Introduction to Programing with Python

Module 09 – Continuous Improvement

# Getting Started

Welcome to module 9! The last module ended with a word about Continuous Improvement. In this module you will learn what that process looks like from the prospective of a programing in the field.

In the last module, you answered the following questions:

* What is the difference between a class and a module?
* How do you connect one module to another?
* What is an import alias?
* What is the "main" module?
* What are four examples of UML diagrams?
* What is a Test Harness?
* What is Unit Testing?

**Exercise:** Take a minute or less to write down a one or two sentence answer to the following question as if you were answering a question from a coworker or interviewer: **What questions do I still have about module 8's content?**

[Getting Started 1](#_Toc155077044)

[Continuous Improvement 2](#_Toc155077045)

[The Scrum Agile Methodology 3](#_Toc155077046)

[Improving Dependencies 5](#_Toc155077047)

[Removing Dependencies 7](#_Toc155077048)

[The raise Statement 11](#_Toc155077049)

[Demo/Video Demo01: Using Continuous Improvement 12](#_Toc155077050)

[Improving Documentation 12](#_Toc155077051)

[Markdown Language 13](#_Toc155077052)

[Demo/Video Demo02: Using Markdown 16](#_Toc155077053)

[Improving Analytics 16](#_Toc155077054)

[Jupyter Notebooks 17](#_Toc155077055)

[Creating Charts in PyCharm 23](#_Toc155077056)

[Demo/Video Demo03: Using Jupyter Notebook 23](#_Toc155077057)

[Summary 23](#_Toc155077058)

# Continuous Improvement

Continuous improvement is a crucial practice that allows teams to refine their processes, enhancing the quality of their applications. Here are some key aspects of continuous improvement in application development:

* **Agile** **Methodologies:** Agile methodologies promote **iterative development, frequent feedback, and adaptability**, enabling teams to respond to changing requirements and deliver value incrementally.
* **Regular Feedback Loops:** Encourage **open communication and feedback** among team members, stakeholders, and end-users. Gather feedback early and often to identify issues, capture user needs, and prioritize improvements.
* **Retrospectives:** Conduct regular retrospectives to **reflect on what went well and what could be improved** at the end of each development cycle. Use retrospective meetings to address process bottlenecks, team collaboration, and technical debt.
* **Quality Assurance and Testing:** Invest in automated testing and quality assurance processes to **catch and fix issues early** in the development cycle. Regularly review and update test cases to ensure comprehensive coverage.
* **Continuous Integration and Continuous Deployment (CI/CD):** Implement CI/CD pipelines to **automate code integration, testing, and deployment**. This reduces the risk of errors and accelerates the delivery of new features and bug fixes.
* **Monitoring and Analytics:** Use monitoring tools and **analytics to gain insights** into how users are interacting with your app. Monitor performance, error rates, and user behavior to identify areas for improvement.
* **Technical Debt Management:** Keep an eye on **technical debt**, which can accumulate over time as **shortcuts are taken to meet deadlines**. Regularly prioritize and address technical debt to maintain code quality and reduce long-term maintenance costs.
* **Skill Development:** Invest in the **continuous learning and skill development** of your development team. Encourage them to stay up-to-date with the latest technologies, coding practices, and industry trends.
* **User-Centered Design:** Place a strong emphasis on user-centered design principles. Conduct **usability testing, gather user feedback, and iterate** on the app's user interface and user experience (UI/UX) to enhance user satisfaction.
* **Documentation and Knowledge Sharing:** Maintain up-to-date **documentation for your app's codebase, architecture, and processes. Encourage knowledge sharing** within the team to ensure that everyone is aware of best practices and changes.
* **Risk Management:** Continuously **assess and mitigate risks** associated with your app's development and operation. Plan for contingencies and have strategies in place to address potential challenges.

Continuous improvement is an ongoing journey that embraces these principles and practices. When an organization uses them, they create a culture of learning and adaptation within their development team, leading to better app quality, faster delivery, and increased customer satisfaction.

## The Scrum Agile Methodology

**Scrum is a popular agile methodology** for managing and delivering software development projects. It provides a structured approach to project management and product development that emphasizes collaboration, flexibility, and iterative progress. Scrum is widely used in the software industry and has applications in various other fields.

People participating in Scrum are grouped into interrelated roles. Here are the common ones you are likely to encounter.

