DEV questions !

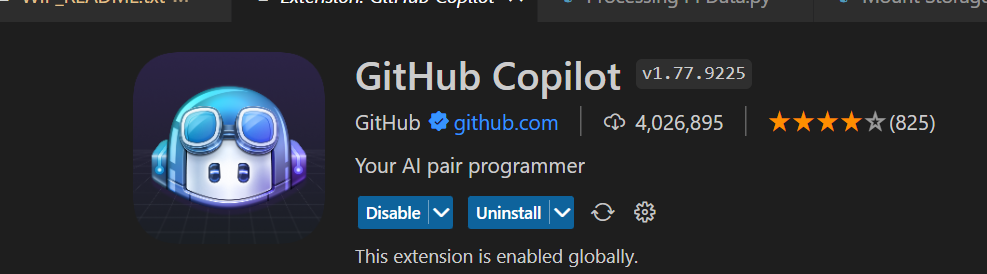
In programming environnement, there’s new things seemingly every week,

new version of .Net every year

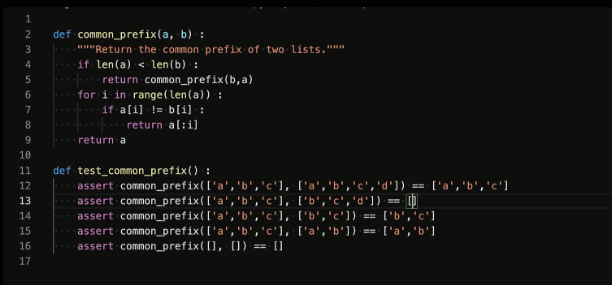
Python 3.6 is discontinued since decembre 2021, 3.8 used in our programs will be stopped on octobre 2024.

Visual Studio has new version every month

You can get exciting extensions.



Helps define new test functions



* Has ability to write assert statements !

Don’t stop learning when you’re a developer.

* It’s important to keep growing by improving!

🡺 7 things that works :

1. Not everything ! A goal at once.
2. List :
3. Prioritize, most important but not perhaps right now
4. Focus : take time, learn well and then move on when you’re confident about having learn good thing
5. Interrupting Topics : learn just enough to be able to set priority
6. It’s OK not to know everything
7. More efficient to follow proven steps

So what [chatGPT](https://chat.openai.com/chat) says:



Sure, here's a presentation of development best practices for data scientists working with Python:

Slide 1: Introduction

Title: Development Best Practices for Data Scientists using Python

Welcome to this presentation on development best practices for data scientists using Python. In this presentation, we will discuss some of the most important practices that data scientists can adopt to improve their productivity, efficiency, and code quality.

Slide 2: Use **Version Control** Title: Use Version Control

Version control is a fundamental best practice for any developer, and data scientists are no exception. By using a version control system, such as Git, you can keep track of changes to your code over time, collaborate with others more effectively, and roll back to previous versions if necessary.

Slide 3: Write **Modular** Code Title: Write Modular Code

Writing modular code means breaking your code down into small, reusable components that can be easily understood and modified. By doing this, you can improve the readability of your code, reduce code duplication, and make it easier to maintain and extend in the future.

Slide 4: **Document** Your Code Title: Document Your Code

Documentation is key to making your code understandable and usable by others. It's important to include comments in your code that explain what it does, how it works, and why it's important. Additionally, you can use tools like Sphinx to generate API documentation automatically from your code.

Slide 5: **Test** Your Code Title: Test Your Code

Testing your code is essential to ensure that it works as expected and doesn't introduce bugs. You can write tests using a testing framework like pytest, which allows you to define test cases and assertions. By doing this, you can catch errors early and ensure that your code is reliable and correct.

Slide 6: Use **Code Reviews** Title: Use Code Reviews

Code reviews are a great way to get feedback on your code from your peers. By having others review your code, you can identify potential issues, improve the quality of your code, and learn from others' experiences. Tools like GitHub and GitLab make it easy to conduct code reviews and collaborate with others.

Slide 7: **Optimize** Your Code Title: Optimize Your Code

Optimizing your code means making it run faster and more efficiently. This can involve using algorithms and data structures that are more efficient, profiling your code to identify bottlenecks, and using libraries like NumPy and Pandas that are optimized for numerical operations. By doing this, you can improve the performance of your code and reduce the time it takes to run.

Slide 8: Conclusion Title: Conclusion

In conclusion, these development best practices can help data scientists write better code, collaborate more effectively with others, and improve the quality and reliability of their work. By adopting these practices, you can become a more efficient and effective data scientist and improve the overall quality of your work.

