

K12 APPLICATION WITH PYTHON FOR AUTOMATION TESTING

A PROJECT REPORT

submitted to

Rayalaseema University, Kurnool

in partial fulfillment of the requirements

for the award of degree of

BACHELOR OF TECHNOLOGY

in

COMPUTER SCIENCE AND ENGINEERING

by

ARIPOGULA DIVAKAR BABU	[Reg. No.: 20RU1A0506]
ERUKULA HARI KRISHNA	[Reg. No.: 20RU1A0513]
MURAHARI VIJAYA SATVIKA	[Reg. No.: 20RU1A0534]
NENAVATH NAGENDRA NAIK	[Reg. No.: 20RU1A0538]

Under the esteemed guidance of

Sri. V.SATISH KUMAR M.Tech (Ph.D)

Assistant Professor (Ad hoc)



DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

RAYALASEEMA UNIVERSITY COLLEGE OF ENGINEERING

[RAYALASEEMA UNIVERSITY, KURNOOL – 518 007, A.P., INDIA]

MAY – 2024



RAYALASEEMA UNIVERSITY COLLEGE OF ENGINEERING

KURNOOL – 518007, Andhra Pradesh (INDIA)

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

CERTIFICATE

This is to certify that the project work entitled **K12 APPLICATION WITH PYTHON
FOR AUTOMATION TESTING** is the record of bonafide work done by

ARIPOGULA DIVAKAR BABU

[Reg. No.: 20RU1A0506]

ERUKULA HARI KRISHNA

[Reg. No.: 20RU1A0513]

MURAHARI VIJAYA SATVIKA

[Reg. No.: 20RU1A0534]

NENAVATH NAGENDRA NAIK

[Reg. No.: 20RU1A0538]

and is being submitted to Rayalaseema University, Kurnool in partial fulfillment of the requirements for the award of degree of **BACHELOR OF TECHNOLOGY** in **COMPUTER SCIENCE AND ENGINEERING** during the academic year 2023-2024.

The results embodied in this project report have not been submitted to any other University or Institute for the award of any degree or diploma.

Signature of Guide

Signature of HOD

Smt. M.N.P.SWETHA PRIYA

Assistant Professor (Ad hoc)
Dept. of CSE
Rayalaseema University
College of Engineering,
Kurnool – 518 007, A.P., India

Sri. V.SATISH KUMAR

Assistant Professor (Ad hoc) &
Coordinator
Dept. of Computer Science and Engineering
Rayalaseema University
College of Engineering,
Kurnool – 518 007, A.P., India

Submitted for university examination held on date

INTERNAL EXAMINER

EXTERNAL EXAMINER

ACKNOWLEDGEMENT

It is my privilege and pleasure to express my profound sense of respect, gratitude and indebtedness to my guide, **Mr. V. Satish Kumar**, M. Tech Coordinator of Computer Science and Engineering, **Rayalaseema University College of Engineering, Kurnool.**, for his indefatigable inspiration, guidance, cogent discussion and encouragement throughout this dissertation work.

I am grateful to **Mr. V. Satish Kumar**, M. Tech Coordinator of Computer Science and Engineering for his support and uninterrupted cooperation during my seminar work.

I am deeply indebted to my Principal **Dr. Y. Hari Prasad Reddy**, M. Tech, Ph. D for his constant support and valuable guidance was a source of inspiration for me.

This Acknowledgement will be incomplete without mentioning my sincere gratefulness to our Honorable Vice Chancellor **Prof. B. Sudheer Prem Kumar Garu**, who has been observed posing valiance in abundance, forwarding my individuality to acknowledge my project work tendentially.

Finally, I wish to acknowledge **my friends, family members and colleagues** for giving moral strength and helping me to complete this dissertation

By

A. DIVAKAR BABU (20RU1A0506)

E. HARI KRISHNA (20RU1A0513)

M. VIJAYA SATVIKA (20RU1A0534)

N. NAGENDRA NAIK (20RU1A0538)

ABSTRACT

In today's software development landscape, automated testing has become indispensable for ensuring the reliability, functionality, and performance of software products. Python, with its simplicity, versatility, and rich ecosystem of libraries, has emerged as a preferred choice for implementing automated testing frameworks. This project abstract delves into the utilization of Python for automation testing, exploring its various facets and methodologies.

The abstract begins by highlighting the significance of automated testing in software development, emphasizing its role in accelerating release cycles, reducing manual effort, and enhancing overall product quality. It then elucidates the rationale behind selecting Python as the primary programming language for automation testing, citing factors such as readability, ease of learning, and extensive library support.

In conclusion, this project abstract provides a comprehensive overview of leveraging Python for automation testing, underscoring its efficacy in streamlining the testing process, enhancing software quality, and driving continuous improvement in software development practices. It serves as a valuable resource for software developers, testers, and quality assurance professionals seeking to harness the power of Python for automated testing initiatives.

CONTENTS

Sl.no	Page no
1. CHAPTER-1	1
1.1 Introduction	1
1.2 Motivation	2
1.3 Problem Definition	3
1.4 Objective of the project	3
1.5 Organization of the report	3
1.6 Literature Review	4
2. CHAPTER-2	7
Installation For Project	7
2.1 Software Installation For Python Projects	7
2.2 Install Python For Windows	9
2.3 Set Environment Variables In Python	11
2.4 Pycharm	11
3. CHAPTER-3	15
Introduction To Automation Testing	15
3.1 Manual Testing Limitations	20
3.2 Advantages of Automation Testing	23
3.3 Limitations of Automation Testing	25
3.4 Selecting Right Automation Tool	28
4. CHAPTER-4	31
Automation Testing : Automation Tools	31
4.1 Functionality Testing Tools	33
4.2 Performance Testing Tools	35
4.3 Security Testing Tools	38
4.4 Licensed Tools	40

5.CHAPTER-5	43
Selenium webdriver	43
5.1 What is Selenium	44
5.2 Why use Selenium	48
5.3 Location Elements	52
5.3.1 ID	52
5.3.2 BY Name	53
5.3.3 By Xpath	55
5.3.4 By CSS Selector	57
6. CHAPTER-6	58
Application:K12	58
6.1 Introduction about K12	58
6.2 Login for K12	59
6.3 Grade for K12	59
6.4 Subject Art for K12	59
7.Implementation and Result Analysis	65
8. Conclusion and Future Enhancements	68

LIST OF FIGURES

1.CHAPTER-2

2.1 Figure	7
2.2 Figure	10
2.4 Figure-1	13
2.4 Figure-2	14

2.CHAPTER-3

3. Figure	19
3.1 Figure-1	23
3.1 Figure-2	23
3.2 Figure	24
3.3 Figure	27
3.4 Figure	28

3.CHAPTER-4

4. Figure	32
4.2 Figure	37
4.3 Figure-1	39
4.3 Figure-2	39
4.4 Figure-1	41
4.4 Figure-2	41

4.CHAPTER-5

5. Figure	44
5.1 Figure-1	45
5.1 Figure-2	46
5.1 Figure-3	47
5.1 Figure-4	47
5.3.1 Figure-1	52

5.3.1 Figure-2	53
5.3.2 Figure-1	54
5. 3.2 Figure-2	54
5.3.3 Figure-1	55
5.3.3 Figure-2	56
5.CHAPTER-6	
6.2 Figure	60
6.3 Figure	61
6.4 Figure	61
6.CHAPTER-7	
7. Figure-1	66
7 Figure-2	66
7 Figure-3	67

CHAPTER-1

1.1 INTRODUCTION:

"Welcome to our exploration of Python for automation testing! In today's rapidly evolving tech landscape, the demand for efficient and reliable testing methodologies is more crucial than ever. Python, with its simplicity, versatility, and powerful libraries, has emerged as a go-to language for automation testing. In this project, we delve into the fundamentals of Python automation testing, uncovering its benefits, exploring essential tools and libraries, and showcasing real-world applications. Whether you're a seasoned QA engineer or a budding tester, join us on this journey to harness the full potential of Python for automation testing."

"In the realm of software development, automation testing plays a pivotal role in ensuring the quality and reliability of applications. As organizations strive to deliver products faster and more efficiently, the need for robust testing frameworks has become increasingly pronounced. Python, renowned for its simplicity, readability, and extensive ecosystem of libraries, has emerged as a frontrunner in the realm of automation testing.

At its core, automation testing involves the execution of predefined test cases and scripts to validate software functionalities automatically. Python's intuitive syntax and dynamic nature make it well-suited for crafting such test scripts, enabling testers to streamline the testing process and detect defects early in the development lifecycle.

1. Ease of Learning and Use: Python's simple and readable syntax makes it accessible for beginners and efficient for experienced testers. Its straightforward syntax reduces the time required to write and maintain test scripts.

2. Rich Ecosystem of Libraries: Python boasts a vast array of libraries and frameworks specifically designed for automation testing. Popular ones include Selenium WebDriver for web testing, Appium for mobile testing, and Pytest for test organization and execution.

3. Cross-platform Compatibility: Python is platform-independent, meaning test scripts written in Python can be executed across different operating systems without modification. This flexibility simplifies the testing process across diverse environments.

4. **Integration Capabilities:** Python seamlessly integrates with various tools and technologies commonly used in software development and testing. It can interact with databases, APIs, and other systems, facilitating end-to-end testing scenarios.

5. **Community Support:** Python has a large and active community of developers and testers who contribute to its ecosystem. This community-driven approach ensures continuous improvement, with ample resources, tutorials, and support available online.

6. **Versatility:** Python can be used for a wide range of testing tasks, including functional testing, regression testing, and performance testing. Its versatility enables testers to address different testing requirements within a unified scripting language.

Overall, Python's combination of simplicity, power, and versatility makes it an excellent choice for automation testing across various domains and industries.

1.2 MOTIVATION :

Python is highly motivating for automation testing due to its simplicity, readability, and extensive libraries/frameworks like Selenium, PyTest, and Behave. It enables testers to write efficient, maintainable, and scalable test scripts, reducing manual effort and increasing test coverage. Additionally, Python's versatility allows integration with various tools and technologies, facilitating seamless automation across different platforms and applications.

Python's appeal for automation testing is deeply rooted in its combination of simplicity, versatility, and robust ecosystem. Its straightforward syntax and readability make it accessible for testers of all levels, allowing them to quickly grasp concepts and write efficient automation scripts. Moreover, Python's extensive libraries and frameworks, such as Selenium, PyTest, and Behave, provide powerful tools tailored specifically for testing needs, enabling testers to automate various aspects of the testing process with ease. The language's cross-platform compatibility ensures that automation scripts can be seamlessly executed across different operating systems, reducing the need for platform-specific configurations. Python's integration capabilities further enhance its utility, allowing testers to seamlessly integrate with other tools and technologies commonly used in the testing workflow, such as continuous integration systems and version control platforms. Additionally, Python's active and supportive community ensures that testers have access to a wealth of resources, tutorials, and community-driven solutions, making it easier to overcome challenges and stay up-to-date with best practices. Overall, Python's motivation for

automation testing lies in its ability to streamline the testing process, improve efficiency, and empower testers to create reliable and scalable automation solutions.

1.3 PROBLEM DEFINITION:

The problem definition for employing Python in automation testing revolves around the inefficiencies and limitations of manual testing processes. Manual testing often consumes significant time and resources, leading to slower release cycles and potential human errors. Testers face challenges in executing repetitive tasks across different platforms, maintaining test coverage as applications grow in complexity, and meeting time-to-market demands without compromising quality. Additionally, resource constraints may hinder comprehensive testing efforts. By leveraging Python for automation testing, these challenges can be addressed through streamlined test execution, increased test coverage, improved accuracy, and faster feedback cycles, ultimately enhancing the overall quality and efficiency of software development processes.

1.4 OBJECTIVE OF THE PROJECT:

The objective of employing Python in automation testing is to streamline and optimize the testing process through the use of automated scripts and tools. By leveraging Python's simplicity, versatility, and extensive libraries/frameworks, testers aim to reduce manual effort, increase test coverage, improve accuracy, and accelerate the testing cycle. The primary goal is to identify and address software defects efficiently, ensuring the quality and reliability of the application while minimizing time-to-market. Additionally, Python automation facilitates the execution of repetitive tasks, integration with other testing tools and systems, and scalability to accommodate complex testing scenarios. Ultimately, the objective is to enhance the overall efficiency and effectiveness of the testing process, enabling faster delivery of high-quality software products.

1.5 ORGANIZATION OF THE REPORT:

The organization of a Python project for automation testing involves structuring the codebase in a logical and maintainable manner to facilitate test development, execution, and maintenance. Typically, this includes:

1. **Directory Structure:** Establishing a clear directory structure to organize test scripts, test data, configuration files, and any other relevant resources.

2. **Modular Design:** Breaking down automation scripts into modular components to promote reusability and maintainability. This involves creating separate modules for different functionalities, such as test cases, page objects, utilities, and configuration settings.
3. **Test Framework:** Selecting and implementing a suitable test framework, such as PyTest or unittest, to provide a foundation for organizing and executing tests efficiently.
4. ***Separation of Concerns:** Ensuring separation of concerns by keeping test logic separate from implementation details. This allows for easier maintenance and debugging, as well as facilitating collaboration among team members.
5. **Documentation:** Including comprehensive documentation to explain the purpose, functionality, and usage of each component within the project. This helps onboard new team members and ensures clarity and understanding throughout the development process.
6. **Version Control:** Utilizing version control systems like Git to manage changes to the codebase, track revisions, and collaborate effectively with team members.
7. **Continuous Integration:** Integrating the automation project with a continuous integration system, such as Jenkins or Travis CI, to automate the build, test, and deployment process. This ensures that tests are run automatically whenever changes are made to the codebase, providing rapid feedback to developers.

By organizing the project in this manner, testers can maintain a structured and efficient workflow, promote collaboration, and ensure the long-term maintainability and scalability of the automation testing solution.

1.6 LITERATURE REVIEW:

A literature review on Python for automation testing would typically encompass various research papers, articles, and publications that explore the use of Python in the context of automated testing. Here's an overview of the key points such a review might cover:

1. **Introduction to Automation Testing:** Provide an overview of automation testing, its importance in software development, and the challenges it addresses.

2. **Python in Testing:** Discuss why Python is popular in the field of testing, highlighting its ease of use, versatility, and rich ecosystem of libraries and frameworks.
3. **Comparative Studies:** Compare Python with other programming languages commonly used in automation testing, such as Java, JavaScript, and Ruby. Highlight the advantages and disadvantages of using Python in comparison to these languages.
4. **Case Studies and Examples:** Present case studies or examples where Python has been successfully employed in automation testing projects. These could include real-world applications, industry-specific use cases, or academic research projects.
5. **Key Libraries and Frameworks:** Review popular Python libraries and frameworks used for automation testing, such as Selenium WebDriver, Appium, Pytest, and Robot Framework. Discuss their features, advantages, and limitations.
6. **Best Practices and Guidelines:** Summarize best practices and guidelines for writing effective automation test scripts in Python. This could include topics such as test script design, code organization, handling dynamic elements, and managing test data.
7. **Challenges and Limitations:** Address the challenges and limitations associated with using Python for automation testing. This could include issues related to performance, scalability, cross-browser testing, and maintaining test scripts over time.
8. **Future Directions and Trends:** Discuss emerging trends and future directions in the field of automation testing with Python. This could include advancements in testing frameworks, integration with emerging technologies (e.g., AI/ML, IoT), and novel approaches to test automation.
9. **Conclusion:** Summarize the findings of the literature review and provide insights into the current state and future prospects of Python for automation testing. Highlight areas for further research and development.

By synthesizing findings from existing research and publications, a literature review on Python for automation testing provides valuable insights into the benefits, challenges, and best practices associated with using Python in this domain.

A comprehensive literature review of Python for automation testing involves examining various resources, including academic papers, technical articles, online documentation, and industry best practices, to understand the state-of-the-art techniques, methodologies, and tools in the field. This review encompasses topics such as the use of Python in test automation, popular testing frameworks and libraries, best practices for test design and implementation, case studies of successful automation projects, and emerging trends and technologies in automation testing. By synthesizing information from diverse sources, testers gain insights into the strengths and limitations of Python for automation testing, identify effective strategies for overcoming common challenges, and stay abreast of advancements in the field. Additionally, a literature review helps testers make informed decisions when selecting tools and techniques for their automation projects, ensuring that they leverage Python's capabilities to their fullest potential while adhering to established industry standards and best practices.

By organizing the project in this manner, testers can maintain a structured and efficient workflow, promote collaboration, and ensure the long-term maintainability and scalability of the automation testing solution.

testers gain insights into the strengths and limitations of Python for automation testing, identify effective strategies for overcoming common challenges, and stay abreast of advancements in the field. Additionally, a literature review helps testers make informed decisions when selecting tools and techniques for their automation projects, ensuring that they leverage Python's capabilities to their fullest potential while adhering to established industry standards and best practices.

CHAPTER-2

INSTALLATION FOR PROJECT

2.1 SOFTWARE INSTALLATIONS FOR PYTHON PROJECTS:

To set up a Python project in PyCharm, start by downloading and installing PyCharm from the JetBrains website. Once installed, open PyCharm and create a new project by selecting "File" > "New Project" from the menu. Choose a location for your project and select the Python interpreter you want to use, either an existing interpreter or a virtual environment.



2.1 Figure

Certainly, here are the steps for installing and setting up a Python project in PyCharm:

- 1. Download and Install PyCharm:** Visit the JetBrains website and download the version of PyCharm (Community or Professional) compatible with your operating system. Follow the installation instructions provided by the installer.
- 2. Create a New Project:** Launch PyCharm and create a new project by selecting "File" > "New Project" from the menu. Choose a location for your project and select the Python interpreter you want to use. You can either use an existing interpreter or create a new virtual environment.