* **Scrum Master:** The Scrum Master is responsible for **facilitating the Scrum process**, **removing impediments** that block the team's progress, and ensuring that Scrum principles are followed.
* **Product Owner:** The Product Owner represents the interests of the stakeholders and is responsible for **defining and prioritizing the product backlog**.
* **Development Team:** The Development Team is a group of individuals responsible for delivering the product increment during each sprint.
* **QA Team:** The Quality Assurance Team is a group of individuals responsible for testing the product increment during each sprint.
* **DevOps Team:** DevOps team streamlines the software delivery process, making sure that the software released by the Development and QA teams can reliably installed and ran on each intended computer.

***Note:******Scrum emphasizes cross-functional teams*** *where members have diverse skills, including development and testing. This means that QA activities are not isolated to a separate QA team but are integrated into the development process.*

### Scrum Artifacts

Scrum defines several "artifacts" used to ensure transparency, provide information to the team and stakeholders, and guide the work throughout the development process.

The three primary Scrum artifacts are:

**Product Backlog:** The Product Backlog is **a prioritized list of all the features, enhancements, bug fixes,** and other work items that need to be addressed in the product. It serves as the single source of requirements for the Scrum team. The **Product Owner is responsible for managing and prioritizing the Product Backlog**. The Product Backlog items can include user stories, technical tasks, and any other work needed to create or enhance the product.

**Sprint Backlog:** The Sprint Backlog is a subset of the Product Backlog items selected by the Development Team for implementation during a specific sprint. The Sprint Backlog represents the work the team commits to delivering during the sprint. **The Development Team owns and manages** the Sprint Backlog. It includes **the detailed tasks**, user stories, and other work items that the team plans **to complete during the sprint**.

**Sprint Increment:** The Increment is the **sum of all** the **Backlog items that have been completed and integrated into the product during a sprint**. The entire Scrum team (Development Team, Product Owner, and Scrum Master) is collectively responsible for the Increment. The Increment should be in a potentially shippable state, meaning it meets the Definition of Done and is of high quality. **Each Increment adds value to the product**.

A screenshot of a computer

Description automatically generated

These artifacts work together to guide the Scrum team's work. The Product Backlog helps prioritize and manage the overall scope of the product, while the Sprint Backlog contains the work selected for the current sprint. The Increment represents the tangible outcome of the sprint and demonstrates progress toward completing the product.

### Events or Ceremonies

Scrum is made up of several events, sometime called ceremonies. These represent a collection of meetings where all or part of the teams come together to plan and review progress. Here is a list of the key ones you should know about:

**Sprint: A time cycle, typically lasting 2-4 weeks**, during which the Development/QA teams work to complete a set of backlog items.

**Sprint Planning:** **A meeting at the beginning of each sprint** where the Development Team **selects items from the Product Backlog** to work on **and plan**s how they will accomplish them.

**Daily Scrum:** A **daily** **meeting** where the Development Team **review its activities, discusses progress, and identifies any obstacles**.

**Sprint Review:** A **meeting at the end of each sprint** where the Development Team **demonstrates** the **completed work** to stakeholders and obtains feedback.

**Sprint Retrospective:** A meeting at the end of each sprint **where the team reflects** on their performance, discusses what went well, what could be improved, and creates an action plan for the next sprint.

Scrum is known for its simplicity and can be applied to a wide range of projects, not just software development. Many organizations find it effective in improving collaboration, communication, and the delivery of high-quality products.

It's important to note that Scrum promotes transparency and frequent inspection of these artifacts. The Product Backlog is regularly refined and reprioritized, the Sprint Backlog is updated as the team progresses, and the Increment is inspected during the Sprint Review to gather feedback and determine what to work on next. This iterative and transparent approach helps ensure that the product is continually improved and aligned with the stakeholders' needs.

As an example, let's work on one of the backlog items from Module 08, the removal of dependencies.

A screenshot of a computer

Description automatically generated

# Improving Dependencies

When we designed and created our application in module 08, we uncovered a dependency flaw in its design. While this flaw was not enough to cause the program to fail, it represented a "technical debt" that needs to be fixed.

**Technical debt is a concept** in software development that refers to the cost (in terms of time, effort, and resources) **of not addressing suboptimal or imperfect aspects of a software system during its initial development** or subsequent maintenance. It's a metaphorical term borrowed from the financial world, where debt represents an obligation that needs to be paid off in the future.

Technical debt can accumulate for various reasons, including:

**Time Constraints:** Developers may need to meet tight deadlines, leading them to make quick decisions or take shortcuts that result in less-than-optimal code.