In Optimize I think also “**Design patterns**” :

There are many useful design patterns in Python that can help developers write cleaner, more maintainable code. Here are some of the most common design patterns in Python:

1. **Singleton** pattern - This pattern ensures that only one instance of a class exists throughout the application's lifetime. This can be useful when you need to manage global state or resources that should only be instantiated once.
2. **Factory** pattern - This pattern provides a way to create objects without specifying their exact class. This can be useful when you want to create multiple objects that share a common interface.
3. **Strategy** pattern - This pattern defines a family of interchangeable algorithms and encapsulates each one, allowing them to be used interchangeably. This can be useful when you need to switch between different algorithms based on certain conditions.
4. **Observer** pattern - This pattern defines a one-to-many relationship between objects, so that when one object changes state, all its dependents are notified and updated automatically. This can be useful when you need to monitor changes to objects and take action accordingly.
5. **Decorator** pattern - This pattern allows you to dynamically add functionality to an object by wrapping it in another object that provides the additional behavior. This can be useful when you need to add features to an object without modifying its code directly.
6. **Template** method pattern - This pattern defines the skeleton of an algorithm in a base class, with specific steps implemented in subclasses. This can be useful when you want to provide a standard framework for a process, but allow for customization of certain steps.
7. **Iterator** pattern - This pattern provides a way to access the elements of an object sequentially without exposing its underlying representation. This can be useful when you want to iterate over a collection of objects in a standardized way.

These are just a few of the many design patterns that can be used in Python. It's important to remember that design patterns should be used judiciously, and only when they are appropriate for the specific problem you are trying to solve.



explain Docker to data scientists!

Docker is a platform that allows developers to package, distribute, and run applications in a containerized environment. A container is a lightweight, standalone executable package of software that contains everything needed to run an application, including code, libraries, and system tools.

Here are some key concepts related to Docker:

1. Docker image: A Docker image is a read-only template that contains the instructions for creating a container. It includes all the necessary files and dependencies needed to run the application.
2. Docker container: A Docker container is a lightweight, standalone, and executable package of software that runs in a containerized environment. It is created from a Docker image and contains the application code, libraries, and other dependencies.
3. Dockerfile: A Dockerfile is a text file that contains a set of instructions for creating a Docker image. It specifies the base image to use, the files to include, the software to install, and other configuration details.
4. Docker registry: A Docker registry is a storage system for Docker images. It allows developers to store and distribute images to other team members or across different environments.

Now, here are some reasons why data scientists might find Docker useful:

1. Reproducibility: Docker makes it easy to create a standardized environment for running data science projects. This can help ensure reproducibility of results, as all team members are working in the same environment.
2. Portability: Docker containers can be easily moved between different environments, such as development, testing, and production. This makes it easier to deploy data science models and applications across different environments.
3. Collaboration: Docker images can be shared and reused across different teams and projects, making it easier to collaborate on data science projects.
4. Isolation: Docker containers provide isolation between applications and environments, which can help prevent conflicts and ensure that different applications and projects are running smoothly.

Overall, Docker is a powerful tool for data scientists that can help improve the reproducibility, portability, collaboration, and isolation of data science projects.

Haut du formulaire



**Flake8** is a tool for checking Python code for style and quality issues. It combines three different tools - **pycodestyle** (formerly known as **pep8**), **pyflakes**, and **McCabe** - to provide a comprehensive set of checks for Python code.

Here are some key concepts related to Flake8:

Pycodestyle: Pycodestyle (formerly known as pep8) is a tool that checks Python code against the PEP 8 style guide. It checks for things like line length, indentation, and naming conventions.

Pyflakes: Pyflakes is a tool that checks Python code **for errors and potential issues**. It checks for things like undefined variables, unused imports, and syntax errors.

McCabe: McCabe is a tool that checks Python code for **complexity**. It calculates a "cyclomatic complexity" score for each function, which is a measure of how many different paths through the code there are.

Now, here are some reasons why data scientists might find Flake8 useful:

**Consistency**: Flake8 can help ensure that code is written consistently across a project, which can make it easier to read and maintain.

**Quality**: Flake8 can help catch errors and potential issues before they cause problems in production. This can improve the overall quality of the code.

**Efficiency**: Flake8 can help save time by catching errors and potential issues before they are discovered during testing or in production. This can help prevent delays and reduce the need for rework.

Best Practices: Flake8 can help enforce best practices and **coding standards** within a team, which can lead to better overall code quality and more efficient collaboration.



Sure, I'd be happy to explain SonarQube to a development team!

