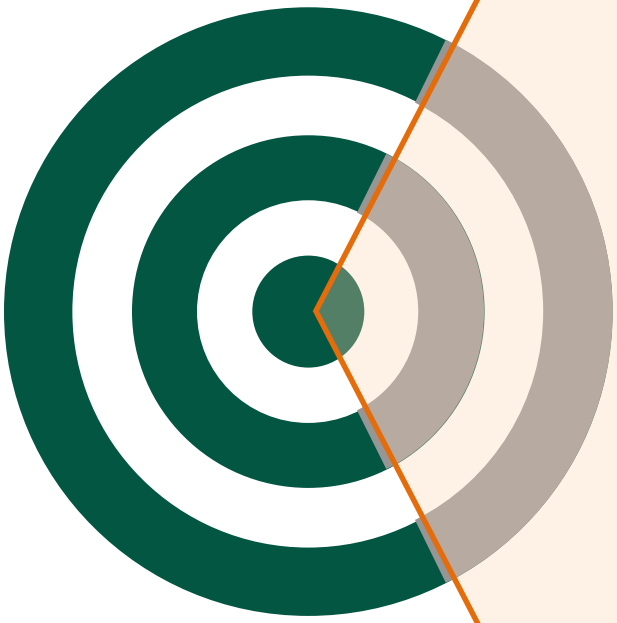
**Parameterization is done to specify the Username and Password in quotes.**