**Lack of Knowledge:** In some cases, developers may not have the necessary expertise or experience to make the best design or architectural choices.

**Changing Requirements:** Software projects often evolve, and new requirements can emerge. Adapting to these changes may lead to code that is not as clean or efficient as it could be.

**Legacy Code:** Older software systems may have been built on outdated technologies or practices, making them harder to maintain and update.

**Unforeseen Issues:** Sometimes, developers encounter unexpected technical challenges that force them to make compromises in the codebase.

**Inadequate Testing:** Skipping thorough testing or quality assurance processes can lead to undetected issues and technical debt. eds to be addressed.

Most projects will accumulate technical debt as the team tries to release an application on time and must make decisions on what is the Minimum Viable Product (MVP).

**A Minimum Viable Product (MVP)** is a simplified version of a new product or software application that **includes only the most essential features and functionality** required to meet the needs of early adopters or early customers. The primary purpose of an MVP is to quickly validate a product idea, gather feedback, and learn from real-world usage while minimizing development time and resources. It serves as a starting point for further development and refinement based on user feedback and market validation.

In our module 8 example **we noted that the dependencies** in the IO and Processor modules should be moved into the main module, but that **we did not have time to fix it**. Since the program worked, passed its tests, and could be **presented to the end users** it represented **the MVP** at that time. This meant that that updating **the dependencies was placed in the** **backlog as a task (or story) that is yet "Todo."**

The issues were the FileProcessor's read and write functions called the IO.output\_error\_messages() function for error handling.

except TypeError as e:  
 IO.output\_error\_messages("Please check that the data is a valid JSON format", e)  
except Exception as e:  
 IO.output\_error\_messages("There was a non-specific error!", e)

Also, the IO module also has a dependency on the data module because the IO uses this code to create a student object.

new\_student = Student(first\_name=student\_first\_name, last\_name=student\_last\_name, gpa=student\_gpa)

As mentioned, it is OK to have dependencies, each one makes your code less reusable in other applications and blurs the lines between the processing, presentation, and layer, disrupting our Separation of Concerns design. To fix this we will modify our code so that only the main module is depended on the processing, presentation, and data modules.

Some dependencies are not worth moving. For example, while we could move the dependency on the json module from the processor module into the main module, it would not just increase the complexity as the main module would have to pass a reference to the json module before the processor could work with the file.

Here is a diagram of our planned changes.

A diagram of a system

Description automatically generated

## Removing Dependencies

To implement these changes, we start with the presentation module and commenting out the import data\_classes statement.

try:  
 if \_\_name\_\_ == "\_\_main\_\_":  
 raise Exception("Please use the main.py file to start this application.")  
 *# else:  
 # import data\_classes as data*except Exception as e:  
 print(e.\_\_str\_\_())

When we do this PyCharm immediately finds an error in our input\_student\_data() function.

A screenshot of a computer code

Description automatically generated

To resolve this, we can **create a new parameter, pass a reference** to the Student class as an argument, and **use the reference to create a Student object**. Here is the code we will use.

@staticmethod *# TODO: Create a Student parameter (Done)*  
def input\_student\_data(student\_data: list, student\_type:object):  *""" This function gets the first name, last name, and GPA from the user  
  
 ChangeLog: (Who, When, What)  
 RRoot,1.1.2030,Created function  
 RRoot,1.2.2030,Converted code to use student objects instead of dictionaries  
  
 :param student\_data: list of dictionary rows to be filled with input data  
 :param student\_type: A reference to data\_class.Student use to create a Student object  
  
 :return: list  
 """* try:  
 *# Input the data  
 # student = data.Student() # TODO: Remove the direct dependency to data\_classes (Done)* student = student\_type()  
 student.first\_name = input("What is the student's first name? ")  
 student.last\_name = input("What is the student's last name? ")  
 student.gpa = float(input("What is the student's GPA? "))  
 student\_data.append(student)  
  
 except ValueError as e:  
 IO.output\_error\_messages("That value is not the correct type of data!", e)  
 except Exception as e:  
 IO.output\_error\_messages("There was a non-specific error!", e)  
 return student\_data

Next, we **modify the main module to pass the reference** like this:

elif menu\_choice == "2": *# Get new data (and display the change)  
 # TODO: Add as Reference Argument to Student (Done)* students = pres.IO.input\_student\_data(student\_data=students, student\_type=data.Student)   
 pres.IO.output\_letter\_by\_gpa(student\_data=students)  
 continue

***Tip:*** *We do not need to fix the calls to IO.output\_error\_messages() since the input\_student\_data() function is in the same class.*

Notice that **we pass a reference to the Student class and not an object**, as we have always done in this course so far. It can be easy to miss if you are not watching for it but note that code argument is Student and not Student(). Using the parentheses would create a Student object argument, but we want to pass a reference to the Student class and create a Student object later within the function.

students = pres.IO.input\_student\_data(student\_data=students, student\_type=data.Student)

...

student = student\_type()

...

