CSCI 4210: Introduction to Software Engineering

University of New Orleans
Department of Computer Science

Project 1: Tech Workshop Development

Overall Objectives

- Develop strong soft skills through technical writing, presentation, and peer support.
- Plan and execute a detailed, hour-long workshop that effectively communicates the assigned technology.
- Gain mastery over a specific industry tool and enhance the understanding and proficiency of peers.
- Foster a collaborative environment by supporting the learning and application of the technology throughout the course.

Description

This project is divided into two stages, focusing on both the delivery of a tech workshop and ongoing support throughout the semester. Each team is assigned a specific technology and is responsible for developing and presenting a comprehensive workshop, followed by providing support to peers as they work with the technology.

- **Stage 1:** Workshop Delivery
- **Stage 2:** Ongoing Technical Support

Stage 1: Workshop Delivery

Responsibilities

Developing the Workshop

Each team will act as Subject Matter Experts (SMEs) for their assigned technology. They are responsible for thoroughly researching the tool, understanding its use cases, and mastering its functionalities.

Preparing the Presentation

Teams will create a comprehensive setup guide, usage demonstration, and presentation slides that effectively communicate the technology's key concepts, benefits, and applications.

Live Walkthrough

During the workshop, teams will conduct a live walkthrough of the technology for their peers. This session should be interactive, allowing peers to follow along and ask questions to ensure they understand the material.

Key Deliverables

- A detailed setup guide and documentation for the technology.
- Presentation slides that explain the motivation, concepts, and practical applications of the technology.
- A live demonstration showcasing the use of the technology in real-time.

Note: The first team to present will receive bonus points for taking the initiative.

Stage 2: Ongoing Technical Support

Responsibilities

Supporting Peers

After the initial workshop delivery, each team is responsible for providing ongoing technical support for their assigned technology throughout the semester. This includes assisting peers as they independently perform the lab tasks demonstrated in the workshop. Teams should be ready to answer questions, troubleshoot issues, and guide peers as they replicate the workshop exercises and apply the technology in their projects.

Maintaining Engagement

Teams should actively engage with their peers to foster a collaborative learning environment, encouraging the use of the technology and addressing any challenges that arise as students work through the lab tasks on their own.

Updating Documentation

As peers work with the technology, teams may need to update their documentation or provide additional resources to address common questions or issues that emerge during independent lab work.

Key Deliverables

- Consistent and effective technical support for peers throughout the semester as they complete the assigned labs.
- Updated and refined documentation based on peer feedback and common issues encountered during independent practice.

Instructions for Students

- Deeply investigate the assigned technology and understand its use cases.
- Explore and thoroughly read API documentation and manuals.
- Install and set up the tools in your development environment.
- Use tutorials and walkthroughs to master the tool.
- Author a comprehensive lab document with a detailed step-by-step guide.
- Utilize AI tools for content creation but ensure the final product is coherent and original.
- Produce presentation slides explaining the motivation, concepts, and value of the tool for team-based development.

Rubric Point Allocation

Walk-through Document (30 points)

Reasoning: This document is crucial as it provides the foundation for understanding and using the assigned technology. It should be comprehensive, clear, and detailed, ensuring that peers can follow the instructions and learn effectively.

Criteria to Evaluate:

- Clarity and organization (10 points)
- Completeness of content (10 points)
- Accuracy and correctness (5 points)
- Use of visuals (diagrams, screenshots) and examples (5 points)

Presentation Slides Explanation (25 points)

Reasoning: Presentation slides are essential for delivering key concepts and guiding the audience through the workshop. They should be well-structured, visually appealing, and effectively communicate the main points.

Criteria to Evaluate:

- Organization and flow of content (8 points)
- Visual design and use of multimedia (7 points)
- Clarity and conciseness of explanations (5 points)
- Inclusion of relevant examples and illustrations (5 points)

Execution of LIVE Workshop (30 points)

Reasoning: The live workshop execution demonstrates the team's understanding of the material and their ability to teach it. This component is vital as it assesses both knowledge and presentation skills.