# Cucumber Selenium Integration

## Training for State Street

IMARTICUS LEARNING

# Agenda

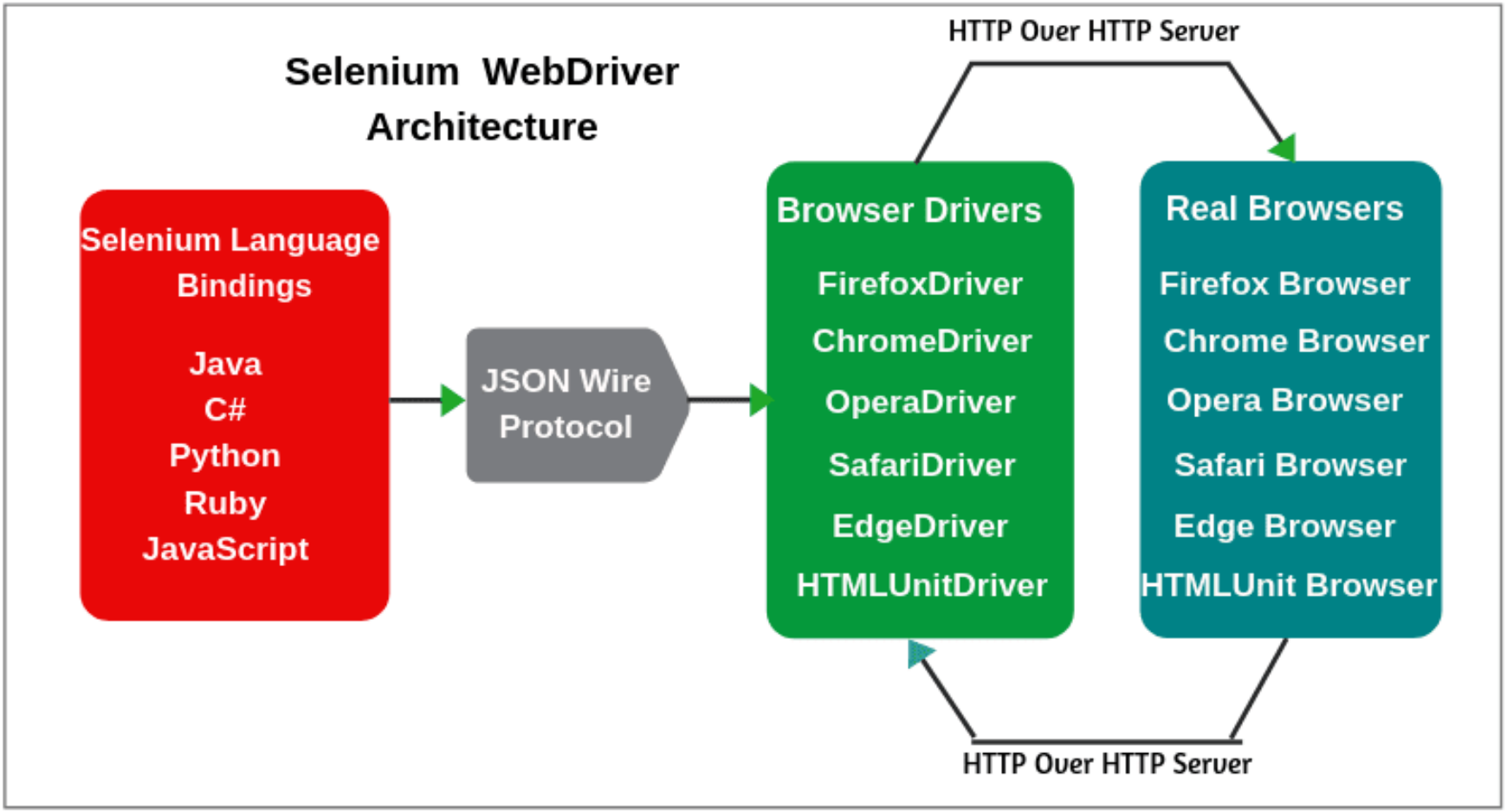


**In this session, you will learn about:**

- Selenium Architecture
- Java client bindings an overview
- Important Packages

# Selenium and WebDriver

# Architecture



# Java Client Binding - Annotations

org.openqa.selenium.Beta (implements java.lang.annotation.Annotation)
org.openqa.selenium.support.FindBy (implements java.lang.annotation.Annotation)
org.openqa.selenium.support.CacheLookup (implements java.lang.annotation.Annotation)
org.openqa.selenium.support.PageFactoryFinder (implements java.lang.annotation.Annotation)
org.openqa.selenium.support.FindBys (implements java.lang.annotation.Annotation)
org.openqa.selenium.support.FindAll (implements java.lang.annotation.Annotation)
org.openqa.selenium.remote.Augmentable (implements java.lang.annotation.Annotation)
org.openqa.selenium.remote.server.jmx.ManagedService (implements java.lang.annotation.Annotation)
org.openqa.selenium.remote.server.jmx.ManagedOperation (implements java.lang.annotation.Annotation)
org.openqa.selenium.remote.server.jmx.ManagedAttribute (implements java.lang.annotation.Annotation)

# Java Client Binding - Enums

org.openqa.selenium.Proxy.ProxyType
org.openqa.selenium.ScreenOrientation
org.openqa.selenium.Platform
org.openqa.selenium.Architecture
org.openqa.selenium.Keys (implements java.lang.CharSequence)
org.openqa.selenium.PageLoadStrategy
org.openqa.selenium.UnexpectedAlertBehaviour
org.openqa.selenium.firefox.FirefoxBinary.Channel
org.openqa.selenium.firefox.FirefoxDriverLogLevel
org.openqa.selenium.interactions.SourceType
org.openqa.selenium.interactions.PointerInput.Kind
org.openqa.selenium.interactions.PointerInput.MouseButton
org.openqa.selenium.interactions.internal.MouseAction.Button
org.openqa.selenium.html5.AppCacheStatus
org.openqa.selenium.internal.ElementScrollBehavior
org.openqa.selenium.json.PropertySetting
org.openqa.selenium.json.JsonType
org.openqa.selenium.support.How
org.openqa.selenium.support.Colors
org.openqa.selenium.ie.InternetExplorerDriverLogLevel
org.openqa.selenium.ie.InternetExplorerDriverEngine
org.openqa.selenium.ie.ElementScrollBehavior
org.openqa.selenium.logging.profiler.EventType
org.openqa.selenium.remote.RemoteWebDriver.When
org.openqa.selenium.remote.Dialect
org.openqa.selenium.remote.http.HttpMethod

# Page Objects

- Modern web applications are characterized by ultra-rapid development cycles, and web testers tend to pay scant attention to the quality of their automated end-to-end test suites. Indeed, these quickly become hard to maintain, as the application under test evolves. As a result, end-to-end automated test suites are abandoned, despite their great potential for catching regressions.

