**In-depth Day-wise Plan for Java-based SDET Journey:**

**Week 1: Java Foundations - Basics**

**Day 1: Introduction and Setup**

* **First 30 mins**: Introduction to what Java is, and its relevance in the industry.
* **Next 30 mins**: Understanding JDK, JRE, and JVM.
* **Next 30 mins**: Installing Java and setting up the environment variables.
* **Last 30 mins**: Writing, compiling, and executing the first "Hello World" program using the command line.

**Day 2: Java Basics - I**

* **First 30 mins**: Understanding data types - primitive vs reference.
* **Next 30 mins**: Declaring and initializing variables.
* **Next 30 mins**: Basic arithmetic operations in Java.
* **Last 30 mins**: Practicing by writing small programs (e.g., calculate the area of a rectangle).

**Day 3: Java Basics - II**

* **First 30 mins**: Introduction to control structures.
* **Next 30 mins**: Hands-on with 'if', 'if-else' conditions.
* **Next 30 mins**: Nested 'if' and 'if-else-if ladder'.
* **Last 30 mins**: Practical exercises (e.g., finding the largest of three numbers).

**Day 4: Loops - I**

* **First 30 mins**: Introduction to looping structures.
* **Next 30 mins**: Deep dive into 'for' loop.
* **Next 30 mins**: Writing programs using 'for' loop (e.g., print the first ten even numbers).
* **Last 30 mins**: Introduction to the 'while' loop.

**Day 5: Loops - II**

* **First 30 mins**: Practice with 'while' loop.
* **Next 30 mins**: Introduction and difference between 'do-while' vs 'while'.
* **Next 30 mins**: Programs using 'do-while' loop (e.g., menu-driven programs).
* **Last 30 mins**: Comparison of loops: when to use which?

**Day 6: Wrap-up and Review**

* **First 30 mins**: Reviewing key concepts learned during the week.
* **Next 60 mins**: Problem-solving exercises combining all concepts.
* **Last 30 mins**: Debugging errors, understanding compiler vs runtime errors.

**Week 2: Java Foundations - OOP**

**Day 1: Introduction to OOP and Classes/Objects**

* **First 30 mins**: Why OOP? Understanding its advantages.
* **Next 30 mins**: Introduction to classes and objects.
* **Next 30 mins**: Writing a basic class and creating objects.
* **Last 30 mins**: Initializing object attributes and using methods.

**Day 2: Constructors and Method Overloading**

* **First 30 mins**: What are constructors? Default vs parameterized constructors.
* **Next 30 mins**: Writing and using constructors in classes.
* **Next 30 mins**: Introduction to method overloading.
* **Last 30 mins**: Writing overloaded methods and understanding their use cases.

**Week 2: Java Foundations - OOP (Continuation)**

**Day 3: Inheritance**

* **First 30 mins**: Introduction to inheritance - the concept of "is-a" relation.
* **Next 30 mins**: Creating a base class (or superclass).
* **Next 30 mins**: Deriving a subclass and using the **extends** keyword.
* **Last 30 mins**: Accessing superclass members, understanding the **super** keyword.

**Day 4: Polymorphism**

* **First 30 mins**: What is polymorphism? Understanding its two main types: compile-time and runtime.
* **Next 30 mins**: Deep dive into method overriding (runtime polymorphism).
* **Next 30 mins**: Method overloading (compile-time polymorphism) revisited with practical examples.
* **Last 30 mins**: Rules for method overriding, understanding the importance of **@Override** annotation.

**Day 5: Abstraction and Interfaces**

* **First 30 mins**: Understanding abstraction, the concept of hiding complex implementation.
* **Next 30 mins**: Introduction to abstract classes and methods.
* **Next 30 mins**: What are interfaces? Differentiating between interfaces and abstract classes.
* **Last 30 mins**: Creating an interface and implementing it using the **implements** keyword.

**Day 6: Encapsulation and Packages**

* **First 30 mins**: What is encapsulation? Understanding its significance in data protection.
* **Next 30 mins**: Using private modifiers, creating getters and setters.
* **Next 30 mins**: Introduction to packages in Java.
* **Last 30 mins**: Creating, accessing, and using packages. Understanding the **import** statement.

**Week 3: Advanced Java Topics and Introduction to Testing**

**Day 1: Exception Handling - I**

* **First 30 mins**: What are exceptions? Differentiating between errors and exceptions.
* **Next 30 mins**: Understanding checked vs unchecked exceptions.
* **Next 30 mins**: Basics of try-catch blocks.
* **Last 30 mins**: Implementing multiple catch blocks and understanding the flow.