Criteria to Evaluate:

- Clarity and engagement in delivery (10 points)
- Ability to answer questions and provide explanations (8 points)
- Effective demonstration of the technology (7 points)
- Time management and pacing (5 points)

Technical Support and Engagement with Peers (15 points)

Reasoning: Providing ongoing technical support and engaging with peers is important for reinforcing learning and fostering a collaborative environment. This encourages teamwork and helps build a supportive learning community.

Criteria to Evaluate:

- Responsiveness to peer questions and issues (5 points)
- Quality and helpfulness of support provided (5 points)
- Proactive engagement and encouragement of peer learning (5 points)

Total Points: 100

Bonus Points Rubric

Early Presentation (5 Points)

Description: The first team to present their workshop will receive bonus points for taking the initiative and setting the tone for the rest of the class.

Criteria: This bonus is automatically awarded to the team that volunteers and successfully delivers the first workshop.

Creativity and Innovation (5 Points)

Description: Teams that demonstrate creativity and innovation in their workshop delivery, such as unique presentation styles, interactive elements, or creative use of the technology, will be awarded bonus points.

Criteria:

- Use of innovative teaching methods or tools (e.g., interactive quizzes, live coding sessions)
- Creative presentation design that enhances understanding and engagement
- Demonstration of a unique or advanced feature of the assigned technology

Exceptional Documentation Quality (5 Points)

Description: Bonus points are awarded for exceptional quality in the setup guide and documentation, including clarity, thoroughness, and usefulness.

Criteria:

- Comprehensive coverage of all necessary setup steps and potential troubleshooting tips
- High-quality visuals (diagrams, screenshots) that enhance understanding
- Use of advanced formatting and organization to improve readability

Outstanding Peer Support and Engagement (5 Points)

Description: Teams that provide outstanding ongoing technical support and actively engage with peers throughout the semester will receive bonus points.

Criteria:

- Proactively offering help and checking in with peers on their progress
- Responding to questions and issues promptly and effectively
- Providing additional resources or tutorials beyond the initial workshop

---

Team-Based Collaborative Tooling

- IDE (Integrated Development Environment)
- Version Control
- Project Management
- Continuous Integration/Continuous Deployment (CI/CD)
- Testing Tools
- Documentation Tools
- Code Quality Tools
- Containerization
- Data Management and Serialization
- API Design and Development
- GUI Design and Development
- Gesture Recognition and Interaction Tools (Leap Motion SDK)

Tooling Descriptions & Workshop Suggestions

---

1. IDE (Integrated Development Environment)

Description: An Integrated Development Environment (IDE) is a software application that provides a comprehensive suite of tools for software development. VS Code is a popular, lightweight, and highly extensible IDE that supports various programming languages and development workflows. This workshop will focus on using VS Code to facilitate teamwork, streamline development processes, and integrate seamlessly with Git and GitHub for version control and collaboration.

Tools to Investigate:

- **Visual Studio Code (VS Code):** A versatile and customizable IDE that supports numerous programming languages and offers robust features for code editing, debugging, and extensions.
- **Git Integration in VS Code:** Built-in Git support that allows users to manage version control directly within the IDE, including staging, committing, branching, and merging.
- **Extensions for Collaboration:** Explore extensions such as Live Share for real-time collaborative coding, and GitLens for enhanced Git capabilities and insights.
- **GitHub Integration:** Utilize the GitHub extension for VS Code to streamline the workflow of pulling, pushing, cloning repositories, and managing pull requests directly within the IDE.

Project Focus:

- **Setup and Configuration:** Guide students on installing and configuring VS Code on Raspberry Pis, ensuring optimal performance and compatibility with their development environment.
- **Git and GitHub Workflow:** Teach students how to use VS Code's integrated Git features to manage code versions. This includes initializing repositories, making commits, branching, merging, and resolving conflicts.
- **Collaborative Coding:** Demonstrate how to use VS Code extensions like Live Share to facilitate real-time collaboration between team members, enabling them to work together on the same codebase simultaneously.
- **Enhanced Git Capabilities:** Explore the GitLens extension to provide deeper insights into Git repositories, such as detailed file histories, blame annotations, and rich comparison tools. This helps teams understand changes and contributions more effectively.
- **Integrated Pull Request Management:** Show how to manage pull requests directly from VS Code using the GitHub extension, allowing students to review, comment, and merge changes without leaving the IDE.