3. Set Up Virtual Environment (Optional): If you choose to create a new virtual environment, PyCharm provides an option to do so during project creation. This isolates your project's dependencies and ensures consistency across different environments.

4. Install Dependencies: If your project has dependencies, you can install them using PyCharm's integrated package manager. Open the terminal within PyCharm (View > Tool Windows > Terminal) and use the pip command to install dependencies, for example:

```
pip install package_name
```

5. Open Existing Project (Optional): If you're working on an existing Python project, you can open it in PyCharm by selecting "File" > "Open" from the menu and navigating to the project directory.

6. Configure Python Interpreter: PyCharm allows you to configure the Python interpreter for your project. You can do this by going to "File" > "Settings" > "Project: <your_project_name>" > "Python Interpreter". Here, you can add, remove, or switch Python interpreters as needed.

7. Set Up Version Control (Optional): If your project is version-controlled using Git or another version control system, you can set up version control integration in PyCharm. Go to "VCS" > "Enable Version Control Integration" and select your preferred version control system.

8. Configure Editor Settings: Customize PyCharm's editor settings according to your preferences. You can adjust settings related to code style, indentation, auto-completion, and more by going to "File" > "Settings" > "Editor".

9. Run and Debug Configuration: Configure run and debug configurations for your project. PyCharm allows you to specify the Python script to run, command-line arguments, working directory, and other settings for running and debugging your code.

10. Start Coding: With your project set up in PyCharm, you're ready to start coding! Write your Python code, run tests, debug, and use PyCharm's features to enhance your development workflow.

By following these steps, you can effectively set up and manage Python projects in PyCharm, leveraging its features to streamline your development process.

2.2 INSTALL PYTHON FOR WINDOWS:

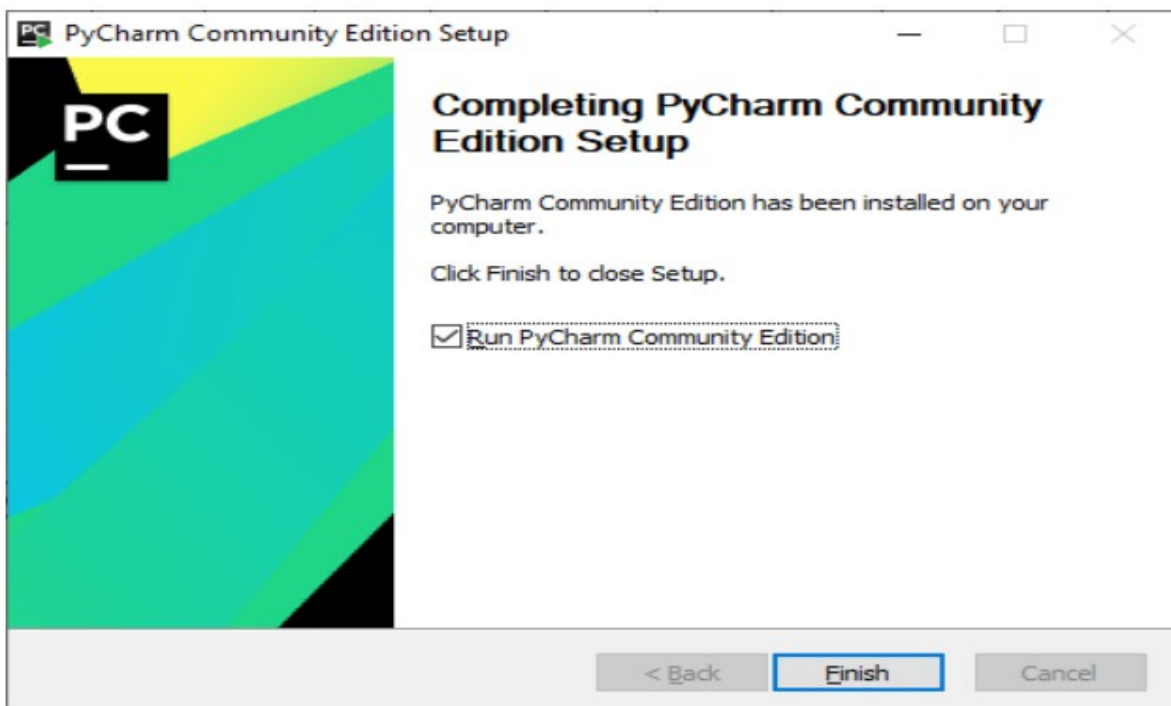
To install Python on a Windows system, follow these steps:

1. **Download Python Installer:** Visit the official Python website at <https://www.python.org/downloads/> and download the latest version of Python for Windows. You can choose between Python 3.x versions.
2. **Run the Installer:** Once the installer is downloaded, double-click on it to run it. You may need administrative privileges to install Python on your system.
3. **Customize Installation (Optional):** During the installation process, you may be given the option to customize the installation. You can choose to add Python to the system PATH, which makes it easier to run Python from the command line, and select additional features to install.
4. **Install Python:** Follow the prompts in the installer to complete the installation process. By default, Python will be installed in the C:\Python39 directory (assuming Python 3.9), but you can choose a different directory if desired.
5. **Verify Installation:** After the installation is complete, you can verify that Python is installed correctly by opening a command prompt and typing `python --version`. This should display the version of Python installed on your system.
6. **Access Python:** You can access Python by opening a command prompt and typing `python`. This will launch the Python interpreter, allowing you to execute Python code interactively. You can also run Python scripts by typing `python path/to/your/script.py` in the command prompt.

That's it! Python is now installed on your Windows system, and you're ready to start coding. You can use any text editor or integrated development environment (IDE) to write and run Python code on your Windows machine.

Windows is a dominant operating system developed by Microsoft, renowned for its user-friendly interface, extensive compatibility, and robust features. Since its inception, Windows has evolved through numerous iterations, each introducing advancements in performance, security, and functionality. Its intuitive graphical user interface (GUI) allows users to navigate seamlessly through applications and settings, making it accessible to users of varying technical expertise. Windows boasts extensive compatibility with a diverse array of software and hardware, catering to both personal and enterprise needs. From productivity suites to multimedia applications, Windows offers a vast ecosystem of software solutions. Moreover, its compatibility with a wide range of hardware configurations ensures versatility and scalability, accommodating the needs of diverse computing environments. With continuous updates and improvements, Windows remains a cornerstone of the computing landscape, empowering users worldwide with a dependable and versatile operating system.

Configure run and debug configurations for your project. PyCharm allows you to specify the Python script to run, command-line arguments, working directory, and other settings for running and debugging your code.



2.2 Figure

it accessible to users of varying technical expertise. Windows boasts extensive compatibility with a diverse array of software and hardware, catering to both personal and enterprise needs. From productivity suites to multimedia applications, Windows offers a vast ecosystem of software solutions. Moreover, its compatibility with a wide range of hardware configurations ensures versatility and scalability, accommodating the needs of diverse computing environments. With continuous updates and improvements

2.3 SET ENVIRONMENT VARIABLES IN PYTHON:

In Python, setting environment variables involves configuring the system environment to hold specific values that can be accessed by Python scripts or other programs running on the system. This process is crucial for configuring the runtime environment of a Python application, enabling it to interact with the operating system, external services, or custom configurations. Environment variables can store information such as paths to directories, authentication credentials, or configuration parameters. Python

provides the `os` module, which includes functions for accessing and modifying environment variables, such as `os.environ` for accessing the current environment variables dictionary and `os.environ.setdefault()` for setting default values. Additionally, external tools like `python-dotenv` can simplify the management of environment variables by loading them from a `.env` file into the Python environment. Properly configuring environment variables in Python ensures portability, security, and flexibility in application development and deployment, allowing developers to seamlessly manage runtime configurations across different environments.

2.4 PYCHARM:

PyCharm is an integrated development environment (IDE) specifically designed for Python development. Developed by JetBrains, PyCharm provides a comprehensive set of tools for writing, debugging, and deploying Python code efficiently. It offers features such as intelligent code completion, syntax highlighting, code analysis, and refactoring tools, which enhance productivity and code quality. PyCharm supports various Python frameworks, including Django, Flask, and Pyramid, with features tailored to each framework's specific requirements. It also integrates with version control systems like Git, enabling seamless collaboration and code management. Additionally, PyCharm offers a robust debugging environment, with support for breakpoints, watches, and interactive debugging sessions. Its intuitive user interface and customizable layout make it suitable for developers of all skill levels. PyCharm is available

in both free (Community) and paid (Professional) editions, catering to the diverse needs of Python developers.

Pycharm is an integrated development environment used explicitly for the Python programming language. In Software development, testing is crucial. Test Automation is the practice of running tests automatically, managing test data, and taking the results to improve the quality of the software. I will show you how to use Pycharm for Test Automation using Selenium.

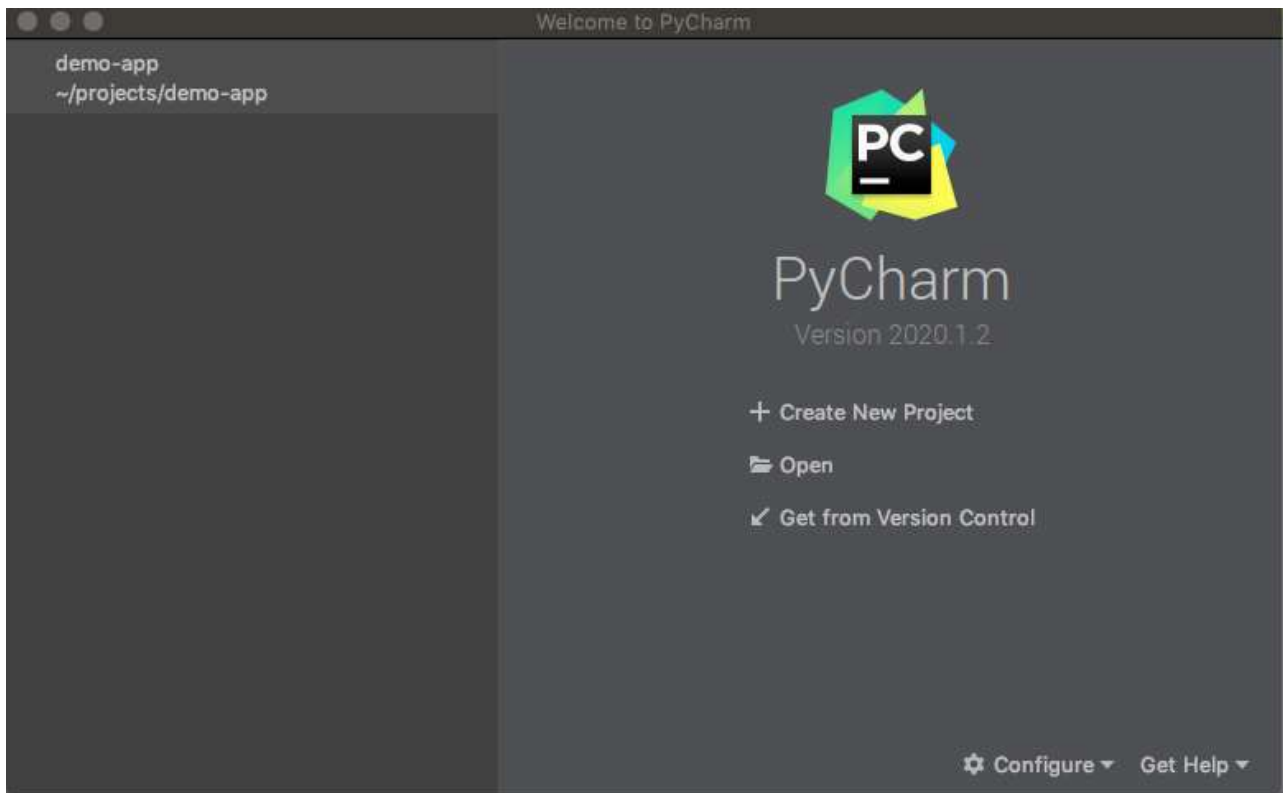
Creating a project in PyCharm is straightforward. Here's a step-by-step guide:

1. **Open PyCharm:** Launch the PyCharm IDE on your computer.
2. **Create New Project:** Click on "File" in the top menu bar, then select "New Project."
3. **Choose Interpreter:** In the New Project dialog box, select the interpreter you want to use for your project. This could be an existing interpreter or a new one.
4. **Project Location:** Specify the location where you want to save your project files. You can choose an existing directory or create a new one.
5. **Project Type:** Select the type of project you want to create, such as a pure Python project, a Django project, or a Flask project.
6. **Project Name:** Give your project a name. This will be the name of the directory where your project files are stored.
7. **Create Project:** Click on the "Create" button to create your project. PyCharm will set up the project structure and open the project in the IDE.

That's it! You've successfully created a project in PyCharm. Now you can start writing code, adding files, and working on your project.

PyCharm is an integrated development environment (IDE) specifically designed for Python development. Developed by JetBrains, PyCharm provides a comprehensive set of tools for writing, debugging, and deploying Python code efficiently. It offers features such as intelligent code completion, syntax highlighting, code analysis, and refactoring tools, which enhance productivity and code quality. PyCharm

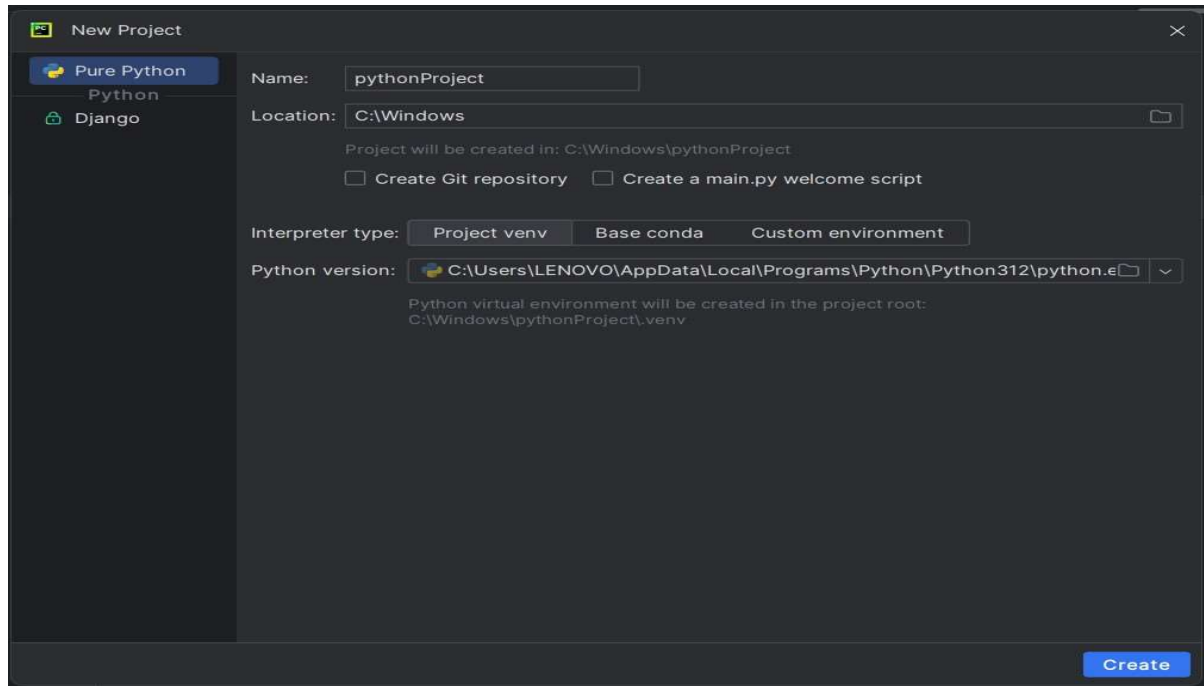
supports various Python frameworks, including Django, Flask, and Pyramid, with features tailored to each framework's specific requirements. It also integrates with version control systems like Git, enabling seamless collaboration and code management. Additionally, PyCharm offers a robust debugging environment, with support for breakpoints, watches, and interactive debugging sessions. Its intuitive user interface and customizable layout make it suitable for developers of all skill levels. PyCharm is available in both free (Community) and paid (Professional) editions, catering to the diverse needs of Python developers.



2.4 Figure-1

PyCharm can be used for code analysis, debugging, and testing, among other things. It is particularly useful for web creation using web application frameworks like Django and Flask. Python plugins can be built by programmers using various APIs. It also allows programmers to access a range of databases without integrating with other tools.

While designed specifically for programming with Python, it can also be used to create HTML, CSS, and Javascript files. It also comes with a great user interface that can be modified based on applications using plugins.



2.4 Figure-2

In PyCharm, creating directories and Python files is straightforward. To create a directory, simply right-click on the project's root folder in the project pane on the left-hand side, select "New," and then "Directory." You can then give the directory a name. Similarly, to create a Python file, right-click on the directory or subdirectory where you want to create the file, select "New," and then "Python File." You can then name the file and start writing your Python code. PyCharm also offers shortcuts for these actions, such as pressing Alt + Insert (on Windows/Linux) or Command + N (on macOS) to bring up a context menu for creating new files or directories. This integrated development environment (IDE) simplifies the process of organizing your project's structure and creating Python files, allowing you to focus on coding efficiently.

CHAPTER-3

INTRODUCTION TO AUTOMATION TESTING:

Software testing is the process in which a developer ensures that the actual output of the software matches with the desired output by providing some test inputs to the software. Software testing is an important step because if performed properly, it can help the developer to find bugs in the software in very less amount of time. Software testing can be divided into two classes, **Manual testing** and **Automated testing**. Automated testing is the execution of your tests using a script instead of a human. In this article, we'll discuss some of the methods of automated software testing with Python. Let's write a simple application over which we will perform all the tests.