...

### Polymorphic Types

Also, notice **the default data type of the student\_type parameter is "object"**, not Student. It is needed since using Student would require us to import the data\_classes modules again. We use the object class is always available in Python as the ultimate class that all others inherit from.

def input\_student\_data(student\_data: list, student\_type=object):

This is replacement work since all "student" objects are "person" objects and also "object" class objects. **Student is just one of the "many versions" of the object class.** The fancy term for these relationships is **called Polymorphic**.

A blue oval with black text

Description automatically generated

Polymorphism is a fundamental concept that allows objects of different classes to be treated as if they are objects of a common parent class. This means that you can use a common class to interact with objects of various classes, even if those classes have different implementations for the same methods or attributes.

Here's how you can understand this relationship and polymorphism:

Class Hierarchy:

* **object**: This is the base class for all Python objects. Every class you create implicitly or explicitly inherits from the object class.
* **person**: This class represents a generic person with certain attributes and behaviors.
* **student**: This class represents a specialized type of person, which is a student. It inherits properties and behaviors from the "person" class.

Polymorphism allows you to treat objects of different classes in a consistent way. In our case, since "student" objects inherit from "person" objects, it can use methods and attributes defined in the "person" class on "student" objects. This is a powerful aspect of polymorphism in programming, making your code more flexible and easier to maintain.

We can use the same technique to remove the dependency from the processing\_classes module. First, we will comment out the imports in that module.

try:  
 if \_\_name\_\_ == "\_\_main\_\_":  
 raise Exception("Please use the main.py file to start this application.")  
 else:  
 import json  
 *# import data\_classes as data  
 # import presentation\_classes as pres*except Exception as e:  
 print(e.\_\_str\_\_())

Next, we **locate the errors** caused by this removal and find errors based on the reference to the data\_classes module and others base on the reference to the presentation classes (**look for the red underlines**.)

A computer screen shot of a code

Description automatically generated

To fix the data\_classes reference we once again add a parameter to the read\_data\_from\_file() function like this:

@staticmethod *# TODO: Create a Student parameter (Done)*  
def read\_data\_from\_file(file\_name: str, student\_data: list, student\_type: object):  *""" This function reads data from a json file and loads it into a list of dictionary rows  
  
 ChangeLog: (Who, When, What)  
 RRoot,1.1.2030,Created function  
 RRoot,1.5.2030,Converted list of dictionaries to list of student objects  
  
 :param file\_name: string data with name of file to read from  
 :param student\_data: list of dictionary rows to be filled with file data*

*:param student\_type: A reference to data\_class.Student use to create a Student object*

*:return: list  
 """* try:  
 with open(file\_name, "r") as file:  
 list\_of\_dictionary\_data = json.load(  
 file) *# the load function returns a list of dictionary rows.* for student in list\_of\_dictionary\_data:  
 *# student\_object: data.Student = data.Student(first\_name=student["FirstName"],  
 # last\_name=student["LastName"],  
 # gpa=student["GPA"])* student\_object = student\_type(first\_name=student["FirstName"],  
 last\_name=student["LastName"],  
 gpa=student["GPA"])  
   
 student\_data.append(student\_object)  
 except FileNotFoundError as e:  
 pres.IO.output\_error\_messages("Text file must exist before running this script!", e)  
 except Exception as e:  
 pres.IO.output\_error\_messages("There was a non-specific error!", e)  
 return student\_data

We then update the code in the main module as we did before.

*# TODO: Add as Reference Argument to Student (Done)*students = proc.FileProcessor.read\_data\_from\_file(file\_name=data.FILE\_NAME,   
 student\_data=data.students,  
 student\_type=data.Student)

No doubt you noticed that **we still need to fix the dependency on the presentation** **module**, but to do that we use a different technique via the raise statement.

## The raise Statement

The raise statement in Python is used to **intentionally cause exceptions**. It allows you to signal that an error or exceptional condition has occurred in your code such as with validation code or to **re-raises the exception and send it to a higher-level of your code, in our case the main module**.