- **Task Automation and Snippets:** Utilize VS Code's tasks and snippets to automate repetitive development tasks and standardize code snippets across the team, increasing productivity and consistency.
- **Customization for Efficiency:** Encourage students to customize their VS Code environment with settings, themes, and keybindings that enhance their workflow and make development more efficient.

---

2. Version Control

Description: Version control systems are essential tools for managing changes to code and facilitating collaboration among developers. Git, the most widely used version control system, helps teams track code changes, manage branching and merging, and maintain a clean and organized codebase. This workshop will focus on mastering advanced Git commands and workflows to enhance team collaboration and maintain code integrity.

Tools to Investigate:

- **Git:** A distributed version control system that allows multiple developers to work on a project simultaneously without overwriting each other's work. The workshop will cover advanced Git techniques and strategies to improve workflow efficiency.

Branching Strategies:

- **Feature Branching:** Use feature branches to develop new features independently from the main branch, enabling multiple developers to work concurrently without affecting the main codebase.
- **Git Flow:** Explore the Git Flow branching model, which defines roles for different branches and provides guidelines for merging, enhancing workflow structure and clarity.
- **Trunk-Based Development:** Discuss trunk-based development, where all developers work on a single branch (the trunk) and integrate small, frequent changes, reducing long-lived branches and merge conflicts.

Merging Techniques:

- **Fast-Forward Merging:** Understand how fast-forward merges work when the branch being merged has not diverged from the target branch, resulting in a linear history.
- **Recursive Merging:** Learn recursive merging for combining diverged branches by creating a merge commit, preserving the histories of both branches.
- **Squash Merging:** Practice squash merging to condense multiple commits from a branch into a single commit on the main branch, keeping the commit history clean and concise.

Rebasing:

- **Rebasing Basics:** Use rebasing as an alternative to merging for integrating changes from one branch into another, creating a cleaner, linear project history.
- **Interactive Rebasing:** Perform interactive rebasing to edit commit messages, combine commits, or reorder commits, offering more control over the commit history.
- **Rebase vs. Merge:** Compare rebasing and merging, discussing scenarios where each is more appropriate and the potential pitfalls of rebasing, such as rewriting commit history.

Resolving Conflicts:

- **Conflict Detection:** Detect merge conflicts during merging or rebasing processes to ensure smooth integration of code changes.
- **Manual Conflict Resolution:** Resolve conflicts manually using VS Code or the command line, understanding conflict markers and selecting the correct changes to retain.
- **Best Practices for Conflict Resolution:** Follow best practices for minimizing and efficiently resolving conflicts, including frequent pulls, commits, and clear team communication.

Advanced Git Commands and Techniques:

- **Cherry-Picking:** Learn to cherry-pick specific commits from one branch to another without merging the entire branch, useful for applying hotfixes or selective changes.
- **Stashing:** Utilize git stash to temporarily save changes not ready for commit, allowing for branch switching or updates without losing progress.

---

3. Project Management

Description: Project management tools help teams plan, execute, and monitor the progress of projects. These tools facilitate collaboration, streamline workflows, and ensure projects are completed efficiently. The focus of this workshop will be on utilizing GitHub's project management features to organize and manage team projects effectively.

Tool to Investigate:

- **GitHub Projects and GitHub Issues:** Use GitHub Projects for kanban-style project management and GitHub Issues for tracking tasks, bugs, and enhancements. Integrate these tools with GitHub Actions for automated workflows, ensuring a seamless project management experience from planning to execution.

Project Focus:

- **GitHub Projects:** Develop a project plan using GitHub Projects, creating boards to visualize tasks and workflows. Learn to set up columns for different stages of work (e.g., To Do, In Progress, Done) and move issues through these stages to track progress.
- **GitHub Issues:** Utilize GitHub Issues to create and manage tasks, report bugs, and request new features. Learn how to assign issues to team members, label them for categorization, and prioritize them for efficient project execution.
- **Progress Tracking:** Focus on how to use GitHub Projects and Issues to monitor the progress of tasks, track milestones, and ensure team members are aligned and on schedule.

---

4. CI/CD (Continuous Integration/Continuous Deployment)

Description: CI/CD is a set of practices and tools that enable teams to deliver software more frequently and reliably by automating various stages of app development, testing, and deployment. This workshop will focus on setting up and optimizing CI/CD pipelines using GitHub Actions and Git hooks to automate workflows both locally and in the cloud.