**Day 2: Exception Handling - II**

* **First 30 mins**: The significance of the **finally** block.
* **Next 30 mins**: Using the **throw** keyword.
* **Next 30 mins**: Creating custom exceptions with the **throws** keyword.
* **Last 30 mins**: Practical scenarios - when and how to handle exceptions effectively.

**Day 3: Collections Framework - I**

* **First 30 mins**: Introduction to Java Collections Framework.
* **Next 30 mins**: Understanding and using List interface, exploring ArrayList.
* **Next 30 mins**: Set interface and its implementations like HashSet.
* **Last 30 mins**: Practical exercises using List and Set.

**Day 4: Collections Framework - II**

* **First 30 mins**: Map interface, diving into HashMap.
* **Next 30 mins**: Iterating over collections using iterators and enhanced for-loop.
* **Next 30 mins**: Basic operations like add, remove, and search in collections.
* **Last 30 mins**: Introduction to the Comparable and Comparator interfaces for sorting.

**Day 5: Introduction to Software Testing**

* **First 30 mins**: What is software testing? Importance in the software development lifecycle.
* **Next 30 mins**: Different testing levels: unit testing, integration testing, system testing.
* **Next 30 mins**: Introduction to automated testing.
* **Last 30 mins**: Manual testing vs automated testing - pros and cons.

**Day 6: Dive into Software Testing Types**

* **First 30 mins**: Deep dive into manual testing - how it works, and its use cases.
* **Next 30 mins**: Basics of automated testing and tools available.
* **Next 30 mins**: Exploring black box and white box testing.
* **Last 30 mins**: Understanding regression testing and its significance.

**Week 4: Advanced Testing Concepts & Starting with Automation**

**Day 1: Understanding the Testing Lifecycle**

* **First 30 mins**: Dive deep into the Software Testing Life Cycle (STLC) - its stages and significance.
* **Next 30 mins**: Phases of STLC: Requirement Analysis.
* **Next 30 mins**: Test Planning and Test Case Design.
* **Last 30 mins**: Test Environment Setup and its importance.

**Day 2: Deeper into STLC and Test Case Design**

* **First 30 mins**: Test Execution phase and its nuances.
* **Next 30 mins**: Defect Reporting: logging defects effectively.
* **Next 30 mins**: Test Closure: understanding when to conclude testing.
* **Last 30 mins**: Creating effective test cases and best practices.

**Day 3: Introduction to Automation Testing**

* **First 30 mins**: Why automate? Discussing scenarios ideal for automation.
* **Next 30 mins**: Overview of popular automation tools: Selenium, JUnit, TestNG.
* **Next 30 mins**: Setting up the environment for Selenium.
* **Last 30 mins**: Writing the first automated test using Selenium WebDriver.

**Day 4: Diving into Selenium WebDriver**

* **First 30 mins**: Understanding web elements and the DOM.
* **Next 30 mins**: Locating elements using various strategies: ID, Class, Name, XPath.
* **Next 30 mins**: Basic Selenium actions: Click, Input Text, Selecting from Dropdowns.
* **Last 30 mins**: Handling alerts and pop-up windows in Selenium.

**Day 5: Advanced Selenium Concepts**

* **First 30 mins**: Implicit vs Explicit Waits in Selenium.
* **Next 30 mins**: Handling frames and iframes.
* **Next 30 mins**: Introduction to the Page Object Model (POM) for organized test scripts.
* **Last 30 mins**: Creating basic page objects and implementing a simple test using POM.

**Day 6: Beginning with Test Frameworks**

* **First 30 mins**: Introduction to JUnit: Why it's essential for Java-based testing.
* **Next 30 mins**: Setting up JUnit and writing basic test cases.
* **Next 30 mins**: Introduction to TestNG and its advantages over JUnit.
* **Last 30 mins**: Setting up TestNG and exploring basic annotations.

**Week 5: Delving Deeper into Automation & Performance Testing**

**Day 1: JUnit & TestNG Deep Dive**

* **First 30 mins**: Exploring assertions in JUnit.
* **Next 30 mins**: Grouping and prioritizing tests in TestNG.
* **Next 30 mins**: Parameterized testing in TestNG.
* **Last 30 mins**: Running parallel tests and understanding test suites in TestNG.