One of the major problems with manual testing is that it requires time and effort. In manual testing, we test the application over some input, if it fails, either we note it down or we debug the application for that particular test input, and then we repeat the process. With unittest, all the test inputs can be provided at once and then you can test your application. In the end, you get a detailed report with all the failed test cases clearly specified, if any. The unittest module has both a built-in testing framework and a test runner. A testing framework is a set of rules which must be followed while writing test cases, while a test runner is a tool which executes these tests with a bunch of settings, and collects the results. **Installation:** unittest is available at PyPI and can be installed with the following command –

```
pip install unittest
```

Use: We write the tests in a Python module (.py). To run our tests, we simply execute the test module using any IDE or terminal. Now, let's write some tests for our small software discussed above using the unittest module.

1. Create a file named tests.py in the folder named “tests”.
2. In tests.py import unittest.
3. Create a class named TestClass which inherits from the class unittest.TestCase. All the tests are written as the methods of a class, which must inherit from the class unittest.TestCase.
4. Create a test method as shown below. Name of each and every test method should start with “test” otherwise it'll be skipped by the test runner.

Automation is used widely to perform tasks without needing manual intervention. Since, Python is used across various domains, including software development, data analysis, and more. So, there is a need for professionals familiar with automation.

Why is Automation Important?

Automation is a very important aspect of Python Programming and there are many applications and advantages of automation:

- Automation is crucial in today's fast-paced world as it can help you save time by eliminating repetitive tasks.
- Reduce errors and improve accuracy.
- Increase productivity by allowing you to focus on more important work.
- Perform tasks at a consistent pace 24/7.

Where is Automation Required?

We can use automation in various domains such as:

- Data processing and analysis.
- File and folder management.
- Web scraping and data extraction.
- Task scheduling and reminders.
- GUI automation for repetitive user interactions.

Requirements for Python Automation

Here we have provided you with some of the topics and concepts you should know before starting to learn automation.

1. Python

Python is a high-level, general-purpose, and very popular programming language. Python programming language (latest Python 3) is being used in web development, machine learning applications, along with all cutting-edge technology in the Software Industry.

Python language is being used by almost all tech-giant companies like – **Google, Amazon, Facebook, Instagram, Uber...** etc.

How to Automate a Task?

Automating a task might look very complex, but if you know the basics of Python and you follow these steps. You can easily create an automation script.

Step 1: Identify the Task

Before writing the code, you should have it clear what you want your code to do. So having a clear picture of what the task is, will help you in finishing it.

Step 2: Divide the Task into Smaller Steps

Now break down the desired task into smaller tasks and achieve the initial goal. After achieving the initial goal you can then go to the next one. Breaking the main goal into smaller goals helps you in solving one problem at a time.

Step 3: Research Python Libraries

Now you need to find optimal Python libraries that are most suitable for the task. Using the right libraries will increase efficiency and decrease time consumption.

Step 4: Write the Code

Now you just need to write the Python script, that can automate your desired tasks. Use the right libraries and functions to make writing easier.

Step 5: Test the Code

After you have completed the writing part, test your code on different instances and variations. Testing your code will help you in finding the issues with your code. It also helps in building confidence in the results.

Step 6: Update the Code

If your code is not working properly or there are some instances of wrong results, you can update your code and make corrections. In cases where you might want to add additional features to your code, you can do it in this step.

Python Automation Examples

Let's look at one **simple example of automation**. In this example, we are automating the task of creating a file and writing text in it.

Automating Workflows with Python Scripts

Let's look at some workflows that can be automated using Python scripts. We will also cover some examples of each workflow to understand the work.

1. Web scraping

Web scraping is a technique to fetch data from websites. While surfing on the web, many websites don't allow the user to save data for personal use.

One way is to manually copy-paste the data, which is both tedious and time-consuming. Web Scraping is the automation of the data extraction process from websites. This event is done with the help of web scraping software known as web scrapers.

They automatically load and extract data from the websites based on user requirements. These can be custom-built to work for one site or can be configured to work with any website.

2. GUI Automation

GUI (Graphical User Interface) automation is the process of automating interactions with graphical elements in the user interface of a software application or system.

GUI automation involves simulating user actions, such as clicking buttons, filling out forms, navigating menus, and interacting with windows, to perform tasks or tests without manual intervention.

GUI automation is commonly used in software testing, repetitive task automation, and robotic process automation (RPA).

3. Software Testing Automation

Software testing is the process in which a developer ensures that the actual output of the software matches the desired output by providing some test inputs to the software.

Software testing can be divided into two classes, **Manual testing** and **Automated testing**. Automated testing is the execution of your tests using a script instead of a human. Here we'll discuss some of the methods of automated software testing with Python.

4. API Automation

API automation, often referred to as API testing or automated API testing, is the practice of using automation tools and scripts to test and validate the functionality of Application Programming Interfaces (APIs).

Some examples of API Automation:

- [How To Track ISS \(International Space Station\) Using Python?](#)
- [Track Covid-19 Vaccine Slots using cowin in Python](#)

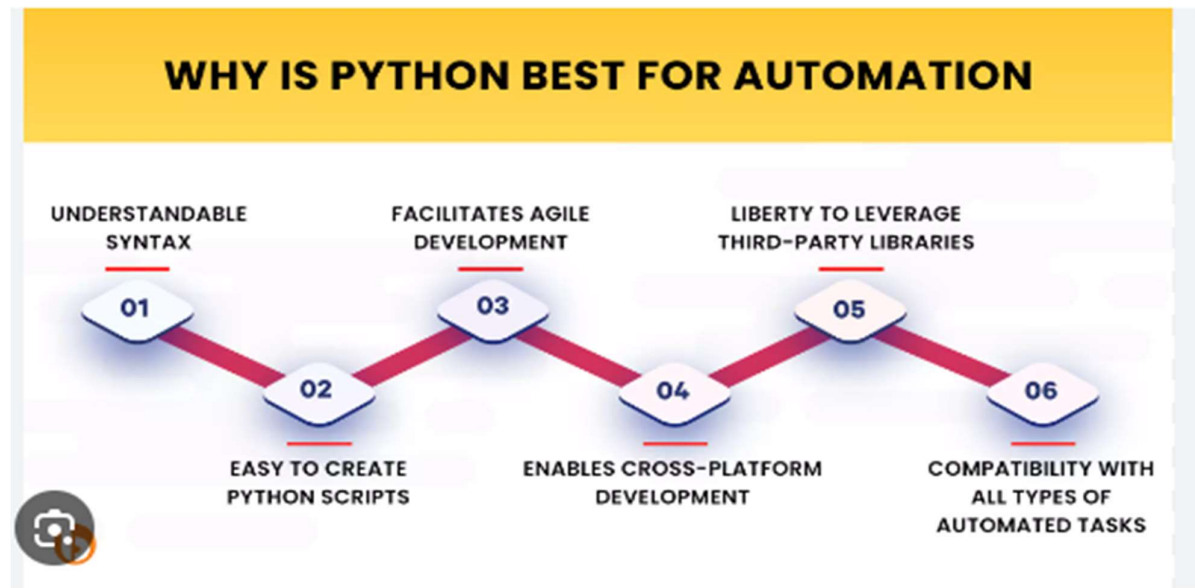
5. Robotic Process Automation

Robotics Process Automation (RPA) is a useful and widely emerging technology in the business world nowadays. RPA is based on Machine Learning and Artificial Intelligence (AI) which uses various software robots to perform business-oriented tasks.

Previously, in many organizations, the large volume of data was handled by the humans themselves. But with the use of RPA, all the maintenance of data can be done easily with the help of a few software, and no manual or human interference is needed.

Some examples of Robotic Process Automation:

- [Robotic Process Automation \(RPA\) – Email Automation using UIPath](#)
- [Hello World Bot Using Automation Anywhere](#)



3 Figure

The process of discovering faults in a created product is known as software testing. Additionally, it assesses if the actual outcomes correspond to the outcomes expected and aids in the identification of errors, needs that are lacking, or gaps. The final step before a product is made available on the market is testing. It involves looking at, studying, observing, and judging a variety of product components. Software testers in the industry combine human and automated testing methods. After running the tests, the testers inform the development team of the results. Software testing is essential since the end goal is to give the user a high-quality product.

Many entrepreneurs neglect to test their products. They may claim that their financial constraints are to blame for skipping this crucial phase. They believe it will have little impact. It must, however, be excellent in order to establish a robust and favorable initial impression. And this necessitates thorough bug testing of the product.

Only when the product supplied is perfect can a company provide value to its clients. And, in order to accomplish so, businesses must ensure that users have no problems when utilizing their products. Making your goods bug-free is the foolproof approach to achieving it. Before delivering a product, companies must focus on testing apps and fixing any flaws that are discovered during testing. The quality of the output improves when the team fixes issues before the product reaches the consumer.

3.1 MANUAL TESTING LIMITATIONS:

Manual testing is the process of manually testing software for defects without using any automated tools or scripts. Testers follow predefined test plans and test cases to perform systematic testing of the software's functional and non-functional requirements.

In manual testing, testers act as end users and test the software step-by-step by applying real world operational scenarios. The aim is to identify any bugs, issues or deviations from the expected behavior.

Some examples of manual testing activities include:

Exploratory Testing: The tester freely explores the software without test cases and tries to break it from an end user perspective. The aim is to discover defects not captured in existing test plans.

Functionality Testing: Verifying all the specified functional requirements of the software are working without problems.

UI Testing: Checking the user interface elements like menus, buttons, icons etc. to ensure correct look, feel, navigation and flow.

Integration Testing: Testing interfaces between modules and different integrated systems to ensure seamless data flow and functionality.

User Acceptance Testing (UAT): Getting feedback from end users to ensure the software meets business needs and user expectations.

Regression Testing: Re-testing previously tested features when changes, enhancements or bug fixes are made to ensure no new issues were introduced.

Compliance Testing: Testing the software for compliance with regulatory standards, organizational policies, compatibilities, etc.

Manual test execution is suited for catching errors related to real world human usage that may not be uncovered by automated testing alone. It allows testers to notice subtle visual or workflow defects.

The process involves developing test scenarios, executing them step-by-step according to the test plan, recording the results and reporting any discrepancies or issues. Meticulous logging is important so defects can be properly tracked, reported and fixed.

Advantages of Manual Testing

While manual testing takes more effort than automated testing, it still provides some unique advantages:

1. Early Defect Detection

Manual testing can begin at the start of the software development lifecycle even before the software is fully developed. Bugs and defects can be detected earlier which are often more economical to fix.

2. Better User Insight

By mimicking real users, manual testers can provide subjective insights into the actual user experience which automated tests may miss. They can evaluate subtle usability issues.

3. Flexible Exploratory Testing

Manual testers can dynamically adapt exploratory tests based on results observed. They are not limited to pre-scripted test cases. Exploratory testing allows catching unpredictable edge cases.

4. Technology Compatibility

Manual testing allows evaluating software compatibility across different hardware devices, operating systems, browsers and interfaces which may be difficult to automate fully.

5. Visual and Design Testing

Subtle visual defects like alignment issues, styling inconsistencies or icon pixelation are easier to catch with manual image comparisons than automation scripts.

6. No Automation Overhead

Manual testing eliminates the time and effort to write and maintain automation scripts. It avoids any disruption when test tools become obsolete or require upgrades.

7. Domain Knowledge Leverage

Manual testers contribute real-world domain expertise which helps identify rational test cases beyond the stated requirements. Their business insight is tough to embed into automated systems.

8. Better Documentation

Manual test execution encourages maintaining detailed documentation of test scenarios, test data, logging and results. Well documented tests increase quality and reproducibility.

For certain types of system, user interface, exploratory, and integrated testing, manual testing still provides advantages over complete automation despite higher effort. It serves as an essential complement to automated testing.

Disadvantages of Manual Testing

While manual testing is indispensable, it also comes with some drawbacks:

1. Time Consuming

Manual testing and result logging take considerable time and effort especially for frequently repeated test cases. Automated tests provide faster test execution.

2. Prone to Human Error

Manual testing can become mundane and repetitive leading to mistakes like data entry errors, missed steps etc. Automation minimizes such errors.

3. Difficult Reproduction

Reproducing bugs or running previous test cases accurately each time requires meticulous manual logging. Small environment changes can alter results.

4. Not Sustainable Long Term

With continuous code changes and new features, manual testing can become unsustainable. Test automation is better for long term cost effectiveness.

5. Limited Coverage

Given effort constraints, manual testing has limited test coverage compared to automation which can run much larger test suites.

6. Limited Scalability

Adding new test scenarios and test data requires continuously updating manual test cases. Automated tests are easier to scale across parameters.

7. Dependent on Skill

Manual testing is heavily dependent on individual tester skill. Automation provides consistent testing with pre-defined test handling.

8. Difficult Reporting

Consolidated tracking, analysis and reporting of manual test execution status, progress, results etc. across teams can become highly complex.

While manual testing cannot be eliminated, test automation strives to take over the repetitive, simple and long running test cases to optimize human tester time and effort.

When is Manual Testing a Good Choice?

While test automation is gaining widespread adoption, manual testing still has an essential role in many cases. Here are some scenarios where manual testing is a good choice:

1. New Software Systems

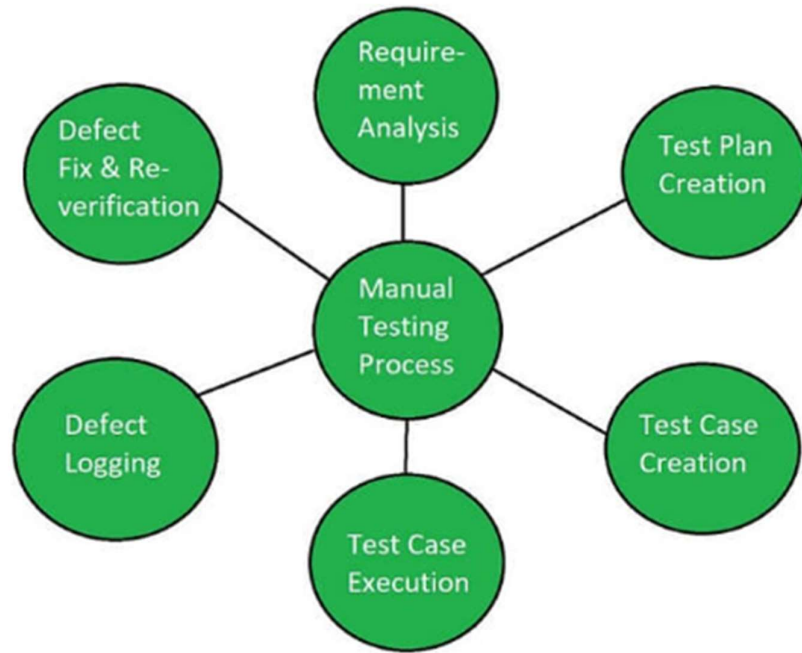
In new software systems with frequently evolving functionality and user interfaces, automation scripts require constant maintenance. Starting with manual testing allows stable system functionality before test automation.

2. User Experience Testing

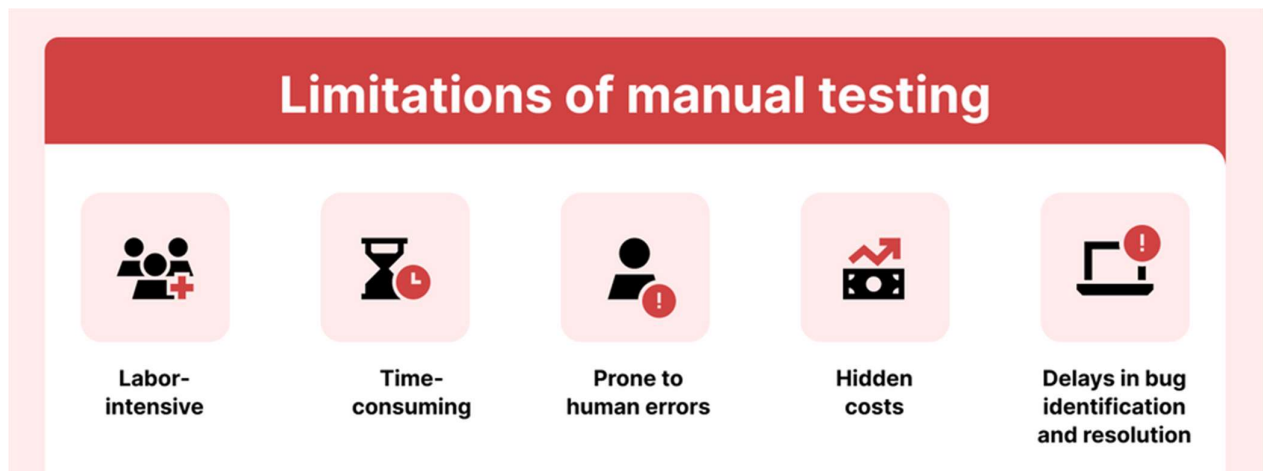
Subtle usability issues like confusing workflows, unintuitive navigation, visual inconsistencies etc. are best identified manually by emulating real users rather than test automation.

3. Exploratory / Ad hoc Testing

Free-form exploratory testing without restrictive test cases provides a creative approach to discover hard-to-anticipate defects based on intuition and domain experience.



3.1 Figure-1



3.1 Figure-2

3.2 ADVANTAGES OF AUTOMATION TESTING:

Automation testing is the offers numerous advantages that significantly enhance the efficiency, reliability

and quality of software development processes.