Tools to Investigate:

- **GitHub Actions:** A tool for automating software workflows directly in GitHub, supporting continuous integration, deployment, and delivery.
- **Git Hooks:** Scripts that Git executes before or after certain events, such as committing, merging, and pushing code, enabling automation at the local development level.

Project Focus:

- **Thorough Coverage of GitHub Actions:** Explore GitHub Actions in depth, covering all its capabilities, including workflow syntax, triggers, jobs, and steps. Explain how to create custom actions and use existing ones from the GitHub Marketplace to automate different stages of the software lifecycle.
- **Git Hooks Usage and Setup:** Demonstrate how to set up and use Git hooks to automate tasks at different points in the Git workflow (e.g., pre-commit, pre-push, post-merge). Show how Git hooks can be used for enforcing coding standards, running tests, and other checks before code is shared with others or deployed.
- **CI/CD Pipeline Setup:** Teach how to set up a complete CI/CD pipeline using GitHub Actions on Raspberry Pis, including automating tasks such as code compilation, testing, linting, and deployment.
- **Advanced Workflows:** Focus on advanced topics like matrix builds, conditional workflows, environment secrets, and artifact management. Showcase how these features can be used to create efficient and flexible CI/CD pipelines.

- **Integration with Other Tools:** Highlight how GitHub Actions can integrate with other tools and services (e.g., Docker, Kubernetes, cloud providers) to facilitate more complex deployment scenarios and environments.
- **Monitoring and Reporting:** Teach how to monitor CI/CD pipelines using GitHub Actions logs and status checks, and how to set up notifications and reporting to keep the team informed about pipeline results and issues.

---

5. Testing Tools

Description: Testing is an essential aspect of software development, ensuring that code functions as expected and is free of bugs, performance issues, and security vulnerabilities. This workshop will explore various testing methodologies, from unit testing to integration and functional testing. Participants will learn how to create effective tests, manage test data, and use testing tools to automate the process of verifying code quality and reliability.

Tools to Investigate:

- **PyTest:** A versatile and powerful testing framework for Python that supports a wide range of testing needs, including unit, integration, and functional testing. PyTest's simple syntax encourages writing readable and maintainable tests.
- **Mocking Libraries:** Tools such as unittest.mock and pytest-mock are essential for simulating dependencies and controlling the behavior of complex components in a test environment. They allow for more isolated and controlled testing, enabling developers to focus on the code under test without interference from external systems.
- **Faker:** A library for generating fake data that can be used in tests to create realistic input scenarios. This ensures that tests are robust and independent of external data sources, providing consistent results across different environments.

Project Focus:

- **Types of Testing:** Teach the different types of automated testing, including unit testing, integration testing, and functional testing. Explain the purpose of each type and when to use them to ensure comprehensive test coverage.
- **Writing Effective Test Cases:** Guide students on how to write clear, concise, and effective test cases using PyTest. Cover best practices for organizing tests, using assertions, and ensuring tests are maintainable and easy to understand.
- **Mocking and Stubbing:** Demonstrate how to use mocking libraries to simulate dependencies and external systems in tests. Show how mocking can be used to isolate the code being tested and avoid side effects from external dependencies. Discuss when and why to use mocks, stubs, and fakes, and provide practical examples.
- **Test Data Management:** Discuss the importance of managing test data and using tools like Faker to generate realistic data for testing. Highlight how to create

reusable and reliable test data sets that help validate the behavior of the code under different conditions.

- **Test-Driven Development (TDD):** Introduce the concept of TDD, where tests are written before the actual code. Show how TDD can drive design and ensure that the code meets its requirements from the start.
- **Behavior-Driven Development (BDD):** Briefly touch on BDD as a collaborative approach to developing test scenarios based on the expected behavior of the software. Discuss how tools like pytest-bdd can be used to implement BDD practices.

---

6. Documentation

Description: Documentation is a critical part of software development, ensuring that code is understandable, maintainable, and usable by others. This workshop will explore various methods of documenting code and projects, from auto-documentation to user-facing documentation, and best practices for maintaining comprehensive and organized project documentation.