**Day 2: Selenium Advanced Topics**

* **First 30 mins**: Handling file uploads and downloads in Selenium.
* **Next 30 mins**: Interacting with JavaScript using the JavascriptExecutor.
* **Next 30 mins**: Handling cookies in Selenium for web session manipulation.
* **Last 30 mins**: Best practices for Selenium test scripting.

**Day 3: Introduction to Performance Testing**

* **First 30 mins**: Why is performance testing crucial?
* **Next 30 mins**: Different types of performance testing: Load, Stress, Endurance.
* **Next 30 mins**: Introduction to JMeter as a performance testing tool.
* **Last 30 mins**: Setting up JMeter and understanding its GUI.

**Day 4: Basic Performance Testing with JMeter**

* **First 30 mins**: Creating a basic Test Plan in JMeter.
* **Next 30 mins**: Configuring and running a simple Load Test.
* **Next 30 mins**: Analyzing results using various listeners in JMeter.
* **Last 30 mins**: Understanding Samplers and Logic Controllers.

**Day 5: Advanced Performance Testing**

* **First 30 mins**: Parameterizing tests in JMeter using CSV Data Set Config.
* **Next 30 mins**: Setting up and understanding Thread Groups.
* **Next 30 mins**: Recording and simulating real-world user flows using JMeter's Proxy.
* **Last 30 mins**: Correlation and dynamic data handling in JMeter.

**Day 6: Wrap Up and Reflection**

* **First 30 mins**: Reviewing key concepts learned during the week.
* **Next 60 mins**: Practical scenario: Automating a complete user flow using Selenium and TestNG.
* **Last 30 mins**: Reflect on areas of improvement, gather resources, and plan for the coming weeks.

**Week 6: API Testing & Basics of Mobile Testing**

**Day 1: Introduction to APIs**

* **First 30 mins**: Understanding what an API is and its relevance.
* **Next 30 mins**: Different types of APIs: REST, SOAP, GraphQL.
* **Next 30 mins**: Basics of HTTP: methods (GET, POST, PUT, DELETE) and status codes.
* **Last 30 mins**: Setting up an environment to test APIs (e.g., Postman installation).

**Day 2: API Testing Basics**

* **First 30 mins**: Using Postman to send basic requests.
* **Next 30 mins**: Exploring endpoints, headers, and parameters.
* **Next 30 mins**: Validating API responses.
* **Last 30 mins**: Introduction to automation in API testing.

**Day 3: Automated API Testing**

* **First 30 mins**: Introduction to Rest-Assured for API automation in Java.
* **Next 30 mins**: Setting up Rest-Assured and writing your first automated API test.
* **Next 30 mins**: Validating API responses using Rest-Assured.
* **Last 30 mins**: Parameterizing tests and reusing code for different endpoints.

**Day 4: Introduction to Mobile Testing**

* **First 30 mins**: Understanding the importance of mobile testing.
* **Next 30 mins**: Differences between mobile testing and web testing.
* **Next 30 mins**: Setting up an emulator/simulator for mobile testing.
* **Last 30 mins**: Exploring a basic mobile application and its features for testing.

**Day 5: Mobile Testing Tools**

* **First 30 mins**: Introduction to Appium for mobile testing.
* **Next 30 mins**: Setting up Appium and its prerequisites.
* **Next 30 mins**: Writing your first test for a mobile application using Appium.
* **Last 30 mins**: Exploring mobile elements and actions: swipes, pinches, and multi-touch actions.

**Day 6: Wrap Up and Challenges**

* **First 30 mins**: Reviewing key concepts from the week.
* **Next 60 mins**: Real-world scenario: Testing a RESTful API end-to-end, from authentication to CRUD operations.
* **Last 30 mins**: Identifying challenges faced during the week and seeking solutions or resources to address them.

**Week 7: Continuous Integration, Deployment, and Best Practices**

**Day 1: Introduction to Continuous Integration**

* **First 30 mins**: Understanding CI/CD and its importance.
* **Next 30 mins**: Introduction to Jenkins as a CI/CD tool.
* **Next 30 mins**: Installing and setting up Jenkins.
* **Last 30 mins**: Creating a basic Jenkins job.

**Day 2: Integrating Testing into CI**

* **First 30 mins**: Setting up a Jenkins job for automated tests.
* **Next 30 mins**: Configuring notifications for test results.
* **Next 30 mins**: Introduction to build tools: Maven and Gradle.
* **Last 30 mins**: Integrating Maven with Jenkins.