Automation testing is particularly advantageous for regression testing, where it can quickly verify that new code changes haven't introduced unintended side effects



3.2 Figure

Automation testing offers numerous advantages that significantly enhance the efficiency, reliability, and quality of software development processes. Firstly, automation testing boosts efficiency by executing repetitive test cases swiftly and accurately, saving valuable time and resources. With automation, teams can focus more on critical tasks such as designing complex test scenarios and analyzing results rather than spending time on mundane, manual testing procedures. Additionally, the accuracy of automation testing is unparalleled, as it minimizes the risk of human error, ensuring consistent test execution and reliable results across various environments.

Moreover, automation testing promotes reusability, allowing test scripts to be leveraged across different versions and iterations of software. This not only maximizes the value of test assets but also accelerates testing cycles and improves overall productivity. Furthermore, automation testing excels in achieving comprehensive test coverage by efficiently handling a wide range of scenarios, including edge cases and negative testing, which might be impractical to cover manually. By thoroughly testing different aspects of the software, automation helps identify defects early in the development lifecycle, reducing the cost and effort required for bug fixing.

3.3 LIMITATIONS OF AUTOMATION TESTING:

- Simple vs Complex Task handling. ...
- Test code & Backend Issues. ...
- Limitation by Object Identifiers. ...
- Updation in Agile environment. ...
- Automation is Expensive. ...
- Bugs & Automation. ...
- Needs more resources for Automation. ...
- Manual Testers get Users' Perspective Better.

Automation is quite fascinating since it reduces the manual efforts of the manual testing process. Automation indeed accelerates the whole testing process & generates faster results. But, can automation testing replace manual testing?

If the answer is a 'NO, it can't be replaced. Then that makes us curious enough to think of the most common limitations in automation testing. Well, if you're a total noob to the testing world, then you can get a basic idea of automation testing by reading this.

There are some notable limitations in automation testing which makes us understand that we need both manual & automation testing. So, if you're planning to enter into the automation testing industry, then instead of blindly advocating the automation technology, you need to understand the most common limitations in automation testing.

1. Simple vs Complex Task handling

Simpler & repeatable tasks are better handled through automation testing. While complex test cases are better through manual testing. This point is easily understandable, now let's dive deep.

2. Test code & Backend Issues

Most commonly, test automation fails when it encounters issues with test scripts or test frameworks. And, there is a high chance of an increase in issues of the test scripts when the backend team lacks the skillset to code efficiently. Almost 30% of all automation issues occur due to Backend issues, which are mainly caused by network, availability & data issues.

3. Limitation by Object Identifiers

When a developer designs web pages featuring multiple objects with the same ID, & when it is not conveyed to alter the test script, there is a sure-shot chance that similar objects cause issues in the test scripts for automation.

This will again need additional tools like perfecto scriptless & AI-powered technology for handling such situations.

4. Updation in Agile environment

Constant feedback, UI(User Interface) changes according to users' needs, & a new set of features in the online platform. Changes in all of these are followed by the changes in the testing scripts. This further means a lot of automation updation in agile.

5. Automation is Expensive

Especially, when it comes to small-scale projects, the price of automation software is high. Alongside, the maintenance & management of the scripts is pricey with additional cons such as script writing, rewriting & increased processing times. Furthermore, analysis is essential to see the failed automated test cases. Just count the manual labor and expenses for all of the above!!!

6. Bugs & Automation

Bugs are unexpected surprises. They can be found quicker when manually tested than in automation. Why? It is because automated testing can't notice errors that it wasn't programmed to find. Meaning that scenarios that cause bugs must be scripted for better automation testing.

7. Needs more resources for Automation

Yes! Automation indeed makes the testing faster but it also requires constant script-updating for all the dynamic changes that are to be seen in the product. Here, we need to understand that automation testing assists manual testing but doesn't replace it.

8. Manual Testers get Users' Perspective Better

Testers are also users in their personal life, they understand what users need better than a programmed test script. They add high value to the product with their personal knowledge to write better scripts. Or Automation will need AI technology to understand users' interests & needs. Anyhow, Automation is not independent.

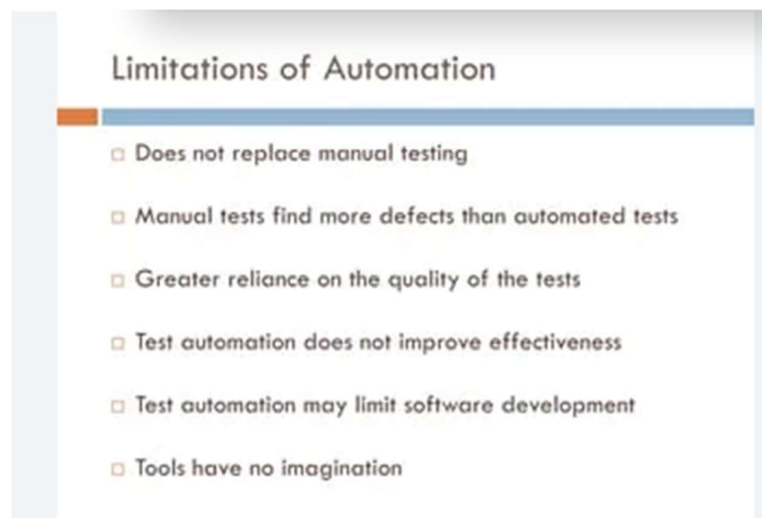
9. Complex with Mobile Devices

Automation testing poses challenges on the cost optimization factor in order to support various types of mobile devices, & operating systems like iOS, Android, and Windows. Sometimes, QA testing gets complex & costlier than the development of mobile apps.

10. Automation is just pass/fail

Considering web forms, now an automated script will be able to easily input values into a web page, but it can't double-check that the values will be saved if the user tries to navigate away & comes back later. Here, manual testers can notice beyond pass/fail & with the change in the abnormal/slow speed in the submission of inputs.

Many beginners who don't have a proper understanding of limitations in automation testing think of automating everything. Clearly, automation testing aids manual testing but can't dominate it.



3.3 Figure

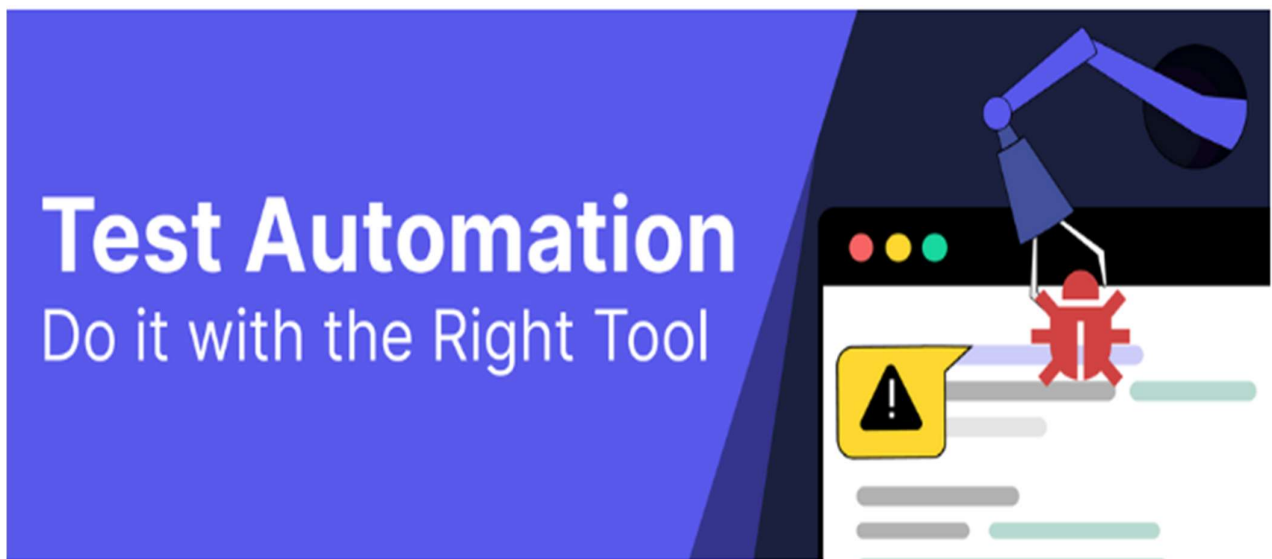
What can be the limitations of automating software testing?

- Designing a perfect automation script need some solid understanding of software development principles which automation testers lack most of the time. This is because they were not trained for development activities by organizations. Thus everyone follow their own standards while developing scripts.

- Where ever there is a change in UI, most the scripts need changes in their code and thus continuous maintenance.
- Many automation developers think they should develop a automated testing script which can handle all possible situations by itself without manual intervention. But by doing so, we will end up in developing another software which is again very difficult to design, develop and maintain. It is not at all suggested to develop such scripts. Every script should assume some preconditions and should proceed assuming all required.
- Not all tools support all UI objects or all kinds of testing. Few tools are specialized for UI testing and few for DB testing and few for web testing and few for performance/load testing.

3.4 SELECTING RIGHT AUTOMATION TOOL:

How to Select the Right Automation Testing Tool



3.4 Figure

Depending on your projects and your QA team's nature, your choice of automation testing tool must be compatible with many different aspects, such as project scope and requirements, in addition to its reputation. The best tool available does not guarantee the best testing outcome. It must be the right one. Selecting the right testing tool depends on various factors such as the type of application, testing objectives, budgets And team expertise. Could you provide more details about your project, so I can suggest a testing tool.

The Significance of Test Automation

A lot of controversy about whether manual testing has fallen behind as automation testing has risen in popularity. Regardless of contrary opinions, the importance of test automation nowadays is no doubt undeniable.

The software market now demands “Quality at Speed” from industry players. As the name implies, “Quality at Speed” means higher quality products must reach end-users but in a shorter time span than before. This difficult demand fueled test automation’s impressive growth and turned it into a game-changer by allowing QA teams to execute faster and more accurate test cases.

Test types that require repetitive actions, like regression testing, are what needed to be automated the most. Frequent changes in software will significantly escalate the total cost in terms of time and human resources used to run tests manually. Therefore, automating tests is the smarter and more efficient choice in such cases.

Regardless of these advantages, test automation does not work the same for all projects. While many QA teams have benefited from automation, other companies have wasted time, efforts, and financial resources in implementing automation tools.

Automation testing success primarily lies in identifying the right tool for different needs and demands. This process takes time and effort at first, but it is a must for your team to automate tests efficiently in the long run.

Types of Automation Testing Tools

All available test automation tools can be divided into three types in general as below.

1. Open-source automation tools

These tools are free platforms that allow users to access and use their source code. Users can opt to fully adopt the code or modify it to suit their testing needs. This type of tool is free of charge and developed by the community. Open-source tools are the top choice for many automation testers with a programming background due to their free access and the ability to customize advanced test cases.

2. Commercial automation tools

Commercial tools are produced to serve commercial purposes and are usually distributed via subscription plans. Users must purchase a paid license to use the software.

Compared with open-source software, this kind of tool usually has more premium features and thorough customer service that completes the whole testing process for companies or enterprises.

3.Custom framework

There are niche projects that a single open-source software or fixed commercial testing tool that cannot fulfill the requirements. They are mainly due to differences in their testing processes and testing environments. In such cases, teams need to develop customized software on their own. The custom framework is much more complicated than the two other solutions and can be deployed by technical experts.

Criteria for Automation Testing Tool Evaluation

Does your team possess the necessary skills to best utilize the tool?

Automation testing is much more technical than manual testing. In many automation tools, especially open-source software, testers must possess a sufficient level of programming knowledge to write and execute test scripts. This technical barrier appears to be the most challenging obstacle in adopting test automation.

Testing tools requiring no coding in execution have proven to be a promising solution to this bottleneck. You can find out more about top codeless automation testing tools for 2024 [here](#).

What is your team budget?

Here's a fact: Test automation is not affordable in many cases. However, it brings out a positive ROI for your team and business in the long run as long as the [budget is calculated thoroughly](#). Depending on the budget, it will be easier for you to pick the appropriate software, an open-source or commercial tool. We will talk about these types of tools further in the following part of this article.

What features to look for?

While the requirements vary from team to team, there are some key factors that you should always take in consideration when choosing a suitable automation tool.

A significant factor that escalates the total cost for test automation is script maintenance. Pre-written scripts in automation testing are fragile by nature. The ideal automation tool should come with capabilities to reduce such effort, such as [eliminating object locator flakiness](#). On the other hand, script reusability saves you and the team a great deal of time for similar test cases as you can reuse test scripts.

CHAPTER-4

AUTOMATION TOOLS

Automation testing is using software to test another software. If applied correctly, the automation strategy helps reduce costs and improve the overall engineering and testing culture. Playback tools can also streamline the process, although it's essential to understand the underlying mechanisms.

However, when choosing the right automation solution, it is easy to get lost in the ocean of modern QA automation software on the market. Here, we have selected some of the top automation testing tools that could help take over the time- and labor-consuming tasks and boost the general performance and functionality of the product.

The most popular languages used for automation testing include Java, JavaScript, Ruby, C#, Python, and PHP. Within the PHP ecosystem, tools like Codeception not only support web application testing but also provide capabilities for writing unit, functional, and acceptance tests. Additionally, familiarity with platforms and frameworks like .NET can be valuable. Moreover, tools today cater to a broad audience, with some allowing non-coders or even offering codeless solutions to deal with test creation.

This question is about two important things: how well the tool you work with inscribes itself into the processes of continuous development and integration, and how well it functions together with programmers' tests. Thus, for example, it is really helpful to know if the tool offers version control and data history if necessary.

Open-Source vs. Commercial QA Automation Software

Another criterion to consider is the software's value. Free automation tools like Selenium or Protractor are quite popular in the community of testers.

There are a lot of benefits to using open-source software, such as better pricing for licensing and the ability to create tailored solutions or extended collaboration.

However, the drawbacks of using open-source automation tools could be quite significant.

Security is the most important one. Thus, open-source QA automation tools might not be the right fit for projects that pay close attention to security parameters, scalability, robust features, and dedicated support. In this case, an enterprise-grade tool would be a better fit. Other disadvantages of open-source tools include time-consuming setup, limited feature availability, and non-specific or incomprehensible reporting.

A free automation testing solution can benefit individual testers and small or middle-sized businesses. Still, it's a good idea to remember that commercial software is big business-oriented, secure, and if wisely chosen, has a full stack of features necessary for the target testing in question.

Comparative Descriptions of Best QA Automation Tools

Selenium



4 Figure

Selenium is still a number one choice among automation testing tools for web applications.

It's a powerful drive for cross-browser testing and can be used for many test types, including compatibility, integration, smoke, sanity, end-to-end, and regression testing. It is compatible with all major browsers (Chrome, Firefox, etc.) and platforms, offering cross-platform capabilities, which is crucial in diverse device environments. One could say that Selenium, consisting of WebDriver, IDE, and Grid, is not a single tool but a toolbox for web browser automation.

Selenium supports the creation of detailed test scripts, enhancing the workflow for a more organized testing process. It is license-free. However, the question of value is a bit tricky with Selenium as, being initially an open-source tool, it might require a workforce to set and maintain it. Here is a comparative chart representing Selenium's pros and cons:

Pros	Cons
<ul style="list-style-type: none">-extremely easy to use and employ Selenium at the GUI level-open source-multiple language support (all major programming languages)-saving and re-running scripts-any browser + any platform-effective integration-can execute tests while the browser is minimized-parallel testing (stressing different parts of the app simultaneously)-IDE is suitable for juniors and novices in QA automation testing	<ul style="list-style-type: none">-demands a lot of technical knowledge on the part of the tester as well as knowledge about using third-party tools-not self-sufficient (requires the support of third party frameworks. For example, Sikuli to test images)-no native support to generate bug reports-requires constant updates, improvements, and maintenance of the complete ecosystem-test-automation is always an isolated process.

Selenium. Selenium is one of the most popular open-source test automation frameworks. It provides a wide range of tools and libraries for automating web applications.

4.1 FUNCTIONLITY TESTING TOOLS:

Functional testing tools automate the process of validating the *functional* behaviors of systems under test; their primary purpose is different in the scope of software quality as compared to tests designed to evaluate systems for performance, usability or behaviors under high load/volume.

Given a specific requirement or acceptance criteria, functional testing checks if a critical feature is behaving as expected. While a load test tests how long a login form processes user inputs before a successful/unsuccessful login attempt, a functional test for the same login form verifies the following behaviors:

- Can a user successfully log in with a valid email and password?
- Can a user not log in with an invalid email and password?
- Does a pop-up message display when an invalid email is entered?
- Can API requests pass and store user credentials for future logins?
- Does the sign-in link lead to the user profile page?
- Are all objects visible on the screen: username, password, sign-in button, remember me checkbox, forgot password link and create an account link?
- Is there a maximum/minimum length for the username and password?
- Is the UI responsive, rendering in the right layout on the different operating systems, browsers and device versions?

1. Open-source functional testing tools, libraries and frameworks

The first option for your team is selecting a suitable open-source testing library to build a testing framework. This will be the core library, forming the foundation of your framework and providing all of the essential features needed for testing. The aim is to create higher levels of abstraction around the core library's functionality so that the framework is more user-friendly and versatile.

The brick and cement you use to build frameworks are:

- Libraries of functions (e.g., Selenium, Playwright, Appium, Rest Assured)
- Browser drivers (e.g., geckodriver, chromium)
- Design patterns (e.g., Page Object Model, Screenplay, Fluent)
- Coding standards (KISS, DRY, camelCasing)
- Test artifact management structures (e.g., object repositories, helper utilities)
- Test reports and execution logs (plugins, structure)

Likewise, the way that you author, run, analyze and maintain tests will require some time getting used to. If you're opting to build your own testing framework using open-source libraries, you must be well-equipped with extensive coding and testing knowledge. You need to know how to connect all of the "brick and cement" listed above together to form a framework that does simplify your testing activities.