Tools to Investigate:

- **Sphinx or Doxygen:** Tools for generating documentation from code comments and metadata.
- **Markdown:** A lightweight markup language for creating README files and other text-based documentation.
- **Commit Message Strategies:** Best practices for writing clear, descriptive commit messages that communicate changes effectively.
- **GitHub Pages:** A service for hosting web-based documentation directly from a GitHub repository.
- **GitHub Wiki:** A platform for collaborative documentation within a GitHub repository, allowing teams to manage and update project information dynamically.

Project Focus:

- **Generating Auto-Documentation:** Guide students through setting up Sphinx or Doxygen to automatically generate documentation from code comments and metadata. Include steps for configuring these tools, running them to produce documentation, and customizing the output format to suit different needs.
- **Creating and Managing README Files:** Teach students how to write effective README files using Markdown. Cover key components of a good README, such as an overview, installation instructions, usage examples, and contribution guidelines. Emphasize the importance of keeping README files up-to-date and informative.

- **Effective Commit Messages:** Demonstrate how to write clear and descriptive commit messages that communicate changes effectively. Provide examples of good and bad commit messages and explain how well-written messages improve collaboration and project tracking.
- **Hosting Documentation on GitHub Pages:** Show students how to host their documentation on GitHub Pages. Include steps for setting up a GitHub Pages site, linking it to their repository, and deploying documentation generated with Sphinx or Doxygen. Discuss the benefits of using GitHub Pages for making documentation easily accessible.
- **Using GitHub Wiki for Collaborative Documentation:** Explain how to use GitHub Wiki to manage project documentation dynamically. Demonstrate creating and organizing pages in the wiki, adding and formatting content, and collaborating with team members to keep the wiki updated.
- **Documenting Best Practices:** Discuss the importance of documenting best practices within a project. Encourage students to create a "Best Practices" guide or section in their documentation that includes coding standards, naming conventions, workflow guidelines, and other relevant practices that help maintain consistency and quality in the project.

---

7. Code Quality

Description: Code quality is crucial in software development, ensuring that code is not only functional but also maintainable, readable, and free of common errors or bad practices. This workshop will explore tools and techniques for analyzing code quality, identifying potential issues before they become bugs, and enforcing coding standards. Participants will learn how to integrate these tools into the development workflow, enabling continuous code quality checks and improvements.

Tools to Investigate:

- **SonarQube:** A powerful tool for static code analysis that inspects code for bugs, vulnerabilities, code smells, and compliance with coding standards. SonarQube supports multiple programming languages and provides detailed insights into code quality and maintainability.
- **ESLint or Pylint:** Linters for JavaScript and Python that analyze code for syntax errors, style violations, and potential issues, enforcing consistent coding standards across the team.
- **Prettier:** An opinionated code formatter that enforces consistent code style, improving readability and reducing formatting-related merge conflicts.

Project Focus:

- **Setting Up SonarQube:** Guide students through the installation and configuration of SonarQube on Raspberry Pis, including setting up the SonarQube server and connecting it to a code repository for analysis.

- **Running Code Analysis:** Demonstrate how to run static code analysis using SonarQube and interpret the results. Show how to identify and resolve issues such as bugs, vulnerabilities, and code smells that can affect code quality and maintainability.
- **Integrating with CI/CD:** Teach how to integrate SonarQube with GitHub Actions for continuous code quality analysis. Explain how to set up a pipeline that automatically runs code quality checks on every commit or pull request, providing instant feedback to developers.
- **Using Linters and Formatters:** Introduce tools like ESLint, Pylint, and Prettier to enforce coding standards and style guidelines. Show how to configure these tools for a project and integrate them with the development environment and CI/CD pipeline for automated checks.
- **Customizing Quality Gates:** Discuss how to configure SonarQube's quality gates to define thresholds for code quality metrics, such as code coverage, duplications, and maintainability ratings. Show how to enforce these gates in the CI/CD pipeline to prevent code that does not meet quality standards from being merged.
- **Code Review Best Practices:** Emphasize the importance of code reviews in maintaining code quality. Provide best practices for conducting effective code reviews, focusing on identifying code smells, ensuring adherence to coding standards, and improving code readability.