**Day 3: Continuous Deployment Basics**

* **First 30 mins**: Understanding deployment and its stages.
* **Next 30 mins**: Automating deployment using Jenkins.
* **Next 30 mins**: Introduction to Docker and containerization.
* **Last 30 mins**: Basic commands in Docker and understanding images/containers.

**Day 4: Best Practices in Automation Testing**

* **First 30 mins**: Code reusability and modularity.
* **Next 30 mins**: Importance of comments, code clarity, and simplicity.
* **Next 30 mins**: Efficiently handling test data: external files, databases.
* **Last 30 mins**: Continuous learning and staying updated with industry trends.

**Day 5: Exploring Advanced Concepts**

* **First 30 mins**: Dive deeper into Docker and container orchestration.
* **Next 30 mins**: Basics of Kubernetes for container management.
* **Next 30 mins**: Scaling and managing application deployments with Kubernetes.
* **Last 30 mins**: Monitoring and logging: Tools like Grafana and ELK stack.

**Day 6: Recap and Future Path**

* **First 30 mins**: Review the week's key topics.
* **Next 60 mins**: A hands-on project integrating all concepts: Write tests, automate, integrate into CI, deploy.
* **Last 30 mins**: Planning for the next steps, exploring areas of specialization, and setting goals for the coming weeks.

**Week 8: Advanced Frameworks & Tools**

**Day 1: Introduction to Behavior-Driven Development (BDD)**

* **First 30 mins**: Understanding BDD and its advantages.
* **Next 30 mins**: Introduction to Cucumber as a BDD tool.
* **Next 30 mins**: Writing your first Gherkin script.
* **Last 30 mins**: Setting up Cucumber with Java and Selenium.

**Day 2: Advanced Cucumber Features**

* **First 30 mins**: Cucumber step definitions and glue code.
* **Next 30 mins**: Parameterizing tests with Example keyword.
* **Next 30 mins**: Using Scenario Outline for multiple sets of data.
* **Last 30 mins**: Understanding and implementing Hooks in Cucumber.

**Day 3: Integrating BDD with CI/CD**

* **First 30 mins**: Running Cucumber tests in Jenkins.
* **Next 30 mins**: Generating and analyzing Cucumber reports.
* **Next 30 mins**: Best practices for BDD implementation.
* **Last 30 mins**: Integrating BDD frameworks with Git for version control.

**Day 4: Exploring Advanced TestNG Features**

* **First 30 mins**: Dependency testing in TestNG.
* **Next 30 mins**: Grouping and sequencing tests.
* **Next 30 mins**: Using DataProvider for data-driven testing.
* **Last 30 mins**: Integrating TestNG with CI tools like Jenkins.

**Day 5: Cloud Testing Basics**

* **First 30 mins**: Introduction to cloud testing and its advantages.
* **Next 30 mins**: Exploring cloud testing platforms: Sauce Labs, BrowserStack.
* **Next 30 mins**: Setting up a basic test in Sauce Labs.
* **Last 30 mins**: Running parallel tests on different OS-browser combinations in the cloud.

**Day 6: Advanced Automation Techniques**

* **First 30 mins**: Exploring visual validation tools like Applitools.
* **Next 30 mins**: Introduction to robotic process automation (RPA).
* **Next 30 mins**: Combining AI with automation testing: an overview.
* **Last 30 mins**: Discussing future trends in automation testing.

#### ****Week 9: Security Testing & QA Best Practices****

**Day 1: Introduction to Security Testing**

* **First 30 mins**: What is security testing and why is it essential?
* **Next 30 mins**: Understanding common vulnerabilities: SQL Injection, XSS, CSRF.
* **Next 30 mins**: Basic tools for security testing: OWASP Zap, Burp Suite.
* **Last 30 mins**: Conducting a basic vulnerability scan.

**Day 2: Advanced Security Testing Techniques**

* **First 30 mins**: Deep dive into SQL Injection and its prevention.
* **Next 30 mins**: Understanding and preventing Cross-Site Scripting (XSS).
* **Next 30 mins**: Basics of authentication and authorization testing.
* **Last 30 mins**: Tools and practices for secure code development.

**Day 3: QA Processes and Methodologies**

* **First 30 mins**: Understanding QA beyond just testing: Processes and methodologies.
* **Next 30 mins**: Exploring Agile and its impact on QA.
* **Next 30 mins**: The role of QA in DevOps.
* **Last 30 mins**: QA in Waterfall vs. Agile projects.