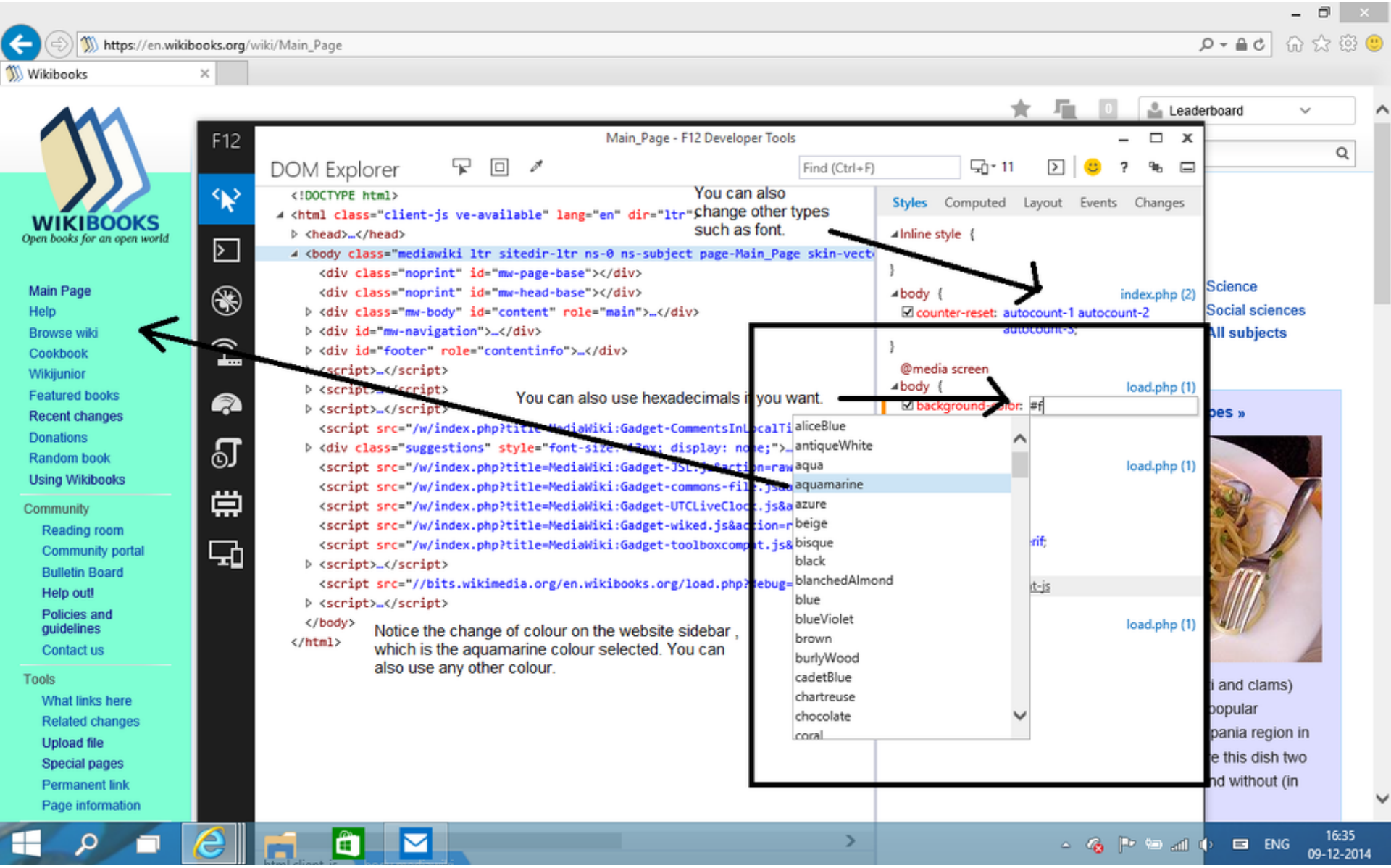
- The use of the Page Object pattern has proven to be very effective in end-to-end web testing.

- Page objects are façade classes abstracting the internals of web pages into high-level business functions that can be invoked by the test cases.