---

8. Containerization

Description: Containerization is a technology that packages an application and its dependencies into a single, lightweight container, ensuring consistent behavior across different environments. This workshop will introduce the fundamentals of containerization, focusing on Docker, the most widely used containerization platform. Students will learn how to create, manage, and deploy Docker containers, enabling them to build applications that are portable, scalable, and easy to deploy in various environments, including development, testing, and production.

Tools to Investigate:

- **Docker:** A platform for developing, shipping, and running applications inside containers. Docker allows developers to package applications with all necessary components, such as libraries and dependencies, to ensure they run consistently in any environment.
- **Docker Compose:** A tool for defining and running multi-container Docker applications. With Docker Compose, students can manage multiple containers as a single application, making it easier to orchestrate complex development environments.
- **Docker Hub:** A cloud-based repository service that allows developers to share and store container images. Students will learn how to push and pull images from Docker Hub to streamline development workflows.

Project Focus:

- **Understanding Docker Basics:** Teach students the fundamentals of Docker, including how containers differ from virtual machines, and the benefits of using containers in software development. Cover core concepts like Docker images, containers, Dockerfiles, and the Docker Engine.
- **Building Docker Containers:** Guide students through the process of creating Docker containers for a sample application. Include instructions for writing Dockerfiles to define the environment, dependencies, and commands needed to build and run the application.
- **Managing Docker Containers:** Show students how to manage Docker containers effectively, including starting, stopping, and restarting containers. Cover how to inspect running containers, view logs, and troubleshoot common issues.
- **Using Docker Compose:** Introduce Docker Compose for managing multi-container applications. Demonstrate how to create a docker-compose.yml file to define and run multiple interconnected containers, such as a web server and a database, as a single application.
- **Deploying Containers on Raspberry Pis:** Demonstrate how to deploy Docker containers on Raspberry Pis. Discuss the challenges and considerations of running containers on ARM-based devices and how to optimize Docker images for performance and efficiency on Raspberry Pis.

---

9. Serialization & Data Management

Description: Serialization is the process of converting objects into a format that can be easily stored or transmitted, enabling data to be saved to a file or sent over a network. Data management involves organizing, storing, and retrieving data efficiently to ensure its integrity and accessibility. This workshop will explore various methods of data serialization and management, focusing on best practices for using serialization formats and data storage solutions. Students will learn how to manage data locally on Raspberry Pis and synchronize it with a centralized database to ensure consistent and reliable data access across devices.

Tools to Investigate:

- **JSON (JavaScript Object Notation):** A lightweight data-interchange format that is easy for humans to read and write and easy for machines to parse and generate. JSON is widely used for serializing and transmitting structured data over a network connection.
- **SQLite:** A self-contained, serverless, zero-configuration, transactional SQL database engine. SQLite is ideal for embedding in applications to manage local storage efficiently.

- **CSV (Comma-Separated Values):** A simple file format used to store tabular data, such as a spreadsheet or database. CSV files are commonly used for data export and import between different systems.
- **Pickle:** A Python-specific binary serialization format that allows objects to be saved to a file and restored later. It is useful for saving complex data structures but is less portable than JSON.
- **MySQL or PostgreSQL:** Relational database management systems (RDBMS) that can be used to store and synchronize data across multiple Raspberry Pis, providing centralized data management and access.
- **rsync:** A utility for efficiently transferring and synchronizing files between computers and storage devices, which can be used to synchronize data between local file systems on Raspberry Pis.

Project Focus:

- **Understanding Serialization Formats:** Introduce students to different serialization formats, including JSON, CSV, and Pickle. Explain the advantages and disadvantages of each format and when to use them based on the data type and use case.
- **Working with JSON:** Teach students how to serialize and deserialize data using JSON. Demonstrate how to convert Python objects to JSON strings and back, and how to work with JSON data in different contexts, such as web APIs and data storage.
- **Managing Data with SQLite:** Guide students through setting up and using SQLite as a lightweight database solution. Show how to create, read, update, and delete data using SQL commands within Python scripts, and discuss when SQLite is a suitable choice for local data management.
- **Synchronizing Data with a Central Database:** Introduce students to MySQL or PostgreSQL as centralized database solutions for synchronizing data across multiple Raspberry Pis. Demonstrate how to connect to a central database from a Raspberry Pi, execute SQL queries, and synchronize data between local SQLite databases and the central database.
- **Using rsync for File Synchronization:** Demonstrate how to use rsync to synchronize data files between Raspberry Pis. Show how to set up rsync commands to keep local file systems in sync, ensuring data consistency across devices.
- **Data Import and Export with CSV:** Demonstrate how to read and write CSV files using Python. Cover best practices for handling CSV data, such as dealing with headers, delimiters, and encoding issues.
- **Data Integrity and Security Best Practices:** Cover best practices for maintaining data integrity and security. Discuss topics such as data validation, encryption, access control, and regular auditing to protect data and ensure its accuracy.