SonarQube is an open-source platform for code quality management that provides automated code analysis and testing for a variety of programming languages, including Java, C++, Python, and more. It helps development teams ensure that their code is reliable, efficient, and maintainable by identifying code smells, bugs, vulnerabilities, and other quality issues.

Here are some key concepts related to SonarQube:

1. Code analysis: SonarQube uses static code analysis to detect code smells, bugs, vulnerabilities, and other quality issues. It analyzes code without actually running it, which allows it to detect issues that may not be apparent during testing.
2. Quality gates: SonarQube provides a set of predefined quality gates that can be customized to suit the needs of a particular project. Quality gates are used to define and enforce quality standards for a project, such as code coverage, complexity, and maintainability.
3. Integrations: SonarQube integrates with a variety of development tools, including IDEs, build systems, and code repositories. This allows it to provide continuous feedback and analysis throughout the development process.
4. Dashboards: SonarQube provides interactive dashboards that allow development teams to view and analyze code quality metrics. This can help identify trends and areas for improvement.

Now, here are some reasons why a development team might find SonarQube useful:

1. Quality control: SonarQube provides automated code analysis and testing that can help ensure that code is reliable, efficient, and maintainable. This can improve the overall quality of the codebase.
2. Continuous feedback: SonarQube provides continuous feedback throughout the development process, which can help catch issues early and reduce the need for rework.
3. Customizable quality gates: SonarQube allows development teams to customize quality gates to suit the needs of their project. This can help enforce coding standards and best practices.
4. Improved collaboration: SonarQube provides interactive dashboards that allow development teams to view and analyze code quality metrics. This can improve collaboration by providing a shared view of the codebase.

Overall, SonarQube is a powerful tool for development teams that can help improve the quality, efficiency, and maintainability of code.



Sure! Here's an example pre-commit hook using Flake8 to check for code quality issues before committing:

1. Install Flake8:

pip install flake8

1. Navigate to the repository's **.git/hooks** directory:

cd /path/to/repo/.git/hooks/path/to/repo/.git/hooks

1. Create a new file called **pre-commit**:

touch pre-commitpre-commit

1. Open the **pre-commit** file in a text editor and add the following code:

#!/bin/sh

# Run Flake8 on all modified Python files

git diff --cached --name-only --diff-filter=ACMR "\*.py" | xargs flake8

# If Flake8 finds any issues, exit with a non-zero status code

if [ $? -ne 0 ]; then

echo "Flake8 found issues. Please fix them before committing."

exit 1

fi

This code uses **git diff** to get a list of all modified Python files that are about to be committed, and then runs Flake8 on those files. If Flake8 finds any issues, the script exits with a non-zero status code, preventing the commit from being created.

1. Make the **pre-commit** file executable:

chmod +x pre-commit +x pre-commit

That's it! Now, every time a developer tries to commit changes to a Python file, Flake8 will automatically check for issues and prevent the commit if any issues are found. This can help catch potential issues early and improve the overall quality of the codebase.



Let's say you're writing a function to calculate the average of a list of numbers. Here's an implementation:

def average(numbers):

sum = 0

count = 0

for number in numbers:

sum += number

count += 1

return sum / countpythonCopy code

/ count

Here are some best practices that were used in this implementation:

1. Use descriptive variable names: Instead of using a single letter variable name like "s" or "c", we used "sum" and "count" to make the code more readable.
2. Use whitespace: We added whitespace around operators and after commas to make the code easier to read.
3. Use comments: We didn't use comments in this example, but it's often a good idea to add comments to explain what the code is doing, especially for more complex functions.
4. Use consistent formatting: We used consistent formatting throughout the function, which makes it easier to read and understand.
5. Use error handling: We didn't add error handling to this function, but it's often a good idea to add it to ensure that the function behaves correctly even if unexpected input is provided.

These best practices help make the code more readable, maintainable, and less error-prone.

Haut du formulaire

There’s a wide variety of tasks and way to adapt,

But stay open can lead to success :

[GitHub Codespaces and GitHub Copilot: 2 stories, thousands of possibilities - Microsoft Community Hub](https://techcommunity.microsoft.com/t5/educator-developer-blog/github-codespaces-and-github-copilot-2-stories-thousands-of/ba-p/3740811)

Vs Code 

Use a powerful editor with useful addons and share with others your tricks 😉

* + Example (JFO 10/03/2023) : Markdown editing

Use Ctrl+Shift-V For previewer ([Build an Amazing Markdown Editor Using Visual Studio Code and Pandoc | thisDaveJ](https://thisdavej.com/build-an-amazing-markdown-editor-using-visual-studio-code-and-pandoc/))

Use markdownlint to detect markdown errors :