The **raise** statement is a built-in part of Python's exception handling, allowing you to manage and communicate errors to other parts of your code. It is **often associated with the separation of concerns since you can use it to pass error handing from one layer to another**.

Here is an example from our code:

*# TODO: Remove Dependency on presentation\_classes (Done)*except FileNotFoundError: *# as e:  
 # pres.IO.output\_error\_messages("Text file must exist before running this script!", e)* raise FileNotFoundError("Text file must exist before running this script!")  
except Exception: *# as e:  
 # pres.IO.output\_error\_messages("There was a non-specific error!", e)* raise Exception("There was a non-specific error!")  
return student\_data

In this code, **the raise statement is used to re-raise exceptions** after they have been caught and possibly handled. Inside the try block, we attempt to read data from a file which may raise different exceptions, including FileNotFoundError and more generic Exception.

When a FileNotFoundError is raised, it is caught in the first except block and raised again, **with our custom error message**, as the same FileNotFoundError exception.

Similarly, if any other exception (an instance of Exception) occurs, it is caught in the second except block and re-raised.

**Those exception objects are caught by the try-except block in the main module** that called the read\_data\_from\_file() method, and passed on to the output\_error\_messages() function in the Presentation layer.

try:  
 students = proc.FileProcessor.read\_data\_from\_file(file\_name=data.FILE\_NAME,  
 student\_data=data.students,  
 student\_type=data.Student)  
except FileNotFoundError as e:  
 pres.IO.output\_error\_messages(e)  
except Exception as e:  
 pres.IO.output\_error\_messages(e)

By re-raising the exception, you allow higher-level code or a broader error-handling mechanism to catch and handle the exception as needed. This approach is particularly useful when you want to handle exceptions at different levels of your program and provide specific error messages or actions for each type of exception.

## Demo/Video Demo01: Using Continuous Improvement

# Improving Documentation

Improving documentation is essential for maintaining and enhancing the understandability and maintainability of code, libraries, or projects. Before diving into writing documentation, outline what needs to be documented, who the target audience is, and the purpose of the documentation. Having a plan helps you stay organized and focused.

Here are some tips to help you improve your documentation:

**Use a Consistent Style and Format:** Establish a consistent style guide for your documentation. Consistency in formatting, naming conventions, and terminology makes it easier for readers to follow and understand.

**Write Clear and Concise Content:** Aim for clarity and brevity. Use plain language and avoid unnecessary jargon or technical terms that may confuse your audience. Keep sentences and paragraphs short.

**Provide Context and Examples:** Explain the purpose and context of the code or feature being documented. Include usage examples and code samples to illustrate how to use the code effectively.

**Include Tutorials and Getting Started Guides:** Create step-by-step tutorials or getting started guides to help new users understand how to use your code or project. Provide a clear path for beginners to get up and running.

**Update Documentation Regularly:** Keep your documentation up to date with the latest code changes. Outdated documentation can be misleading and frustrating for users.

**Version Your Documentation:** If your project has multiple versions, maintain separate documentation for each version and clearly indicate which version the documentation corresponds to. This helps users find the information relevant to their version.

**Provide Troubleshooting and FAQs:** Include a troubleshooting section or frequently asked questions (FAQs) to address common issues or challenges users may encounter.

**Ask for User Feedback:** Encourage users to provide feedback on your documentation. They can point out areas that need clarification or suggest improvements.

**Collaborate with Others:** If multiple contributors work on a project, encourage collaboration on documentation. Different perspectives can lead to more comprehensive documentation.

**Proofread and Review:** Before publishing or updating documentation, have someone else review it for accuracy, clarity, and completeness. Fresh eyes can catch errors or omissions.

**Document Code Internally:** Don't forget to write clear comments and docstrings within your code. Good internal documentation helps both you and other developers understand the code's logic.

**Be Patient and Iterative:** Documentation is an ongoing process. Start small, and gradually improve your documentation over time. It doesn't have to be perfect from day one.

**Seek Inspiration:** Look at well-documented projects or libraries for inspiration. Study their documentation to see what works well and adapt those practices to your own project.

**Use Proper Documentation Tools:** Choose appropriate documentation tools and formats. **Popular choices include Markdown**, reStructuredText, and tools like Sphinx for Python. Use tools that support version control systems like Git for easy maintenance.

Remember that good documentation is an **investment that pays off in the long run** by making your code more accessible, reducing support requests, and helping others collaborate effectively on your project.