10. API Design & Development

Description: API (Application Programming Interface) design and development involve creating interfaces that allow different software components and applications to communicate. This workshop focuses on strategies for designing robust, efficient, and scalable APIs that can be used both as local Python modules and as REST APIs. Students will learn best practices for modular design, clear interface definition, and code reuse, ensuring that their APIs are easily accessible, maintainable, and usable by other developers, regardless of whether they are accessed locally or over a network.

Tools to Investigate:

- **FastAPI:** A modern, fast web framework for building APIs with Python. FastAPI allows for the creation of RESTful APIs with automatic data validation, type checking, and OpenAPI documentation.
- **Flask:** A lightweight web framework in Python that is flexible and easy to use for building RESTful APIs. Flask allows developers to define routes and endpoints with minimal setup and provides support for extensions and middleware.
- **Python Packaging Tools:** Tools like setuptools and wheel for packaging Python code into distributable modules that can be easily installed and imported by other developers.
- **Postman or Insomnia:** Tools for testing APIs by making HTTP requests and inspecting responses. These tools help developers ensure their APIs function correctly and meet expected behaviors.
- **Swagger/OpenAPI:** A framework for defining and documenting APIs, allowing for consistent API design and auto-generation of documentation.

Project Focus:

- **Principles of Modular API Design:** Teach students the fundamentals of designing modular APIs that can be accessed as local Python modules or through REST endpoints. Discuss principles such as separation of concerns, single responsibility, and DRY (Don't Repeat Yourself), which help create reusable, maintainable, and scalable components.
- **Developing APIs for Local and Remote Access:** Guide students through developing APIs that can be used both locally and remotely. Show how to structure Python code into reusable modules and how to expose these modules via RESTful endpoints using frameworks like FastAPI or Flask. Emphasize how these strategies provide flexibility in how the APIs are consumed.
- **Packaging Code into Python Modules:** Demonstrate how to package Python code into installable modules using setuptools. Teach students how to create `__init__.py` files to define modules and structure their code for reuse in different projects. Discuss best practices for writing clean, maintainable code that can be easily imported and utilized by other developers in various environments.

- **Creating REST APIs with FastAPI or Flask:** Show how to use FastAPI or Flask to create RESTful APIs that expose the same functionality as local Python modules. Teach students how to define endpoints, handle requests and responses, and implement data validation. Compare the two frameworks and their suitability for different use cases.
- **Testing APIs and Modules:** Teach students how to use tools like Postman or Insomnia to test RESTful APIs and ensure they behave as expected. Also, demonstrate how to write unit tests for local Python modules to validate functionality and maintain code quality.
- **API Documentation and Standards with Swagger/OpenAPI:** Guide students through documenting their APIs using Swagger/OpenAPI standards. Show how to auto-generate documentation with FastAPI and manually create it with Flask. Emphasize the importance of keeping documentation up-to-date and accessible for both local modules and REST APIs.
- **Deploying and Distributing APIs:** Demonstrate how to deploy REST APIs on Raspberry Pis or other servers. Discuss the considerations for deploying Python modules and APIs, including environment setup, dependency management, and performance optimization.

---

11. GUI Design & Development

Description: GUI (Graphical User Interface) design and development involve creating visual interfaces for software applications that allow users to interact with the software in an intuitive and efficient manner. This workshop focuses on designing and developing GUIs that enhance user experience, with an emphasis on best practices for layout, usability, and responsiveness. Students will learn how to create GUI applications using tools like Tkinter and Electron, manage their development workflow using GitHub, and automate builds and deployments with GitHub Actions.