**Day 4: QA Documentation and Communication**

* **First 30 mins**: The importance of QA documentation.
* **Next 30 mins**: Writing effective bug reports.
* **Next 30 mins**: Test plan and test case documentation best practices.
* **Last 30 mins**: Effective communication techniques for QA professionals.

**Day 5: Performance Testing Deep Dive**

* **First 30 mins**: Advanced scenarios in JMeter.
* **Next 30 mins**: Exploring other performance testing tools: LoadRunner, NeoLoad.
* **Next 30 mins**: Monitoring and analyzing performance metrics.
* **Last 30 mins**: Stress and spike testing: theory and practice.

**Day 6: Wrapping Up and Forward Path**

* **First 30 mins**: Reviewing the key concepts from the week.
* **Next 60 mins**: A comprehensive hands-on challenge combining concepts from the past two weeks.
* **Last 30 mins**: Planning further specializations, industry certifications, and setting up goals for the coming months.

**Week 1: Software Testing Fundamentals (45 hours)**

**Day 1-3: SDLC & STLC (15 hours total)**

**Day 1 (7.5 hours)**

* **Morning (3.75 hours)**: Introduction to SDLC
  + Understand the phases of SDLC.
  + Dive into the Requirement Analysis phase.
* **Afternoon (3.75 hours)**: Deep dive into the Design phase and Implementation phase of SDLC.

**Day 2 (7.5 hours)**

* **Morning (3.75 hours)**: Explore the Testing phase in SDLC.
* **Afternoon (3.75 hours)**: Understand the Deployment and Maintenance phase of SDLC.

**Day 3 (7.5 hours)**

* **Morning (3.75 hours)**: Introduction to STLC
  + Understanding the Requirement Analysis phase in STLC.
* **Afternoon (3.75 hours)**: Dive into the Test Planning and Test Design phases of STLC.

**Day 4-5: Types of Testing (15 hours total)**

**Day 4 (7.5 hours)**

* **Morning (3.75 hours)**: Introduction to Manual Testing.
* **Afternoon (3.75 hours)**: Explore Automated Testing and familiarize with some tools.

**Day 5 (7.5 hours)**

* **Morning (3.75 hours)**: Black Box Testing – Concepts and Techniques.
* **Afternoon (3.75 hours)**: White Box Testing – Principles, Techniques, and Use Cases.

**Day 6: Programming for Testers: Java & Python (7.5 hours)**

* **Morning (3.75 hours)**:
  + Dive into Java basics for testers.
  + Understand JUnit for testing.
* **Afternoon (3.75 hours)**:
  + Introduction to Python for testers.
  + Basics of unittest and pytest frameworks.

**Day 7: Rest or Catch-Up Day**

* Use this day to review, practice, or catch up on any content you might have missed during the week.

**Week 2: Software Testing Advanced Concepts & Foundations (45 hours)**

**Day 8-9: Advanced Testing Concepts with Selenium (10 hours total)**

**Day 8 (7.5 hours)**

* **Morning (3.75 hours)**:
  + Setting up Selenium WebDriver.
  + Writing and executing basic automation tests.
* **Afternoon (3.75 hours)**:
  + Selenium Locators: ID, Name, XPath, CSS Selectors.
  + Handling web elements like buttons, checkboxes, dropdowns.

**Day 9 (2.5 hours)**

* **Morning (2.5 hours)**:
  + Page Object Model (POM) introduction and setup.
  + Writing scalable scripts using POM.

**Day 10: Testing Tools and Performance Testing (7.5 hours)**

* **Morning (3.75 hours)**:
  + Introduction to JIRA: Understanding its interface.
  + Creating and Managing Tickets.
* **Afternoon (3.75 hours)**:
  + Introduction to Performance Testing.
  + Basic JMeter setup and test execution.

**Day 13: Rest or Catch-Up Day**

* Similar to the first week, use this day to review, practice, or catch up on any content you might have missed during the week.
* Given the intensive nature of the curriculum, you might also use this day to rest and recharge.

--------------------------------------------------------------------------------------------------------------------------

**Week 2: Advanced Testing Concepts (45 hours)**

**Day 8-9: Advanced Testing Concepts with Selenium (10 hours total)**

**Day 8 (7.5 hours)**

* **Morning (3.75 hours)**:
  + Setting up Selenium WebDriver.
  + Writing and executing basic automation tests.
* **Afternoon (3.75 hours)**:
  + Selenium Locators: ID, Name, XPath, CSS Selectors.
  + Handling web elements like buttons, checkboxes, dropdowns.

