Mohamed Walid Shaker

**When to use Conda and when to use pip**

Conda and pip are both package management tools in Python, but they serve different purposes and are best suited for different scenarios. Here's when to use Conda and when to use pip:

1. **Use Conda When**:
   * **Managing Environments**: Conda is particularly useful for creating and managing isolated Python environments. If you need to set up a clean, isolated environment with specific versions of Python and packages, Conda is a good choice. You can create environments with specific dependencies, which helps avoid conflicts between packages.
   * **Cross-Platform Compatibility**: Conda is platform-agnostic, which means it can handle package management and environment creation across various operating systems (Windows, macOS, Linux). This can be beneficial when you need to ensure that your code and dependencies work consistently on different platforms.
   * **Non-Python Dependencies**: Conda can manage not only Python packages but also non-Python libraries and dependencies. This makes it suitable for projects that require integration with non-Python tools or libraries.
   * **Complex Package Dependencies**: When your project has complex dependencies with a mix of Python and non-Python packages, Conda's ability to manage both types of dependencies can simplify the setup.
2. **Use pip When**:
   * **Installing Python Packages**: Pip is the default package manager for Python, and it's excellent for installing Python packages from the Python Package Index (PyPI). If you only need to install Python libraries, pip is usually sufficient.
   * **Quick Package Installation**: Pip is straightforward and easy to use for installing individual Python packages. It's a good choice when you need to quickly add a package to your Python environment.
   * **Virtual Environments**: While Conda can manage environments, if you only need to create virtual environments and don't require complex dependency management or non-Python packages, you can use Python's built-in **venv** module along with pip for a simpler solution.
   * **Integration with Other Python Tools**: If you are using other Python tools like **virtualenv**, **pyenv**, or **pipenv**, using pip might be more aligned with your workflow.