Tools to Investigate:

- **Tkinter:** A standard Python library for creating simple and lightweight GUI applications. Tkinter provides a variety of widgets and controls for building desktop applications and is ideal for beginners due to its ease of use and integration with Python.
- **Electron:** A framework for building cross-platform desktop applications using web technologies like HTML, CSS, and JavaScript. Electron allows developers to create GUI applications that work on Windows, macOS, and Linux with a single codebase, making it a powerful tool for creating modern, feature-rich applications.
- **Figma or Sketch:** Tools for designing user interfaces and prototyping. These tools help developers and designers create wireframes, mockups, and prototypes of GUI applications, providing a visual representation of the interface before development begins.

Project Focus:

- **Understanding GUI Design Principles:** Teach students the fundamentals of GUI design, including principles like consistency, feedback, simplicity, and user control. Discuss best practices for creating intuitive and user-friendly interfaces that enhance the overall user experience.
- **Creating GUIs with Tkinter:** Guide students through creating simple GUI applications using Tkinter. Show how to create windows, add widgets (such as buttons, labels, and text boxes), and handle user input. Discuss the limitations of Tkinter and when it is most appropriate to use for GUI development.
- **Developing Cross-Platform GUIs with Electron:** Introduce students to Electron for creating cross-platform GUI applications. Demonstrate how to set up an Electron project, create a basic interface using HTML, CSS, and JavaScript, and integrate platform-specific features. Emphasize the advantages of using Electron for more complex, modern applications.
- **Prototyping and UI Design:** Encourage students to create wireframes and prototypes using tools like Figma or Sketch before starting development. Show how to design interfaces that are easy to use and visually appealing, and discuss the importance of user feedback and iterative design.
- **Testing and Debugging GUI Applications:** Teach students how to test and debug GUI applications effectively. Discuss strategies for finding and fixing bugs, optimizing performance, and ensuring that the application works correctly across different platforms and devices.
- **Deploying GUI Applications:** Guide students through the process of packaging and deploying GUI applications. For Tkinter, demonstrate how to create standalone executables for different operating systems. For Electron, show how to package applications for Linux.

---

12. Leap Motion SDK

Description: The Leap Motion SDK is a software development kit that allows developers to create applications that can interact with users through natural hand and finger movements. This technology enhances interactivity and user experience by enabling touch-free controls and gestures. In this workshop, students will learn how to use the Leap Motion SDK to develop innovative, gesture-controlled applications. They will explore the fundamentals of gesture recognition, integrate the Leap Motion API with their applications, and utilize best practices for development, testing, and deployment on Raspberry Pis.

Tools to Investigate:

- **Leap Motion SDK:** A robust SDK that provides tools and libraries for developing applications that detect and interpret hand and finger movements. The SDK supports multiple programming languages and platforms, allowing for versatile application development.

- **Python or JavaScript (with WebSocket API):** Programming languages commonly used with the Leap Motion SDK. Python offers straightforward scripting capabilities, while JavaScript can be used for developing web-based applications that utilize the WebSocket API to communicate with Leap Motion hardware.

Project Focus:

- **Understanding Gesture Recognition and Interactivity:** Teach students the fundamentals of gesture recognition and the capabilities of the Leap Motion SDK. Discuss how the SDK captures hand and finger movements and translates them into inputs that applications can use to enhance interactivity and user experience.
- **Setting Up the Leap Motion SDK:** Guide students through the installation and setup of the Leap Motion SDK on their development environment. Show how to configure the SDK for different programming languages (such as Python or JavaScript) and integrate it with their projects.
- **Developing Gesture-Controlled Applications:** Demonstrate how to develop applications that utilize the Leap Motion API to recognize and respond to gestures. Teach students how to implement basic gestures (such as swipe, pinch, and grab) and create custom gestures to control application features.
- **Designing Intuitive Interfaces:** Encourage students to design user interfaces that complement the interactive nature of Leap Motion applications. Use tools like Figma or Sketch to create wireframes and prototypes, focusing on intuitive layouts and controls that enhance user experience.
- **Testing and Debugging Leap Motion Applications:** Teach students how to test and debug Leap Motion applications effectively. Discuss strategies for capturing and interpreting gesture data, optimizing performance, and ensuring the application responds accurately to user input.