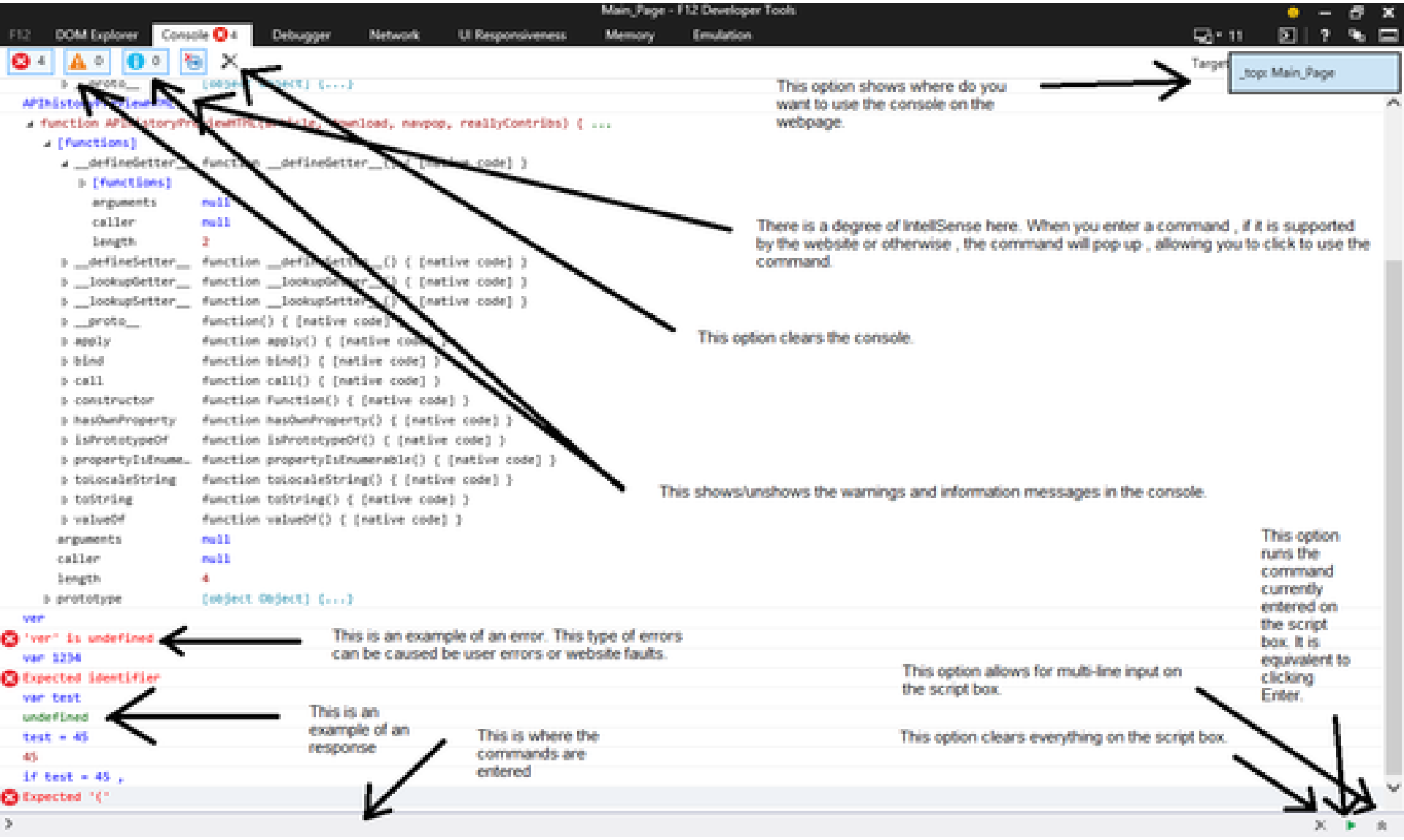
- By decoupling test code from web page details, web test cases are more readable and maintainable.

- However, the manual development of such page objects requires substantial coding effort, which is paid off only later, during software evolution.

# Using the F12 Developer tools



# Using the F12 Developer Tools



# Important System Properties IE

Many of the system properties (read using System.getProperty() and set using System.setProperty() in Java code or the "-DpropertyName=value" command line flag) are used by the InternetExplorerDriver:


# Thank you

**Mumbai | Bangalore | Pune | Chennai | Jaipur**

**ACCREDITED TRAINING PARTNER:**