**Day 9 (2.5 hours)**

* **Morning (2.5 hours)**:
  + Page Object Model (POM) introduction and setup.
  + Writing scalable scripts using POM.

**Day 10: Testing Tools and Performance Testing (7.5 hours)**

* **Morning (3.75 hours)**:
  + Introduction to JIRA: Understanding its interface.
  + Creating and Managing Tickets.
* **Afternoon (3.75 hours)**:
  + Introduction to Performance Testing.
  + Basic JMeter setup and test execution.

**Day 11: BDD (Behavior-Driven Development) with Cucumber (7.5 hours)**

* **Morning (3.75 hours)**:
  + Introduction to BDD and its advantages.
  + Setting up Cucumber in the environment.
* **Afternoon (3.75 hours)**:
  + Writing Gherkin scripts.
  + Binding the Gherkin scripts to step definitions in Selenium.

**Day 12: More on BDD & Mock Testing (7.5 hours)**

* **Morning (3.75 hours)**:
  + Advanced scenarios with Cucumber: Data tables, Scenario Outlines.
  + Integrating BDD into the CI/CD pipeline.
* **Afternoon (3.75 hours)**:
  + Introduction to Mock Testing: Concepts and advantages.
  + Using tools like Mockito (Java) to create mock tests.

**Day 13: Rest or Catch-Up Day**

* As before, utilize this day for review, practice, or catching up on any missed content.
* You can also work on simple hands-on projects to apply learned concepts.

**Week 3: Advanced Testing and BDD Integration (45 hours)**

**Day 14: Continuous Integration with Jenkins (7.5 hours)**

* **Introduction to Jenkins and CI/CD (2 hours)**:
  + Definition and principles.
  + The importance of CI/CD in modern software development.
* **Setting Up Jenkins (2.5 hours)**:
  + Installation and initial configuration.
  + Exploring Jenkins Dashboard, managing plugins.
* **Jenkins Jobs and Build Triggers (3 hours)**:
  + Setting up freestyle projects.
  + Triggering builds manually and automatically.

**Day 15: Integrating Testing with Jenkins (7.5 hours)**

* **Selenium Integration with Jenkins (4 hours)**:
  + Configuration considerations.
  + Running automation scripts using Jenkins.
* **Report Generation in Jenkins (2 hours)**:
  + Post-build actions to generate reports.
  + Visualizing test results.
* **Distributed Testing with Selenium Grid and Jenkins (1.5 hours)**:
  + Introduction to Selenium Grid.
  + Parallel test execution.

**Day 16: Dive into BDD and Cucumber (7.5 hours)**

* **BDD Basics (2 hours)**:
  + The philosophy behind BDD.
  + Difference between TDD and BDD.
* **Introduction to Cucumber and Gherkin (3.5 hours)**:
  + Setting up Cucumber.
  + Writing features using Gherkin language.
* **Step Definitions and Glue Code (2 hours)**:
  + Binding feature steps to Java code.
  + Writing reusable step definitions.

**Day 17: Advanced BDD Practices (7.5 hours)**

* **Scenario Outlines and Data-Driven Testing with Cucumber (3 hours)**:
  + Parameterizing tests.
  + Reusing scenarios with different datasets.
* **Hooks and Background in Cucumber (2.5 hours)**:
  + Setup and teardown procedures.
  + Shared steps for multiple scenarios.
* **Integrating Cucumber with Selenium (2 hours)**:
  + Creating automation scripts from BDD scenarios.
  + Handling web interactions in step definitions.

**Day 18: Test Reporting and Collaboration in BDD (7.5 hours)**

* **Cucumber Reports (3 hours)**:
  + Generating detailed test reports.
  + Visualizing pass/fail statistics.
* **Collaborative Practices in BDD (3 hours)**:
  + Sharing features and scenarios within teams.
  + Ensuring all stakeholders are aligned.
* **Living Documentation (1.5 hours)**:
  + Using BDD artifacts as documentation.
  + Continuous updating of features and scenarios.