## Markdown Language

Markdown is a lightweight markup language that **is commonly used** for formatting documents. It allows you to add structure and formatting to your text using simple and easy-to-read syntax. Markdown is widely **used for creating documentation, README files, blog posts, and more,** both in software development and other domains.

Markdown is a plain text formatting syntax that was created by John Gruber and Aaron Swartz in 2004. It is designed to be human-readable and easy to write, with a focus on simplicity. Markdown allows you to **add formatting elements to text without the need for complex markup or HTML tags**.

Some common elements that can be formatted using Markdown include headings, lists, links, images, emphasis (italic and bold text), code blocks, and more. Markdown files typically have a .md extension.

Markdown offers a variety of commands, often referred to as "syntax" or "elements," to format text and structure documents. While there are many **Markdown commands** available, **here are some of the most common and frequently used ones**:

**Headers:** To create headings, you can use hash (#) symbols. The number of hash symbols indicates   
the level of the heading (e.g., # for the largest heading, ## for a smaller heading, and so on).  
  
*# Heading 1  
## Heading 2  
### Heading 3***Emphasis:** To emphasize text, you can use asterisks (\*) or underscores (\_).

\*italic\* or \_italic\_ for italic text.  
\*\*bold\*\* or \_\_bold\_\_ for bold text.  
  
**Lists:** You can create ordered (numbered) and unordered (bulleted) lists.  
  
*Unordered list:*

- Item 1  
- Item 2  
 - Subitem A  
   
*Ordered list:*  
  
1. First item  
2. Second item  
  
**Links:** To create hyperlinks, use square brackets for the link text followed by parentheses for   
the URL.  
  
[Google](*https://www.google.com*)  
  
  
**Tables:** Markdown allows you to create simple tables using pipes (|) to separate columns and   
hyphens (-) to create headers.

| Header 1 | Header 2 |  
|----------|----------|  
| Cell 1 | Cell 2 |

**Images:** To embed images, use an exclamation mark (!) followed by square brackets for alt text   
and parentheses for the image URL.  
  
![Alt text](**https://example.com/image.jpg**)  
![The use cases of our program](**./images/use\_cases.png**)

These are some of the most used Markdown commands. Markdown's simplicity and readability make it a popular choice for creating formatted documents, README files, documentation, and content for various purposes. Depending on the Markdown processor or platform you use, additional features and extensions may be available.

### Using Markdown in PyCharm

PyCharm, like many other code editors and integrated development environments (IDEs), provides support for editing Markdown files. In PyCharm, you can create a new Markdown file by right-clicking on the project folder, selecting "New," and choosing "File." Then, give the file a .md extension.

Afterward, you open the Markdown file in the PyCharm editor and start writing or editing content using Markdown syntax.

PyCharm provides a built-in Markdown preview feature. You can open the preview pane to see how your Markdown content will be rendered. Look for an option in the editor to enable the Markdown preview.

A screenshot of a computer

Description automatically generated

Once you've edited your Markdown file, save the changes, and commit them to your version control system (e.g., Git) if your project is under version control.

You can use Markdown files in your project for documentation, READMEs, or other forms of content. When you share or publish your project, the Markdown content will be rendered as formatted text.

### Use Markdown on GitHub

GitHub supports Markdown for formatting content in repositories, README files, issues, pull requests, and more. When you create or edit content on GitHub, you can use Markdown to format it.

To use Markdown on GitHub repository, you can create or edit Markdown files directly. Simply write your content using Markdown syntax and save the changes.

In README files, Markdown is often used to provide information about the project, including usage instructions, documentation, and examples. GitHub provides a live preview of Markdown content, making it easy to see how your formatting will appear before you commit your changes.

A screenshot of a computer

Description automatically generated

## Demo/Video Demo02: Using Markdown

# Improving Analytics

A screenshot of a computer program

Description automatically generatedAnother improvement that we may not have thought of, but perhaps the QA team noticed, is the need for better analytics. While the application does record the data correctly, once we get a lot of data in the file, analyzing that data become cumbersome.

Since we have completed our two development tasks in the backlog, we get another and start working on it.

A screenshot of a computer

Description automatically generated

One of the most common and convenient tools available for python is Jupyter notebooks. Jupyter notebooks are a powerful and versatile tool in the Python tool that are widely used in various fields, including data science, machine learning, scientific research, and education. They provide an interactive and flexible environment for writing and executing code, as well as documenting and visualizing your work.