While it's an option, implementing your own test framework takes up a lot of setup/build time before the testing part actually begins. However, the reward is that you get an extremely high level of customization with your testing framework, compared to using an off-the-shelf solution built by another vendor.

This option is also usually free (the only cost is your own time and your expertise), so it is ideal for budget-conscious projects. Sometimes the framework development part is not the most difficult part. The

maintenance of the framework and resolving issues encountered at runtime of larger test suites is what creates a lengthy and cumbersome testing cycle.

2. Single-point commercial automation testing tools

These are commercial tools i.e. you have to shop around for a suitable one and invest a certain amount in order to get access to the features. As their name suggests, these tools only have to meet a single testing purpose, and we can categorize them based on the purpose they serve:

- System-under-test (web, mobile, API or desktop testing)
- Utility (design/run/get reports)
- Testing types (UI and API testing)

For example, Postman's focal point is solely on API performance, security and functionality. If APIs are at the top of your testing list, give it a go. While it is indeed nice to have a tool solely dedicated to a part of your functional testing, there will come a point when you want to have a more comprehensive testing experience. After all, the dedicated nature of these tools also limit their potential for expansion and scaling.

4.2 PERFORMANCE TESTING TOOLS:

Performance testing is an indispensable part of the modern development process. It helps us to determine a system's behavior under normal and unusual load conditions.

Various performance testing tools are used to test the speed, responsiveness, and stability of applications under real-world conditions. By simulating real-world usage scenarios, these tools can help developers identify.

Choosing the best performance testing tools to simulate load and execute our scripts is not an easy task. Among many open source and paid solutions, we need to consider what best suits our client's business needs. In this article, we list tools for performance tests that we work with and have experience with.

What is performance testing?

Performance testing is an umbrella term covering all testing activities focused on the performance (responsiveness) of a system under different loads.

For clarity, it's best to separate the tools for:

Backend (load testing)

- Frontend testing (client side)
- Mobile testing

In this post, we cover load testing and client-side testing tools. We have intentionally left out tools specifically for mobile performance testing. We will tackle this in a separate article, thus paying it due attention.

What's on the menu: Performance testing tools

There are a variety of performance testing tools available, each with its own strengths and weaknesses. The most important factor in choosing a performance testing tool is to select one that will best meet the needs of your particular project.

Load performance testing tools

Load performance testing tools help to ensure that your site can handle sudden spikes in traffic without crashing or suffering from lags or other issues. Let's check which load testing tool can come in handy to help you keep your site running smoothly and efficiently.

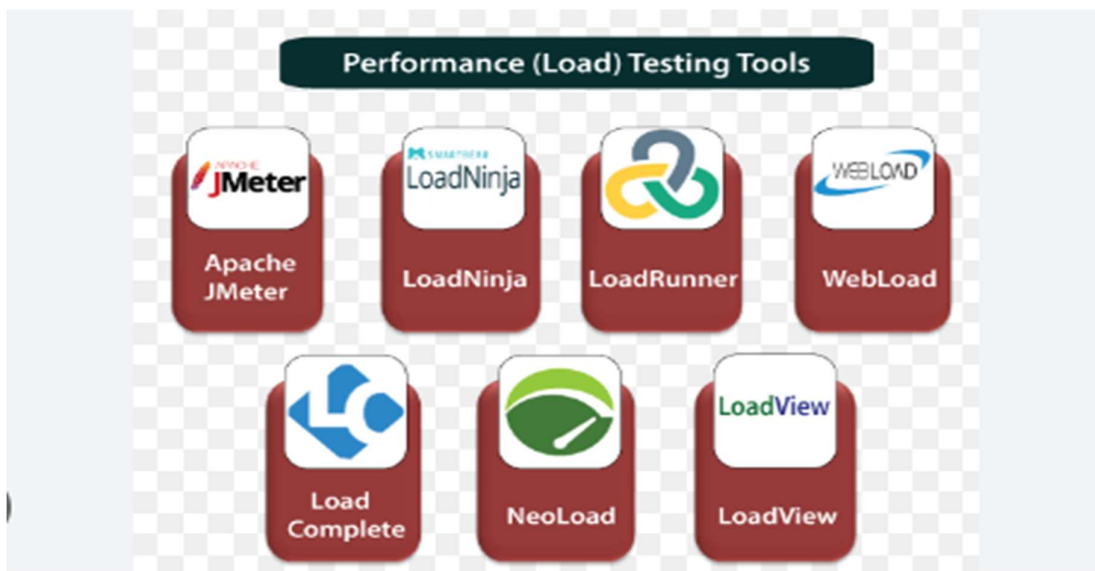
JMeter

JMeter is definitely the most popular open source load testing tool. This multithreading framework allows performance and load testing both on static and dynamic resources and dynamic web applications. It can be used to simulate a heavy load on a server, group of servers, and a network. It helps to analyze the performance under different load types. It supports a vast variety of protocols like: HTTP, HTTPS (Java, NodeJS, PHP, ASP.NET, ...), SOAP / REST Web Services, FTP, LDAP, SMTP(S), POP3(S) and IMAP(S), and TCP.

Advantages:

- Easy to use without extensive knowledge of programming, as it has a user interface (UI) and it is also possible to use a command-line interface (CLI);

- This tool has good reporting and data analysis capabilities. It is possible to generate (and, if necessary, customize) a report after a test to help to analyze results. During the test, we can only see a console dump;
- It's extendible and you can use a Plugin Manager;
- It can be used for load tests for different kinds of applications. As an example:
It allows API, Database, and MQ testing;
- Easy installation;
- It supports distributed load tests;
- Community support



4.2 Figure

Python Performance Testing with Granulate [Granulate's code profiler](#) is absolutely free and open-source (unlike most other tools on the market), and extremely easy to install and start using. Also, its user interface is intuitive and flexible, letting you select different time periods, filter processes, etc.

Load performance testing tools help to ensure that your site can handle sudden spikes in traffic without crashing or suffering from lags or other issues. Let's check which load testing tool can come in handy to help you keep your site running smoothly and efficiently.

4.3 SECURITY TESTING TOOLS:

Security testing tools are software used to assess and improve the security of computer systems, networks, and applications. These tools perform various tasks such as identifying vulnerabilities, simulating cyber-attacks, analyzing code for security flaws, and ensuring compliance with security standards. They play a crucial role in the proactive detection and mitigation of potential security threats.

The benefits and uses of security testing tools include enhancing the overall security posture of digital environments, protecting against unauthorized access and data breaches. They help in identifying and addressing security vulnerabilities early, reducing the risk of cyber-attacks. By ensuring compliance with security regulations, these tools build trust and credibility among users and stakeholders. They are essential in today's digital landscape, where maintaining robust security practices is vital for protecting sensitive information and systems.

New Relic is a performance monitoring and management platform that helps you keep an eye on your applications, infrastructure, and customer experience. It's designed to help you identify and fix issues before they become a problem, which is pretty awesome if you ask me.

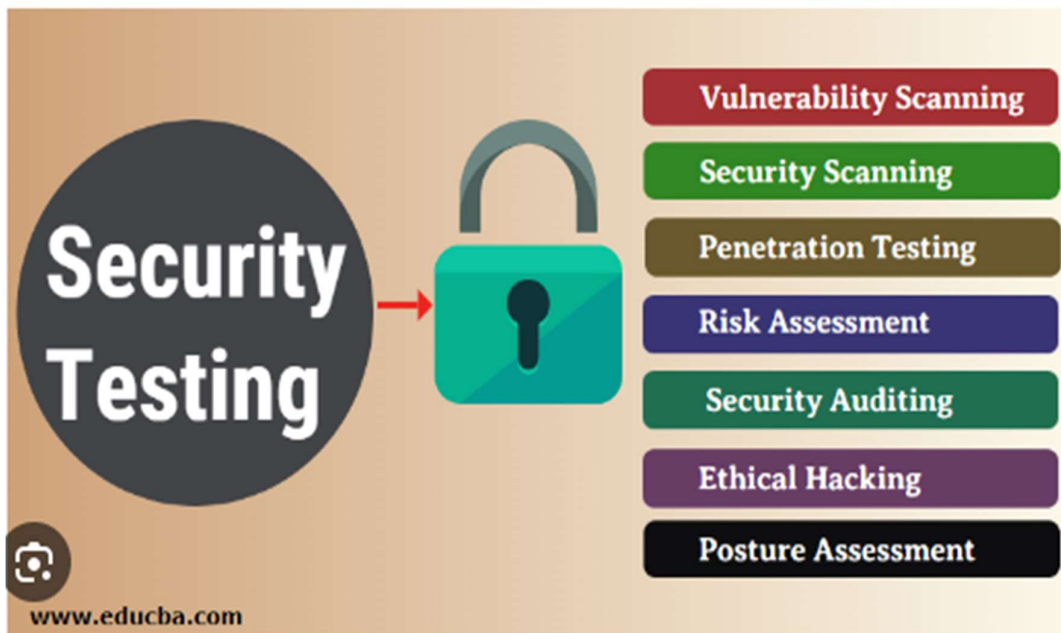
I was impressed by how New Relic combines performance monitoring with security testing. It's not just about making sure your app is running smoothly; it's also about making sure it's secure. And for that, the platform connects with over 500 tools, which allow you to import telemetry from almost any tool you currently use.

Now, let's talk about some standout features that make New Relic different from other tools in the market. Here I would like to point out Grok, an AI assistant that can read your telemetry and identify outliers for you. Not only that, but you can ask questions to it and it will provide with potential code changes and find a root cause for an issue.

As mentioned earlier, a thing that sets New Relic apart is its integrations. It works seamlessly with a wide range of popular tools and platforms, like AWS, Azure, Google Cloud, and more. This means you can easily incorporate it into your existing workflow and get the most out of your security testing efforts. If, however, you cannot find a pre-built integration with a tool you currently use, you can use its API to custom-build it.



4.3 Figure-1



4.3 Figure-2

4.4 LICENSED TOOLS:

- Python is the most popular language when it comes to automation testing. It's also one of the easiest languages to learn because it's based on a simple, easy-to-understand syntax.
- Python is an interpreted programming language and it runs on Linux, MacOSX, Microsoft Windows, BSD, Solaris and other operating systems.
- There are many advantages of using Python for automation testing. Some of them include:
- It's easy to write tests with Python as compared to other languages like Java or C++.
- Python can be used for integration testing with other applications like web browsers or databases.
- The code of Python is easy to understand because it follows a simple syntax which makes it easier for developers to read and understand the code written by them.
- Python has an active community which provides support and resources for its users at any time they need them.
- Python is a high-level, general-purpose, and interactive programming language. It's designed to be easy to read and write, yet powerful.
- Python has become one of the most popular languages for data science and machine learning. Python can be used for web applications, desktop apps, and mobile apps.
- Python is easy to learn and use, but you can't just pick up Python if you don't know any other programming languages – it requires a strong foundation of understanding of how computer systems work. As such, it has many similarities with other programming languages like Java or C++ but also has unique features that make it stand out from the crowd.

Here are the very best Python automation testing tools and frameworks as of 2024:

PYTEST:

Pytest is a powerful testing framework for Python. It's one of the top choices among developers because it's simple to use and flexible enough to accommodate any kind of test scenario.

It features an easy-to-learn syntax that allows you to write test cases quickly without having to become a coding expert. And because it has no dependencies, it can run everywhere.

It also has built-in support for mocking objects so that tests can be run even if they have access to objects outside of their scope. This makes writing tests faster since you won't have to create stubs or mocks for every object used in your code base.



4.4 Figure-1

Pytest is a mature, full-featured testing framework for Python. It's used for unit testing, functional testing and regression testing. You can use it with any programming language that supports the standard library (Python 2 or 3).

Pytest has two main components: test runners and frameworks. The test runner runs tests while they're being written; it gives you access to inputs and outputs from your code as well as information about how long each test takes to run. The framework supplies various tools for writing better tests: fixtures, results collection mechanisms and so on.

2. Robot Framework



4.4 Figure-2

Robot Framework is a Python-based test automation framework. It's designed to be a flexible, high-level testing framework that supports keyword-driven testing. In this approach, you write your tests in terms of test cases and suites with keywords as the driving force behind what happens when those tests are run.

Robot Framework was created by James Edward Gray II (who also wrote Selenium) and has been around since 2006! This means it has been around long enough to have seen significant development over time which means its features are often updated without requiring any major changes to the existing codebase(s).

3.BEHAVE

Behave is a BDD (Behaviour Driven Development) framework for Python. It's based on Gherkin, the business readable language for specifying acceptance criteria.

Behave uses the cucumber-puppeteer library to run the tests in a headless browser, meaning you can run your automated tests without having an actual device or operating system installed on your computer. This makes it easier to test web applications that have been deployed on multiple platforms and locations because no matter which platform or location you're testing from, Behave will give you results based on what behaviour was expected along with any errors that were encountered during execution of that behaviour.

4.LETTUCE

Lettuce is a Python testing tool that provides a higher-level API for writing tests. It provides a clean and simple testing API, with features such as automatic mocking, test templates and more. Lettuce is also built on top of the Behaviour Driven Development (BDD) framework to help make developing tests easier by providing clients with common BDD assertions like `@given` or `@when`.

Lettuce supports many popular Testing Frameworks like pytest, nose and unittest2 but it can be easily integrated with any workflow you prefer through plugins like pytest-plugin-lint, pylint , tox or cProfile.

CHAPTER-5

SELENIUM WEBDRIVER

Selenium WebDriver is an open source tool to perform Browser Automation on real browsers. WebDriver communicates with browsers directly using client libraries and JSON wire protocol. It helps testers ensure that the website functions as intended on different browsers.

Selenium WebDriver is an enhanced version of Selenium RC. It was introduced in the market to overcome the limitation faced in Selenium RC. Though it is an advanced version of RC, its architecture is completely different from that of RC. Just like Selenium RC, Selenium WebDriver too supports multiple programming platforms to provide wider flexibility and requires knowing any one programming language.

- **Selenium Grid**

Selenium Grid is a tool that is used for concurrent execution of test cases on different browsers, machines, and operating systems simultaneously. This tool makes Cross-browser compatibility testing very easy. There are two versions of the Selenium Grid – the older version is known as Grid 1 and the recent version is known as Grid 2.

Now let's move on to the tutorial on **Selenium WebDriver**.

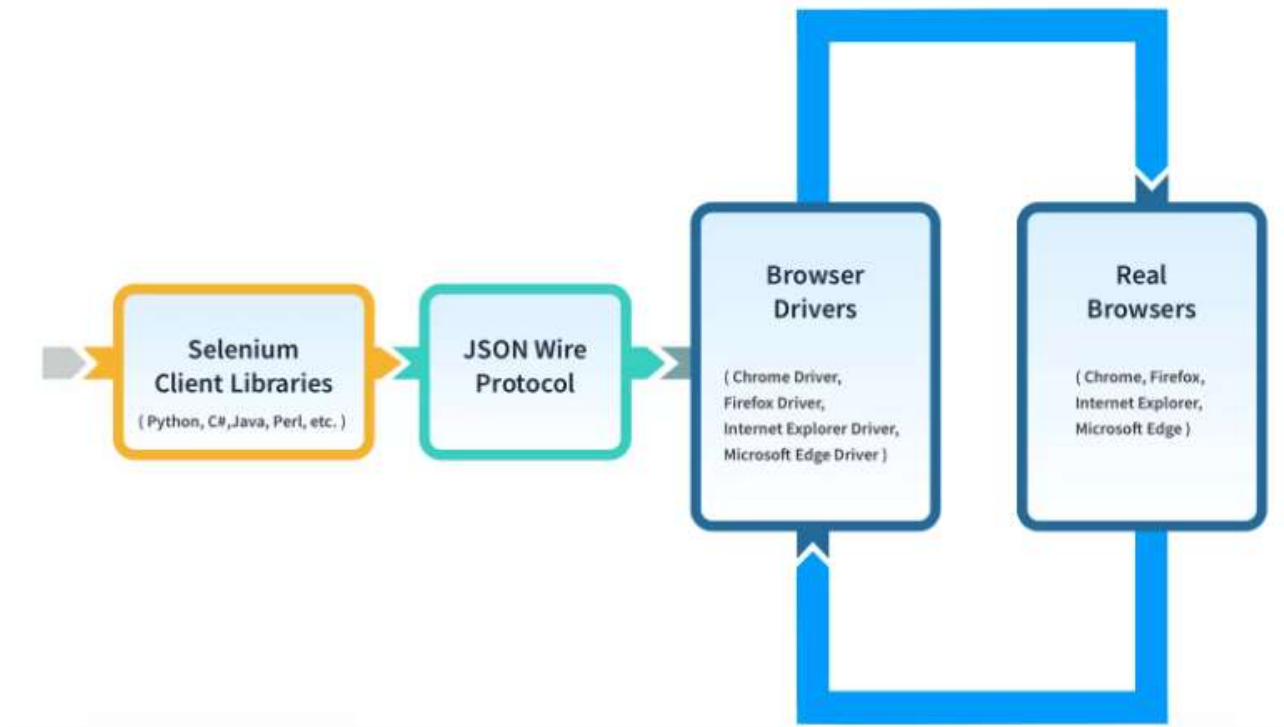
What is Selenium WebDriver?

Selenium WebDriver is a web framework that permits you to execute cross-browser tests. This tool is used for automating web-based application testing to verify that it performs expectedly.