**Day 19: Test Optimization and Best Practices (7.5 hours)**

* **Test Refactoring (3 hours)**:
  + Recognizing code smells in tests.
  + Refactoring for readability and maintainability.
* **Best Practices in Automation (3 hours)**:
  + Avoiding common pitfalls.
  + Ensuring scalable and maintainable tests.
* **Continuous Learning and Staying Updated (1.5 hours)**:
  + Following key testing blogs and forums.
  + Participating in testing communities.

**Day 20: Review, Recap, and Hands-On Practice (7.5 hours)**

* **Review of the Week's Topics (3 hours)**:
  + Highlighting key concepts.
  + Clarifying doubts.
* **Hands-On Exercises (4 hours)**:
  + Real-world testing scenarios.
  + Collaboration with peers, if possible.
* **Looking Ahead (0.5 hours)**:
  + Glimpse into next week's content.
  + Self-study recommendations.

**Unit testing Using Python:**

**Testing Frameworks in Python**

Python supports test automation with the help of its various tools and libraries.

Following is a list of Python testing frameworks that come with free licenses.

* Unit test: available as part of the Python standard library for unit testing
* Pytest: comes as a stand-alone package for unit testing
* Robot: part of generic test libraries for acceptance testing
* Doctest : available as part of Python standard library for unit testing
* Nose : comes with unittest features along with additional features and plugins for unittest extension
* Testify : comes with unittest and nose features along with additional features and plugins for unittest extension

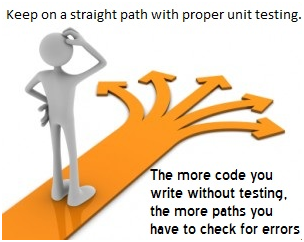
This artifact describes about Unit Test Framework (PyUnit)

**Unit test Framework**

Unit Testing: Testing every single piece of code individually.

Why Unit Testing:

* Quality assurance for the smallest part of code.
* Early detection of Problems (Bugs)
* Implementation of Features
* Cost Reduction



Unit test Overview

* + - A scripted code in python to verify every single module of code.
    - Unittest also referred as PyUnit is a Python language version of Junit (Unit testing framework for Java Programming).
    - Unit test framework is designed to work with Python Library. No PIP (Package Installer for Python) is required.
    - Supports Test Fixture, Test Case, Test Suite, Test Runner.

- Test Fixture: Preparation required for performing tests.

- Test Case: Set of actions to verify specific response to given set of inputs.

- Test Suite: This can be the collection of test cases, test suites or both.

- Test Runner: Component that picks the required set of tests, executes them and displays the outcome to the console or the log files.

Naming Conventions to be followed.

* All unit tests must begin with “test” keyword.
* As good practice name the unit test with test keyword followed by an underscore (\_) then the function name.

Advantages of Unittest Framework

* Comes in-built with Python distribution and hence No PIP installation is needed unlike other frameworks.
* Independence of tests feature as each and every tests run independently of each other
* Unit test helps in reducing bugs in features both existing and new features.
* Helps in reducing the cost of change.
* Provide faster development, debugging and better designing.

import unittest

from LoginPage import LoginPage

class TestLoginPage(unittest.TestCase):

    def test\_login\_successful(self):

        login\_page = LoginPage(" ", "A")

        result = login\_page.Login()

        self.assertEqual(result, "Login successful. Thank you")

    def test\_login\_failed\_incorrect\_username(self):

        login\_page = LoginPage("A", "A")

        result = login\_page.Login()

        self.assertEqual(result, "Please enter the correct username")

    def test\_login\_failed\_incorrect\_password(self):

        login\_page = LoginPage(" ", "Ab")

        result = login\_page.Login()

        self.assertEqual(result, "Password is incorrect please check")

    def test\_login\_failed\_no\_username(self):

        login\_page = LoginPage("", "A")

        result = login\_page.Login()

        self.assertEqual(result, "Please enter username")

    def test\_login\_failed\_no\_password(self):

        login\_page = LoginPage(" ", "")

        result = login\_page.Login()

        self.assertEqual(result, "Please enter password")

"""

    def test\_login\_failed\_username\_and\_password\_empty(self):

        login\_page = LoginPage("", "")

        result = login\_page.Login()

        self.assertEqual(result, "Please enter username and password")

    def test\_login\_failed\_username\_and\_password\_incorrect(self):

        login\_page = LoginPage("J", "24")

        result = login\_page.Login()

        self.assertEqual(result, "Please enter the correct username and password")

"""

if \_\_name\_\_ == "\_\_main\_\_":

    unittest.main()