A screenshot of a computer

Description automatically generated

## Jupyter Notebooks

Jupyter notebook is an open-source web application that allows you to create and share documents that contain live code, equations, visualizations, and narrative text in a "Notebook" file. Each Jupyter notebook enables you to write and execute code in a step-by-step manner. This interactivity is particularly beneficial for learning and experimentation.

For example, you can include various types of outputs in a notebook, such as text, images, charts, and interactive widgets. This makes it a great tool for data analysis and visualization, and also documenting your work, since you can add detailed explanations, comments, and markdown text alongside your code to provide context and insights into your analysis.

Jupyter notebooks can be easily shared with others. You can export them to different formats, including HTML, PDF, and slides, making it simple to communicate your findings and insights.

### Installing Jupyter Notebooks

You can use Jupyter notebooks on the internet or by installing it on your own computer. The installation is simple and quick using the package installer program known as "pip."

The pip package manager for Python allows you to easily install, manage, and uninstall Python packages (collections of libraries and modules) from the Python Package Index (PyPI) and other package repositories. It simplifies the process of installing third-party libraries and packages, making it an essential tool for Python developers.

Here are some key points about pip:

**PyPI:** The Python Package Index (PyPI) is a central repository that hosts thousands of open-source Python packages contributed by the Python community. pip primarily fetches packages from PyPI, but you can also specify other package sources if needed.

A screenshot of a computer

Description automatically generated

**Installing Packages:** You can use pip to install Python packages by running a simple command, such as **pip install package\_name**. For example, to install the requests library, you would run pip install requests. pip will automatically download the specified package and its dependencies from PyPI and install them on your system.

A screenshot of a computer

Description automatically generated

**Managing Packages**: pip provides commands to list installed packages (pip list or pip freeze), upgrade packages (**pip install --upgrade package\_name**), uninstall packages (pip uninstall package\_name), and search for packages (pip search query). This makes it easy to keep your Python environment up to date and well-organized.

**Package Distribution:** Developers can use pip to package and distribute their own Python packages on PyPI or other repositories. The setuptools library and the setup.py script are often used in conjunction with pip for package distribution.

**Cross-Platform:** pip is cross-platform and works on Windows, macOS, and Linux, making it a versatile choice for managing Python packages on different operating systems.

**Virtual Environments**: It is common practice to use pip in conjunction with Python virtual environments. Virtual environments **allow you to create isolated Python environments for different projects**, ensuring that packages and dependencies do not conflict with each other.

Overall, pip is an essential tool in the Python ecosystem, enabling developers to easily access and manage a vast array of Python packages to extend the functionality of their Python projects.

To install, Jupyter Notebook in a PyCharm project's virtual environment, you open a terminal window, type in the install command, and press enter. Afterward the pip will connect to the PyPi website, locate the package of modules, then start downloading and install automatically.

A screenshot of a computer

Description automatically generated

***Note:*** *If you do this in a PyCharm it will only be installed for that project's virtual environment. Which is fine, but you will have to install it again in a different project's environment.*

To start a Jupyter Notebook after installation by first opening the Terminal, then running the following command:

jupyter notebook

A screenshot of a computer

Description automatically generated

After running this command, you should see an output in the terminal indicating that Jupyter Notebook is running, and it will then open a web browser window (or a new tab in an existing browser window) displaying the Jupyter Notebook interface.

The webpage interface is run on a mini-web server that was installed as part of the PyPi Jupyter Notebook package, which is why it displays as a webpage instead of a common windowed application. On this web page, you will see the Jupyter Notebook dashboard, which allows you to create new notebooks, open existing ones, and navigate your file system.

A screenshot of a computer

Description automatically generated

You can create a new notebook by clicking the "New" button and selecting a Python kernel.

A screenshot of a computer

Description automatically generated

Once you have a notebook open, you can add code cells, markdown cells, and start writing and running Python code right within the notebook interface.

Remember that while Jupyter Notebook is running, you can access it through the web browser interface. You can also open multiple notebooks simultaneously in different tabs or browser windows.

When you're done working with Jupyter Notebook, you can shut it down by going back to the terminal where it's running and pressing Ctrl + C. It will ask you to confirm shutting down the notebook server.

***Important: If you close the terminal*** *where Jupyter Notebook was started,* ***it will also shut down the notebook web server****, and you won't be able to access it until you start it again by running Jupyter notebook in a new terminal session.*

### Using Jupyter Notebooks

Notebooks are organized into cells, which can contain either code (code cells) or text explanations (markdown cells). This structure makes it easy to separate code from documentation and helps in code modularity.