Selenium WebDriver allows you to choose a programming language to create test scripts. As discussed earlier, it is an advancement over Selenium RC to overcome a few limitations. Selenium WebDriver is not capable of handling window components, but this drawback can be overcome by using tools like Sikuli, Auto IT, etc.

Selenium Client Libraries/Language Bindings

Selenium provides support to multiple libraries such as Ruby, Python, Java, etc as language bindings have been developed by Selenium developers to provide compatibility for multiple languages. For instance, if you want to use the browser driver in Python, use the Python Bindings. You can download all the supported language bindings of your choice from the official site of Selenium.



5 Figure

5.1 WHAT IS SELENIUM:

What Is Selenium?

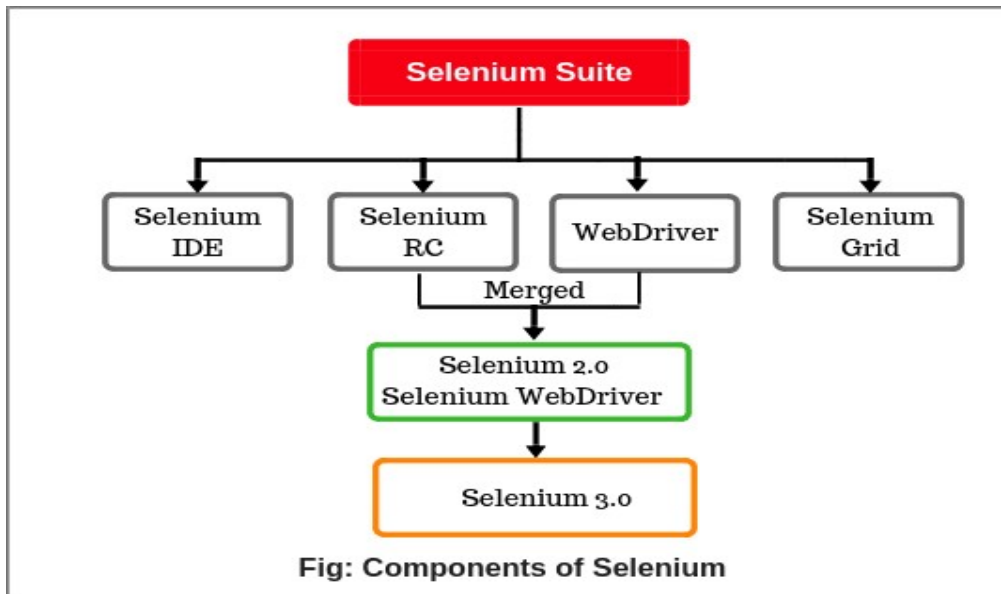
Selenium came into existence in 2004 at ThoughtWorks to test a web application named Time and Expenses by Jason Huggins. The tool was developed to test the front-end behavior of an application in various browsers. The tool was popular and open source.

The increase in demand for automated testing led to the development of several versions of Selenium over the years, which are discussed next.

which took many hours. The testing tended to rely on different scenarios. Each scenario was considered a test case to enact the behavior of the web app before its implementation. These test cases were deployed on various browsers to affirm any issues in the source code.

requires a dedicated team of testers to check all test cases. Accuracy and time are major constraints in web development.

Selenium Tools and Versions



5.1 Figure-1

Any one (or more) of these Selenium tools can be used by an organization; the choice depends on the test environment's requirements. The first tool developed by ThoughtWorks was Selenium Core in 2004. It allowed the tester to write their own code/ script to automate front-end interactions like keyboard or mouse activities. These activities were similar to a user interacting with the application.

Selenium is a powerful tool for controlling web browsers through programs and performing browser automation. It is functional for all browsers, works on all major OS and its scripts are written in various languages i.e Python, Java, C#, etc, we will be working with Python. Selenium Tutorial covers all topics such as – WebDriver, WebElement, Unit Testing with selenium. This Python Selenium Tutorial covers Selenium from basics to advanced and professional uses.

Selenium is a popular open source framework for automated testing of web applications across multiple browsers, operating systems, and devices. Selenium makes it easier for developers to create automated tests for web applications without having to worry about the underlying technology.

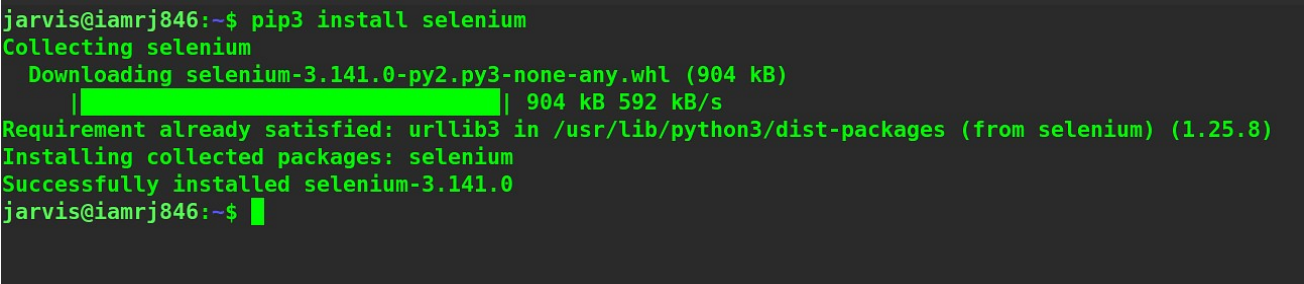
There are multiple reasons as to why Selenium is way more popular and powerful when compared to other web-automation tools in the market. Here are the reasons why:

1. It is an open-source, free-to-use, portable tool that is used to perform web-testing in a seamless manner.
2. It is a combination of several Domain Specific Language (DSL) and other tools which allow you to perform various types of testing.
3. It is quite easy to understand, intuitive, and has a low learning curve. The commands are categorized as multiple classes, making them easier to understand.
4. It takes off a load of the burden from testers, since the amount of time that is required to perform testing, especially on repeated test cases reduces drastically.
5. There is a significant reduction in the costs incurred to the business clients, which is a win-win situation.

How to Install the Package for Using Selenium With Python?

The first step is to install the Selenium package for Python. You can do so using the simple pip command.

\$ pip install selenium

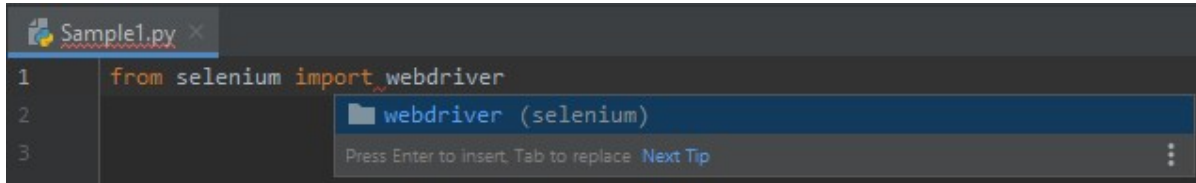


```
jarvis@iamrj846:~$ pip3 install selenium
Collecting selenium
  Downloading selenium-3.141.0-py2.py3-none-any.whl (904 kB)
    |████████████████████| 904 kB 592 kB/s
Requirement already satisfied: urllib3 in /usr/lib/python3/dist-packages (from selenium) (1.25.8)
Installing collected packages: selenium
Successfully installed selenium-3.141.0
jarvis@iamrj846:~$
```

5.1 Figure-2

Step1:In the first step, we will type the following statement to import the web driver:

1. **from** selenium **import** webdriver

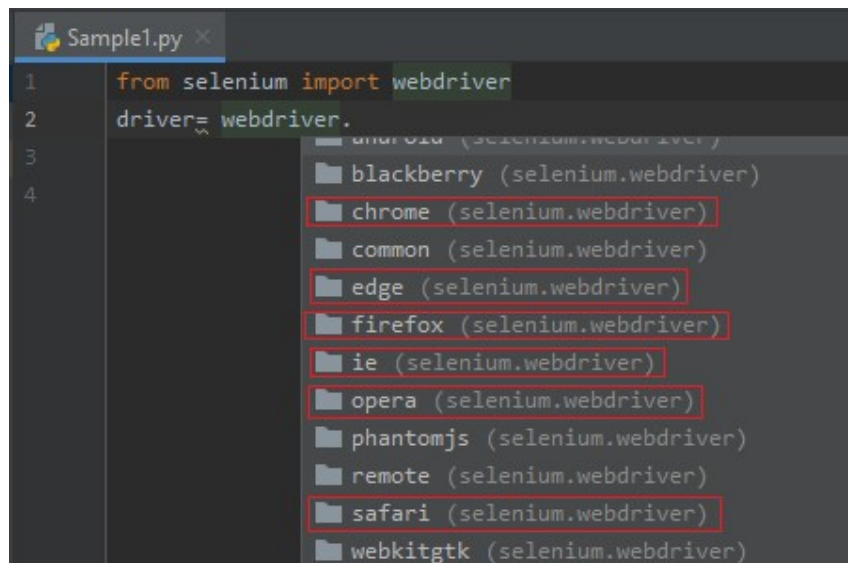


5.1 Figure-3

Step2

After that, we will open the Google Chrome browser.

As we can see in the below screenshot, we have multiple types of browsers options available, and we can select any browser from the list like **Chrome, Edge, firefox, Internet Explorer, opera, safari, etc.**



5.1 Figure-4

Following are the sample code for opening the Google Chrome browser:

1. driver = webdriver.Chrome()

Step3

In the next step, we will be maximizing our browser window size, and the sample code is as below:

1. `driver.maximize_window()`

Step4

Then, we will navigate to the given URL.

The sample code is as below:

1. `driver.get("https://www.google.com/")`

Note: As we know that Python is a very easy language to write code because we don't have to write multiple statements like as we did it java. Or if we want to comment out something, we just simply put a hash[#] in our statements, or we can directly press Ctrl+ Forward slash [/] from our keyboard.

5.2 WHY USE SELENIUM

11 Reasons To Use Selenium for Automation Testing

When we talk about automation testing, Selenium always finds its way to the conversation. Being one of the best tools for automation testing, Selenium is well-loved by developers and testers across the globe. But the question that remains is that there are many automation testing tools available in the market that delivers almost the same results. So, why only Selenium?

Well, this is something that we are going to discuss in detail in the article. By the end of the article, you will be able to list out all the benefits of performing automation testing using Selenium and why is it preferred over other automation testing tools.

Advantages of Using Selenium for Automated Testing

1. Language and Framework Support

When someone chooses a tool the first thing that comes to mind is: "Does my tool supports the language that I know?"

Well, this is not the case with Selenium as it supports all major languages like Java, Python, JavaScript, C#, Ruby, and Perl programming languages for software test automation.

You can write your scripts in any of these programming languages and Selenium converts it into Selenium compatible codes in no time. So, there is no need for knowing Selenium only languages. Also,

You can write your scripts in any of these programming languages and Selenium converts it into Selenium compatible codes in no time. So, there is no need for knowing Selenium only languages. Also, every Selenium supported language has dedicated frameworks which help in writing test script for Selenium test automation. So, when you go for Selenium as a tool for performing automation testing, you don't have to worry about language and framework support as Selenium does that for you!

2. Open Source Availability

One of the many things that adds to the advantages of Selenium is its open source availability. So, being an open source tool, Selenium is a publicly accessible automation framework and is free, with no upfront costs. So, you can save bucks here and use them for other good causes.

The Selenium community is continuously helping developers and software engineers in automating the web browser features and functionalities. Selenium being open source also helps you customize the code for better code management and enhance the functionality of predefined functions and classes. Selenium has become the most reliable web automation tool because of the ease of generating test scripts to validate functionality.

3. Multi-Browser Support

“One Selenium script for all browsers” is what the Selenium community has been working on and improvising every day. As per [StatCounter](#), Chrome, Firefox, Safari, Internet Explorer, Opera, and Edge browsers are the most used browsers worldwide and Selenium script is compatible with all the mentioned browsers. You don't need to rewrite scripts for every browser, just one script for all browsers.

4. Support Across Various Operating Systems

Different people use different operating systems and it is necessary that your automation tool supports all of them. Selenium is yet a highly portable tool that supports and can work across different operating systems like Windows, Linux, Mac OS, UNIX, etc.

You can create Selenium test suites over any platform like Windows and can execute the same test suite on another platform, for example, Mac or Linux. This enables developers and software testers to easily write test automation scripts without laying much emphasis on the platform on which it will run.

5. Ease Of Implementation

Selenium automation framework is very easy-to-use tool. Selenium provides a user-friendly interface that helps create and execute test scripts easily and effectively. You can also watch while tests are running.

You can analyze detailed reports of Selenium tests and take follow-up actions.

And finally, you will never feel alone. A huge Selenium community is always available to help you in case of need. You can ask your queries and perform brainstorming in the community.

6. Reusability and Integrations

As mentioned earlier, Selenium automation test suites are reusable and can be tested across multiple browsers and operating systems. However, the twist is if that Selenium is not an all-inclusive web automation testing tool. Hence, it needs third-party frameworks and add-ons to broaden the scope of testing.

For example, you need to integrate Selenium with TestNG and JUnit for managing test cases and generating reports. For achieving continuous testing, you'll need to integrate it with some CI/CD tools like Jenkins, Maven, and Docker. Also, for performing image-based testing, you need to integrate Selenium with tools like Sikuli, and for performing cross-browser testing with cloud-grid. You can integrate Selenium with almost all management tools.

7. Flexibility

Test management is what which is very important in testing lifecycle. It becomes easier and more efficient with Selenium features like regrouping and refactoring of test cases. This helps developers and testers in quick changes to the code, reducing duplication, minimizing complications and improving maintainability. These features make Selenium more flexible and usable as compared to other automation testing tools and hence helps Selenium to keep an edge.

8. Parallel Test Execution and Faster Go-to-Market

The main aim of automated testing is to save time and efforts. With the help of Selenium Grid, we can execute multiple tests in parallel, hence reducing the test execution time. With the help of cloud-grids for cross-browser testing you can test across as many as hundreds of browsers in parallel using Selenium hence saving you time in multiples of hundreds.

9. Less Hardware Usage

If you compare Selenium with other vendor focused automation tools like QTP, UFT, SilkTest, you will find that Selenium requires less hardware as compared to other testing tools.

10. Easy to Learn and Use

Selenium scripts are not something like writing hundred-page complex algorithm. Writing Selenium scripts is not more than writing a few pieces of codes to automate functionalities of your website. Also, documentation on the [Selenium website](#) is very helpful for developer and testers to start with Selenium automation testing. With the radically growing community, Selenium tutorials, testing, and development support is just a Google search away.

Also with Selenium IDE extension on Firefox browser, you can use record and play functionality to generate Selenium scripts for future reference.

11. Constant Updates

As Selenium is supported by a community and we all know that an active community doesn't like to stay stagnant, the Selenium community is also constantly releasing constant updates and upgrades. The best part about having a community is that these upgrades are readily available and easy to understand hence you do not need any specific training. This makes Selenium resourceful as compared to other tools and cost-effective as well.

What Is Selenium?

Before we delve deep in the benefits of let us first understand what Selenium is and why it is used.

Well, Selenium is an open-source automation testing tool which is used for automating tests carried out on different web-browsers.

It has a suite of tools which caters to different needs of organizations. It basically has four different tools:

1. Selenium RC (which is now deprecated)
2. Selenium IDE (Selenium Integrated Development Environment)
3. Selenium Grid
4. Selenium WebDriver

Why Is Selenium Used?

Selenium is basically used to automate the testing across various web browsers. It supports various browsers like Chrome, Mozilla, Firefox, Safari, and IE, and you can very easily automate browser testing across these browsers using Selenium WebDriver.

You can see live automated tests being performed on your computer screen. But the question that we'll be answering in this article still stands as there are many tools available for automation testing.

5.3 LOCATION ELEMENTS

In Python, for automation testing, you can use several libraries and modules to interact with elements on a web page or in a GUI application. Some commonly used ones include:

Selenium: It's widely used for web automation testing. You can locate elements using methods like `find_element_by_id`, `find_element_by_xpath`, `find_element_by_css_selector`, etc.

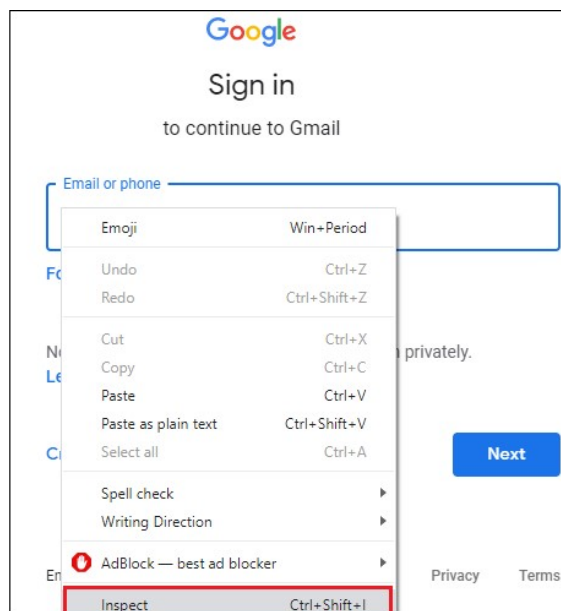
5.3.1 ID

EXAMPLE

Once we navigate to the URL of the Gmail application, we will identify the username text box and passing the value of it.

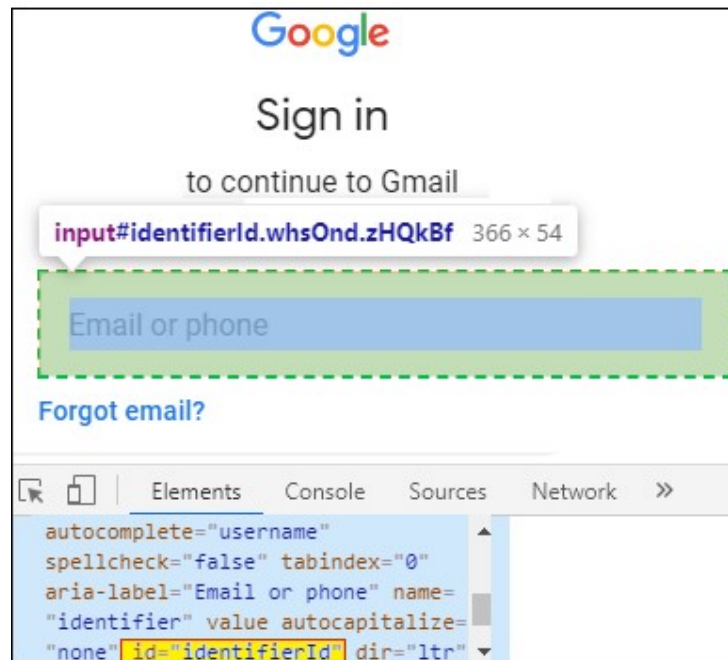
To identify the username text box, follow the below process:

- Right-click on the username text box.
- And select the **Inspect** option in the given pop-up menu as we can see in the below screenshot:



5.3.1 Figure-1

- The developer tool window will open with all the specific codes used in the development of the **username** text box.
- Then, copy the value of its **id attribute** that is: **identifierId** as we can see in the below image:



5.3.1 Figure-2

- And, here the sample code:

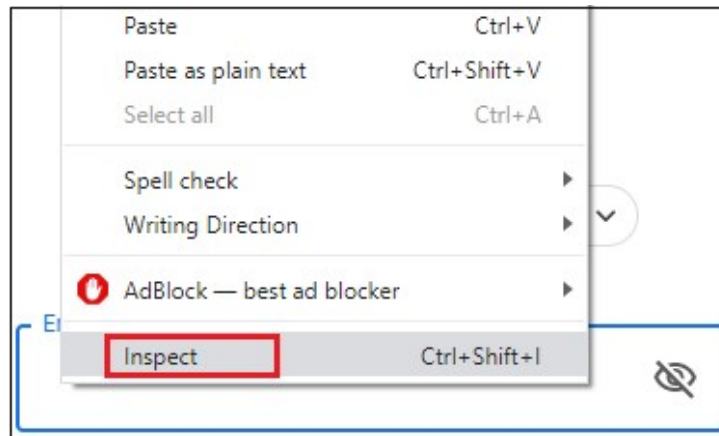
```
#identify the user name text box and enter the value  
driver.find_element_by_id("identifierId").send_keys("xyz11@gmail.com")  
time.sleep(2)
```

5.3.2 NAME

EXAMPLE

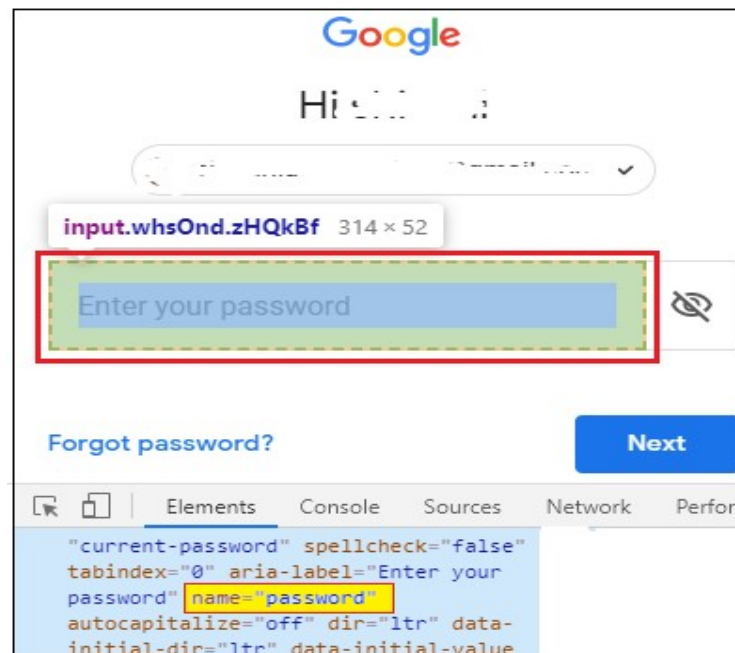
To identify the password textbox, follow the below process:

- Right-click on the **password text box**, and click on the **Inspect** Option from the given pop-up menu as we can see in the below screenshot:



5.3.2 Figure-1

- The developer tool window will open with all the specific codes used in the development of the **password** text box.
- And, copy the value of **name** attribute, i.e., **password** as we can see in the below image:



5.3.2 Figure-2

Here the sample code:

Identify the password text box and enter the value

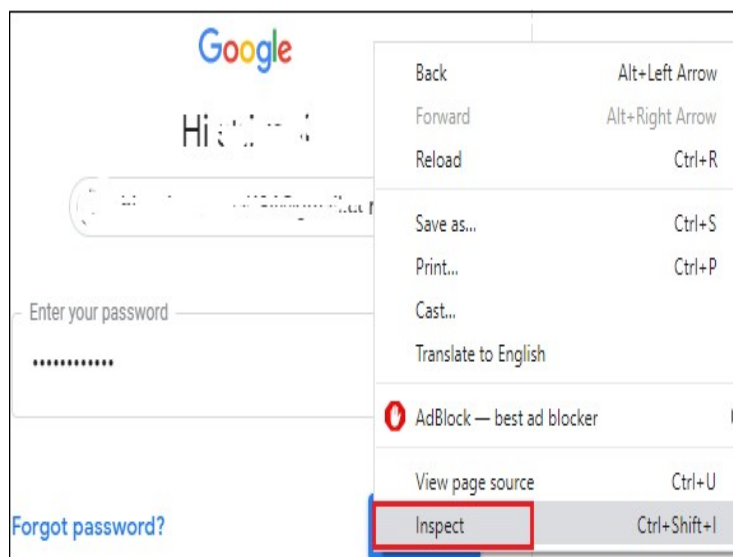
```
driver.find_element_by_name("password").send_keys("#####")  
time.sleep(3)
```

5.3.3 BY XPATH

(EXAMPLE)

To identify the Next button, follow the below process:

- Right-click on the next button, and click on the **Inspect** option in the given pop-up menu as we can see in the below image:



5.3.3 Figure-1

- The developer tool window will open with all the specific codes used in the development of the **Next**
- Copy the value of its **absolute XPath** that is: `//span[contains(text(),'Next')]` from
- the **chxpath** section as we can see in the below image:

- **chropath** section as we can see in the below image:



5.3.3 Figure-2

Here the sample code:

```
#click on the next button
driver.find_element_by_xpath("//span[contains(text(),'Next')][1]").click()
time.sleep(3)
```

CODE FOR USING ID, NAME, XPATH:

```
from Selenium import webdriver
import time
from Selenium.webdriver.common.keys import Keys
print("test case started")
#open Google Chrome browser
driver = webdriver.Chrome()
#maximize the window size
driver.maximize_window()
#delete the cookies
driver.delete_all_cookies()
#navigate to the url
```



```
driver.get("https://www.gmail.com")
#identify the user name text box and enter the value
driver.find_element_by_id("identifierId").send_keys("xyz11@gmail.com")
time.sleep(2)
#click on the next button
driver.find_element_by_xpath("//span[@class='RveJvd snByac']][1]").click()
time.sleep(3)
#identify the password text box and enter the value
driver.find_element_by_name("password").send_keys("#####")
time.sleep(3)
#click on the next button
driver.find_element_by_xpath("//span[contains(text(),'Next')][1]").click()
time.sleep(3)
#close the browser
driver.close()
print("Gmail login has been successfully completed")
```

5.3.4 BY CSS SELECTOR

CSS (Cascading Style Sheets) Selectors in Selenium are used to identify and locate web elements based on their id, class, name, attributes and other attributes. CSS is a preferred locator strategy as it is simpler to write and faster as compared to XPath.

By.cssSelector(String cssSelector) method is used to locate the elements in Selenium WebDriver. This method accepts a CSS Selector String as an argument which defines the selection method for the web elements.

The CSS Selector combines an element selector and a selector value that can identify particular elements on a web page. Like XPath in Selenium, CSS selectors can locate web elements without ID, class, or Name.

CHAPTER-6

APPLICATION:K12 -(HERE THE PROJECT IMPLEMENTED ON K12 APPLICATION WITH PYTHON FOR AUTOMATION TESTING

6.1 INTRODUCTION ABOUT K12:

The k-12 education system refers to the school levels from kindergarten to grade class 12. It is an educational system in which children are required to finish these 12 levels as part of their education. As India has one of the most complicated education sectors in the world, we will discuss how the k-12 education system is being implemented here.

Education is an essential part of our growth and development as human beings. Education helps us achieve our full potential to be valuable contributors to an equitable and prosperous society. That is why education systems in all countries should always be improved.

In India, a new policy named the National Education Policy 2020 was implemented. It is an effort to bridge the gap between the current state of the learning process and what must be done to achieve the highest quality of education. The National Education Policy affected the k-12 education system and many others.

So in this article, we will talk about this new policy, the new education structure, and how it incorporated the previous k-12 education system in India.

The National Education Policy 2020 is said to be the first education-related policy of the 21st century. It was made to address the progressing developmental concerns in India. This policy suggests the revisions needed in order to upscale all aspects of the Indian education structure. This includes the regulation of implemented standards to be able to create a new system including the k-12 education system.

The national education policy framework is aligned with Sustainable Development Goal No. 4 or to “ensure inclusive and equitable quality education and promote lifelong learning opportunities for all”. This is while considering India’s traditions and values. In the previous situation, the education system in India incorporates a 10 + 2 structure. However, with the National Education Policy 2020, there will be a curricular structuring that will incorporate a 5 + 3 + 3 + 4 number of years in school. This will cover children ages 3 to 18.

Children below the age of six were not included in the 10 + 2 structure as class 1 coverage begins at 6 years old. In the new structure, there will be a strong foundation of early childhood care and education or ECCE from the age of 3.

The new policy will be divided into four stages namely foundational, preparatory, middle, and secondary. The foundational stage will be composed of 5 years– 3 years for ECCE and 2 years for class 1 and 2. While Preparatory to secondary will be composed of classes 3 to 12 or the remaining classes of the k-12 education system.

6.2 LOGIN FOR K12:

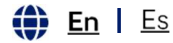
To log in to a K-12 application, you typically need to visit the application's website or use a dedicated mobile app. From there, you'll usually be prompted to enter your username and password provided by your school or educational institution. If you're having trouble logging in, you may need to contact your school's IT department or the application's support team for assistance.

6.3 GRADE FOR K12:

In K-12 education, test grades typically refer to the scores or marks students receive on assessments or exams within an educational application. These grades can range from letter grades (A, B, C, etc.) to numerical scores (out of 100, for example), depending on the grading system used by the school or educational program.

6.4 SUBJECT ART FOR K12:

In K-12 education applications, the subject of art often encompasses various aspects such as visual arts, drawing, painting, sculpture, art history, and more. These applications may offer interactive lessons, creative projects, and resources to help students explore and develop their artistic skills. If you're looking for specific art-related content within a K-12 application, you can usually find it under the art or fine arts category.



Sign in

Don't have an account? [Create one](#)

Email Address

UNIVERSITY

Please enter a valid email address. Ex: k12@gmail.com

Password ⓘ

.....



[Forgot password](#)

Sign In



<https://www.facebook.com/StrideK12>



<https://twitter.com/stridelearn>

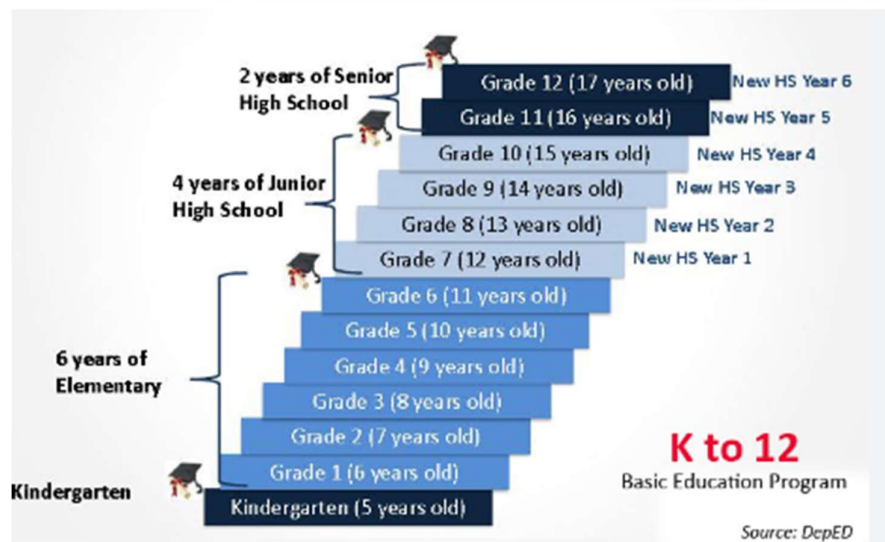


<https://www.instagram>

[Privacy Policy \(http://www.k12.com/privacy-policy#.UtqN22Qo6CU\)](http://www.k12.com/privacy-policy#.UtqN22Qo6CU) [IP Policy \(https://www.stridelearning.com/ip-policy.html\)](https://www.stridelearning.com/ip-policy.html) [Terms of Use \(http://www.k12.com/terms-of-use#.UtqN_mQo6CU\)](http://www.k12.com/terms-of-use#.UtqN_mQo6CU) [Accessibility \(http://www.k12.com/accessibility.html\)](http://www.k12.com/accessibility.html)

Copyright © 2024 K12 Inc. All Rights Reserved

Need help?



6.3 Figure

1



6.4 Figure

What is K-12?

K-12 refers to the educational system in which students study from kindergarten to 12th grade. It is a comprehensive curriculum that covers academic and non-academic subjects and is designed to prepare students for higher education and future careers. The schools teach a variety of subjects to provide students with a well-rounded education.

What is K-12 Indian education system?

In the current Indian education system, K-12 education refers to Kindergarten to the 12th grade. The first ten years of the system provide core education followed by two academic years that prepare the students for higher education in different fields.

The government of India has been taking continual steps to provide education and access to all children despite their backgrounds. The Right to Education Act of 2009 made education compulsory and free of charge for all children of ages 6-14.

What are the different levels of K-12?

In India, educational institutions typically divide the K-12 education system into four levels after Kindergarten. These are as follows:

Primary Level: Classes 1-5

Middle Level: Classes 6-8

Secondary Level: Classes 9 & 10

Senior Secondary Level: Class 11 & 12

Both government and private schools accept these levels. Each school follows a set curriculum, designed keeping in mind the cognitive and emotional level of the students at each level of schooling. The curriculum designers focus on not just academics but also co-curricular activities in a way that would help in the overall development of a student.

What subjects are studied in K-12?

As the world changes and the problems of the modern world come to surface, India's K–12 education system aims to prepare students for facing these challenges head-on by providing a complete and holistic education. This extensive educational curriculum covers basic courses like maths, science, and social studies, along with language and art programs and includes extra-curricular activities such as sports and cultural programs.

Primary Education: Primary education covers the development of the fundamental language, mathematics, and science skills of the students. The teachers introduce students to subjects such as English, Hindi, Mathematics, Science, Social Science, and Environmental Science.

Middle & Secondary Education: Educational authorities in India design the secondary education system to provide students with a broad range of academic and vocational subjects. The curriculum includes subjects like Mathematics, Science, Social Science, Hindi, English, and a third language.

Senior Secondary Education: Designers of the senior secondary education system aim to prepare students for higher education and future careers. The curriculum includes a wide range of subjects such as Mathematics, Physics, Chemistry, Biology, History, Geography, Political Science, Economics, Business Studies, Accountancy, and a choice of electives like Computer Science, Psychology, and Sociology.

Examinations: Examinations play a crucial role in the K-12 education system in India. The primary and secondary education system follows a continuous evaluation system that involves regular assessments and examinations. The senior secondary education system has a more standardized examination pattern, with students appearing for board exams at the end of classes 10 and 12.

What are the benefits of K-12 education?

The system of K-12 education lays the foundation for students to develop and master academic as well as co-curricular skills. K-12 education is important for learning social skills like working together and communicating. It helps students make friends and find role models. The internet is also important for social skills. The system of K-12 education lays the foundation for students to develop and master academic as well as co-curricular skills. This is important for learning social skills like working together and communicating. It helps students make friends and find role models.