A screenshot of a computer

Description automatically generated

To create a simple bar chart in a Jupyter Notebook using our data, we can use Python libraries like Matplotlib or Seaborn for data visualization.

Here is an example using Matplotlib to create a bar chart that represents the distribution of GPAs in our data.

1. First, we need to install Matplotlib if you haven't already. You can do this using pip:

pip install matplotlib

A screenshot of a computer

Description automatically generated

***Important:*** *Open a second terminal window if you need to, so that you can keep the Jupyter notebook window open and running.*

2. Once Matplotlib is installed, we can create a bar chart with the following code in a Jupyter Notebook cell:

import matplotlib.pyplot as plt  
import json  
  
*# Extract GPAs from the data*with open("./Demo03-UsingJupyterNotebooks/GPAData.json", "r") as file:  
 list\_of\_dictionary\_data = json.load(file)  
  
gpas = [entry["GPA"] for entry in list\_of\_dictionary\_data]  
  
*# Count the occurrences of each GPA value*gpa\_counts = {gpa: gpas.count(gpa) for gpa in set(gpas)}  
  
*# Sort the GPA values for plotting*sorted\_gpas = sorted(gpa\_counts.keys())  
  
*# Corresponding counts for the sorted GPAs*count\_values = [gpa\_counts[gpa] for gpa in sorted\_gpas]  
  
*# Create the bar chart*plt.figure(figsize=(6, 4))  
plt.barh(sorted\_gpas, count\_values, align='center', alpha=0.7) *# Use barh for horizontal bars*plt.xlabel('Count')  
plt.ylabel('GPA')  
plt.title('Distribution of GPAs')  
plt.yticks(sorted\_gpas)  
plt.grid(axis='x', linestyle='--', alpha=0.7) *# Adjust grid for horizontal bars*plt.show()

This code will create a horizontal bar chart displaying the distribution of GPAs from the provided data. The x-axis will represent the GPA values (1, 2, 3, 4), and the x-axis will represent the count of each GPA value in the dataset.

A screenshot of a computer program

Description automatically generated

Matplotlib is a versatile Python library for creating a wide range of charts and plots. Here are some of the most commonly used types of charts and plots you can create with Matplotlib:

* Line charts are used to display data points connected by straight lines. They are suitable for showing trends and changes over time.
* Scatter plots are used to visualize the relationship between two continuous variables. Each data point is represented as a marker on the plot.
* Bar charts display data as rectangular bars with lengths proportional to the values they represent. They are useful for comparing categories or showing distributions.
* Similar to bar charts, horizontal bar charts represent data with horizontal bars. They are useful when you have long category labels.
* Histograms are used to represent the distribution of a single variable. They divide the data into bins and display the frequency of data points in each bin.
* Pie charts display data as sectors of a circle, where each sector represents a portion of the whole. They are useful for showing parts of a whole.

Matplotlib provides extensive documentation and examples, making it easy to explore and create various types of charts and plots for your data visualization needs.

In conclusion, Jupyter notebooks are a fundamental tool for Python programmers and data scientists. They offer an interactive, collaborative, and versatile environment for writing and running code, documenting work, and sharing insights. Knowing how to use Jupyter notebooks is a valuable skill in both the world of Python programming and data analysis.

## Creating Charts in PyCharm

You can create charts like the horizontal bar chart using Matplotlib, in PyCharm using the same code as in Jupyter notebook. Once you have the code in your Python script and Matplotlib is installed, you can run the script and a chart output will be displayed.

A screenshot of a computer

Description automatically generated

## Demo/Video Demo03: Using Jupyter Notebook

# Summary

In this module, we looked at ways applications are updated by using Continuous Improvement and Agile methodologies. Our next improvement will be to move our data from our simple Json data file and into a database, but for that you will have to wait for module 10. See you there!

A screenshot of a computer

Description automatically generated

At this point, you should try and answer as many of the following questions as you can from memory, then review the subjects that you cannot. Imagine being asked these questions by a co-worker or in an interview to connect this learning with aspects of your life.

* What is Continuous Improvement?
* What is the purpose of Agile Methodologies?
* What is a Scrum?
* What is a Sprint?
* What is a Sprint Backlog?
* What are the names of some common Scrum Ceremonies?
* What is a Polymorphic Type?
* What is the raise statement?
* What is Markdown?
* Where is Markdown used?
* What is Jupyter Notebook?

When you can answer all of these from memory, it is time move on to the next module.