Schools that follow K-12 teach students to use technology to learn new things and help prepare students

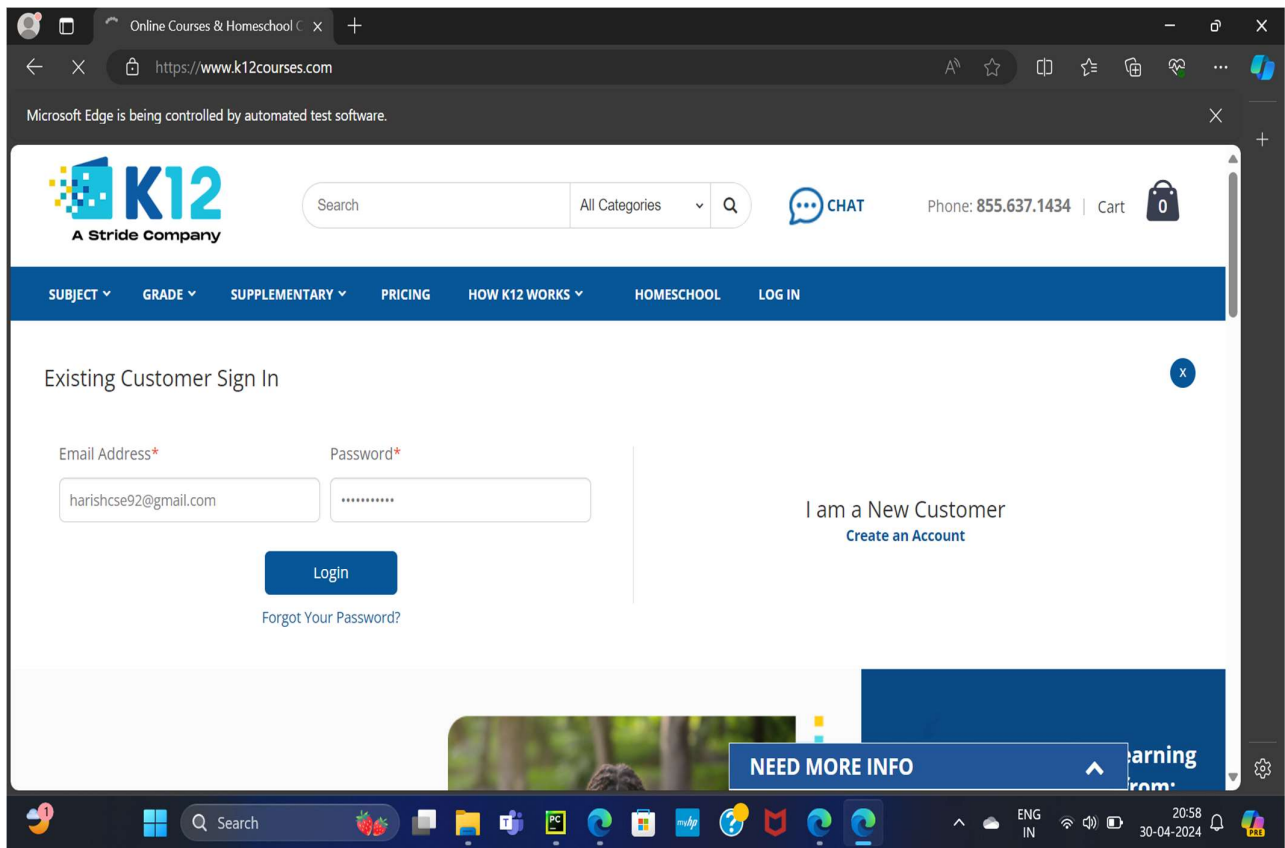
for jobs and make them better at working with different people. It helps them develop skills necessary for when they step out into the 21st-century world. The Indian education system keeps up with the trends in education to help equip its students with the necessary knowledge and skills to compete on a global level. The Indian education system follows a structure where the curriculum is divided into primary, middle, secondary, and senior secondary education. The government has also taken steps to ensure that children between the ages of 6-14 get compulsory free education. The students are introduced to basic subjects like maths, social science, science, and languages along with sports, art, dance, and others. The benefits of K-12 education include mastering academic and social skills, developing teamwork and communication, using technology to learn, and preparing for jobs.

7. IMPLEMENTATION AND RESULT ANALYSIS:

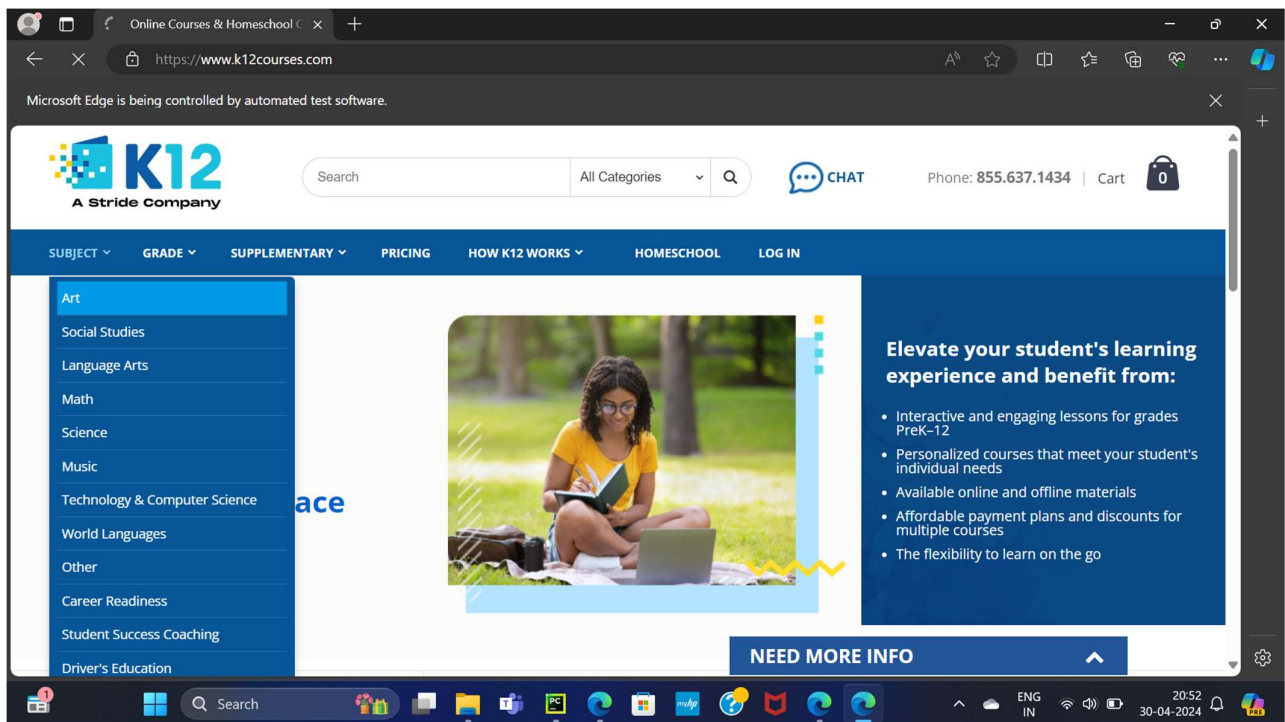
Implementation of testing refers to the practical application of testing methodologies and processes within a software development project. This includes activities such as designing test cases, executing tests, analyzing results, and making improvements based on feedback. It involves selecting appropriate testing tools, establishing testing environments, and integrating testing into the development workflow to ensure the quality and reliability of the software product.

Implementing Python for automation testing involves leveraging Python-based testing frameworks and libraries to streamline the testing process. Developers use frameworks like unittest, pytest, or behave to structure and organize their test suites efficiently. With Python, they write test scripts that simulate user interactions and verify expected outcomes, automating repetitive testing tasks. Libraries such as Selenium for web testing, Appium for mobile testing, and requests for API testing provide essential functionalities to interact with applications programmatically. Integrating these scripts with Continuous Integration/Continuous Deployment (CI/CD) pipelines ensures automatic testing with each code change, promoting faster feedback loops. Python's rich ecosystem facilitates result analysis and report generation, enabling teams to identify and address bugs promptly. Regular maintenance of test scripts ensures their reliability and keeps pace with evolving application features. Overall, Python's versatility and ease of use make it a popular choice for automation testing, improving software quality and development efficiency.

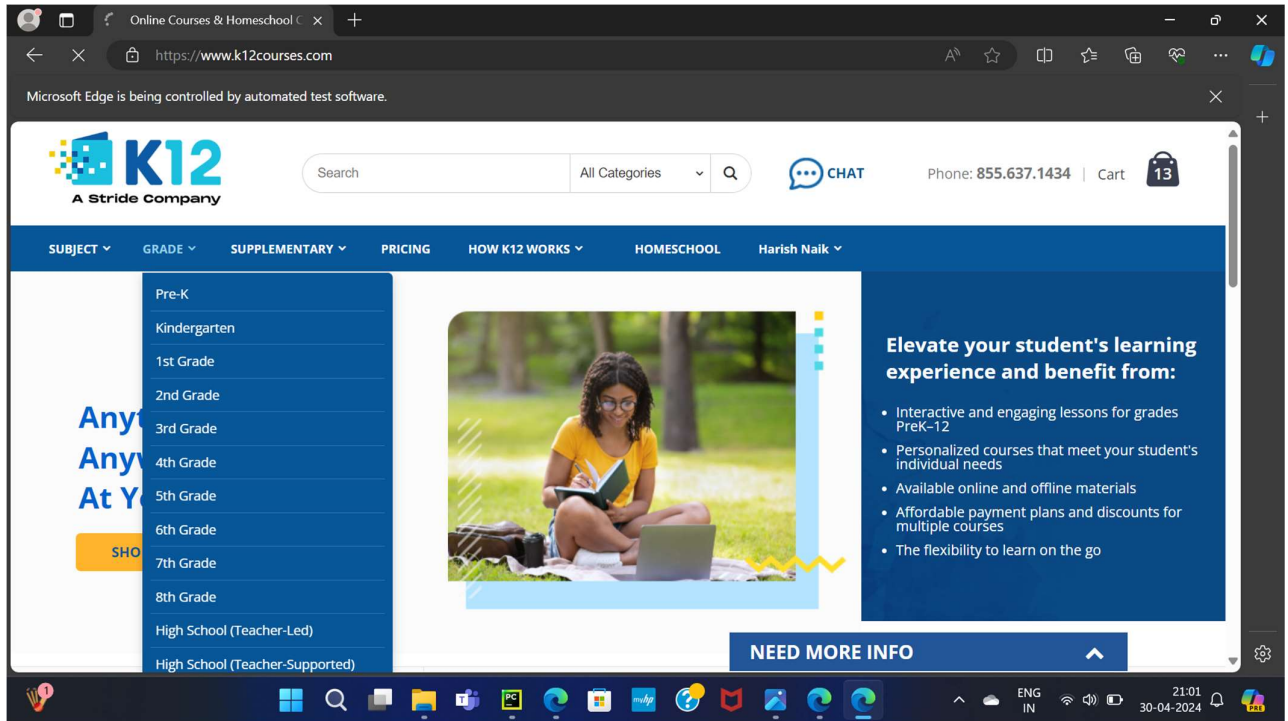
OUTPUT SCREENSHOTS:



7 Figure-1



7 Figure-2



7 Figure-3

Output Analysis: Microsoft Edge is Being Controlled by automated test software.

8. CONCLUSION AND FUTURE ENHANCEMENTS:

CONCLUSION ABOUT PYTHON FOR AUTOMATION TESTING:

In conclusion, automation testing is a critical component of any software development process. It ensures the quality and reliability of applications while reducing the time and effort required to perform manual testing.

Python is a popular programming language extensively used in automation testing due to its simplicity, readability, and vast array of libraries and frameworks. In automation testing, Python enables testers to write scripts to automate repetitive tasks such as test case execution, data manipulation, and result analysis. With libraries like Selenium, Pytest, and Behave, Python empowers testers to automate web testing, unit testing, and behavior-driven development (BDD) scenarios efficiently. Its versatility allows integration with other tools and technologies, making it an ideal choice for building robust automation testing frameworks across various domains and industries.

The conclusion regarding using Python for automation testing is generally positive. Python offers several advantages for automation testing:

1. **Readability:** Python's simple and clean syntax makes it easy to write and understand test scripts, which can improve collaboration among team members.
2. **Rich Ecosystem:** Python has a vast ecosystem of libraries and frameworks that support automation testing, including Selenium WebDriver, pytest, unittest, and more. These tools provide various functionalities to streamline the testing process.
3. **Cross-Platform Compatibility:** Python is cross-platform, meaning that test scripts written in Python can run on different operating systems without modification, enhancing flexibility and scalability.
4. **Community Support:** Python has a large and active community of developers, which means there is extensive documentation, tutorials, and support available for automation testing with Python.

Overall, Python is a robust and versatile language for automation testing, offering a balance of simplicity, power, and community support that makes it an excellent choice for testing web applications and software systems.

FUTURE ENHANCEMENTS:

Python's future in automation testing holds promise with anticipated enhancements focusing on improving efficiency, scalability, and integration capabilities. Enhanced machine learning and AI integration can enable smarter test case generation and predictive analysis, reducing manual effort and enhancing test coverage. Continued development of advanced testing frameworks, such as pytest and Robot Framework, will streamline test automation processes further. Additionally, with the growing adoption of DevOps and continuous testing practices, Python is poised to evolve with better support for CI/CD pipelines and cloud-based testing environments. Collaboration with other languages and technologies, along with ongoing community contributions, will ensure Python remains a top choice for automation testing, adapting to the evolving needs of the industry.

Python is already a powerful tool for automation testing, but there's always room for enhancement. Some potential future enhancements could include:

1. **Improved Integration with AI/ML:** Leveraging machine learning to enhance test automation by predicting potential failures or suggesting optimizations based on past test results.
2. **Enhanced Parallel Testing:** Improving frameworks and tools to facilitate parallel execution of tests across multiple environments, devices, or platforms to speed up testing cycles.
3. **Deeper Integration with DevOps Tools:** Strengthening integrations with popular DevOps tools like Jenkins, Docker, and Kubernetes to seamlessly incorporate automated testing into CI/CD pipelines.
4. **Advanced Reporting and Analytics:** Developing better reporting and analytics features to provide comprehensive insights into test results, trends, and performance metrics.
5. **Native Mobile Testing Support:** Further development of libraries and tools for native mobile app testing, enabling automation of interactions with mobile devices and emulators.
6. **Enhanced Cross-Browser Testing:** Improving capabilities for automated testing across different web browsers and versions, including better handling of browser-specific behaviors and compatibility testing.
7. **Integration with Cloud Services:** Enhanced integration with cloud-based testing services for scalability, resource management, and access to a wider range of testing environments.

8. **Simplified Test Scripting:** Continued efforts to simplify and streamline the process of writing and maintaining test scripts, possibly through improved libraries, frameworks, or IDE integrations.

9. **Better Support for Non-Functional Testing:** Enhancing support for non-functional testing aspects such as performance, security, and accessibility testing within automated test frameworks.

10. **Community Collaboration and Contributions:** Encouraging community collaboration and contributions to continuously enhance existing frameworks, libraries, and tools for automation testing in Python.

We all have experienced an increase in technological advancements in recent years. Automation testing is one of them which is a mandated step in SDLC to avoid software performance issues. The career in automation testing is growing and has also become very demanding and high-paying.

REFERENCES

1. Murphy, R. F. (2019). Artificial intelligence applications to support K-12 teachers and teaching. *Rand Corporation*, 10.
2. Gamido, H. V., & Gamido, M. V. (2019). Comparative review of the features of automated software testing tools. *International Journal of Electrical and Computer Engineering*, 9(5), 4473.
3. Gerard, L., Matuk, C., McElhaney, K., & Linn, M. C. (2015). Automated, adaptive guidance for K-12 education. *Educational Research Review*, 15, 41-58.
4. Qian, Z., Miao, H., & Zeng, H. (2007, December). A practical web testing model for web application testing. In *2007 third international IEEE conference on signal-image technologies and internet-based system* (pp. 434-441). IEEE.
5. Burstein, J., Chodorow, M., & Leacock, C. (2003, August). CriterionSM Online Essay Evaluation: An Application for Automated Evaluation of Student Essays. In *IAAI* (pp. 3-10).
6. Rafi, D. M., Moses, K. R. K., Petersen, K., & Mäntylä, M. V. (2012, June). Benefits and limitations of automated software testing: Systematic literature review and practitioner survey. In *2012 7th International Workshop on Automation of Software Test (AST)* (pp. 36-42). IEEE.
7. Akgun, S., & Greenhow, C. (2022). Artificial intelligence in education: Addressing ethical challenges in K-12 settings. *AI and Ethics*, 2(3), 431-440.
8. Hammes, F., Broger, T., Weilenmann, H. U., Vital, M., Helbing, J., Bosshart, U., ... & Sonnleitner, B. (2012). Development and laboratory-scale testing of a fully automated online flow cytometer for drinking water analysis. *Cytometry Part A*, 81(6), 508-516.
9. Alves, N. D. C., Von Wangenheim, C. G., & Hauck, J. C. (2019). Approaches to assess computational thinking competences based on code analysis in K-12 education: A systematic mapping study. *Informatics in Education*, 18(1), 17.
10. Burstein, J., Tetreault, J., & Madnani, N. (2013). The e-rater® automated essay scoring system. In *Handbook of automated essay evaluation* (pp. 55-67). Routledge.
11. Carbonell, P., Jervis, A. J., Robinson, C. J., Yan, C., Dunstan, M., Swainston, N., ... & Scrutton, N. S. (2018). An automated Design-Build-Test-Learn pipeline for enhanced microbial production of fine chemicals. *Communications biology*, 1(1), 66.

12. Vos, T. E., Kruse, P. M., Condori-Fernández, N., Bauersfeld, S., & Wegener, J. (2015). Testar: Tool support for test automation at the user interface level. *International Journal of Information System Modeling and Design (IJISMD)*, 6(3), 46-83.
13. Shermis, M. D., Burstein, J., Higgins, D., & Zechner, K. (2010). Automated essay scoring: Writing assessment and instruction. *International encyclopedia of education*, 4(1), 20-26.
14. Perez, C. C. (2017). K-12 Multiplatform Gradebook Application. *Southeast Asian Journal of Science and Technology*, 2(1), 122-135.
15. Miao, X., Brooker, R., & Monroe, S. (2024). Where Generative AI Fits Within and in Addition to Existing AI K12 Education Interactions: Industry and Research Perspectives. *Machine Learning in Educational Sciences: Approaches, Applications and Advances*, 359-384.

