Getting Started with Instrument Control Using Python 3 and PyVISA

Introduction

Are you tasked with automating the setup and control of instruments for a test sequence? Python 3 with VISA provides a powerful combination for test automation in a user-friendly way. VISA includes the necessary code interface between the test software on your PC and the hardware in the box. The packaged VISA libraries provide standards for configuring, programming, and troubleshooting instrument systems that use a variety of communication interfaces. Python is a simple yet powerful language and is easy for new users to learn. Python gives a user the ability to write complex scripts that capitalize on VISA's libraries to automate your testing. The IDE, Visual Studio Code, provides users error checking, code completion, and other tools to simplify the coding process in an easy-to-use GUI. The following document includes everything a beginner user needs to start the automation process, including:

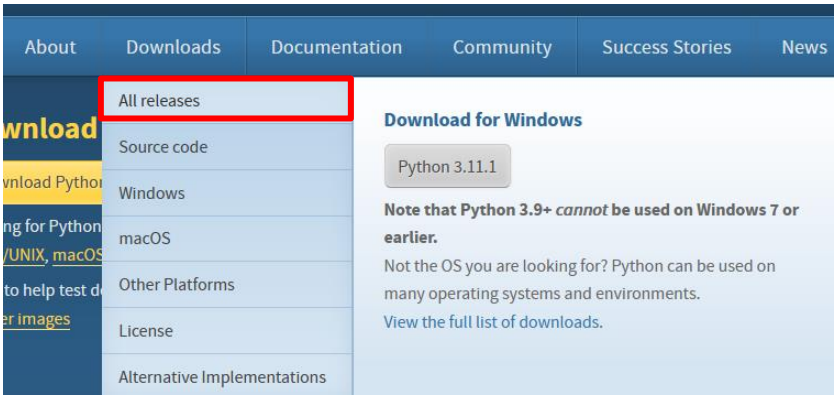
- Instructions for downloading and installing Python 3, Visual Studio Code, NI VISA, and PyVISA in a Windows environment.
- Importing VISA packages into your program.
- Steps for adding the VISA reference to your program.
- An overview of the instrcomms module to communicate with an instrument
- Building and running your simple instrument-controlling Python application.

This guide will not cover good coding practices and Python syntax in detail. For more information on coding in Python, please reference https://www.python.org/.

Contents

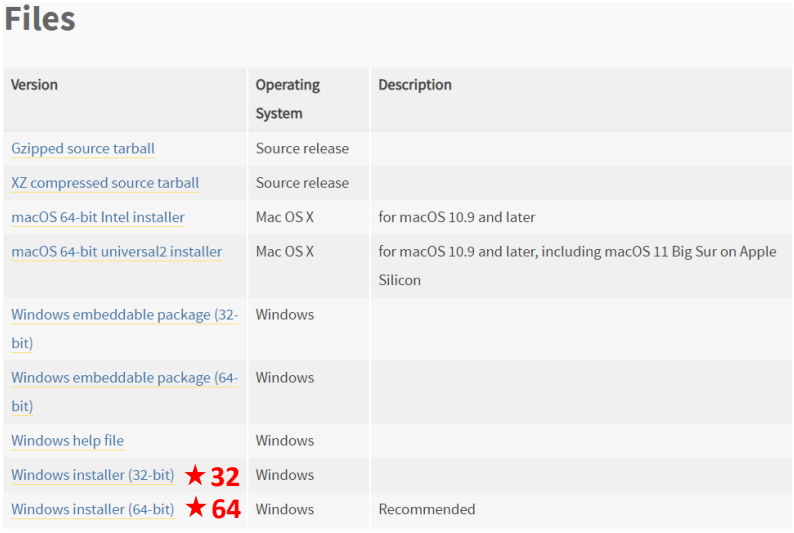
Installing Python 3

1. Go to https://www.python.org/ and click **Downloads**, then **All Releases**.

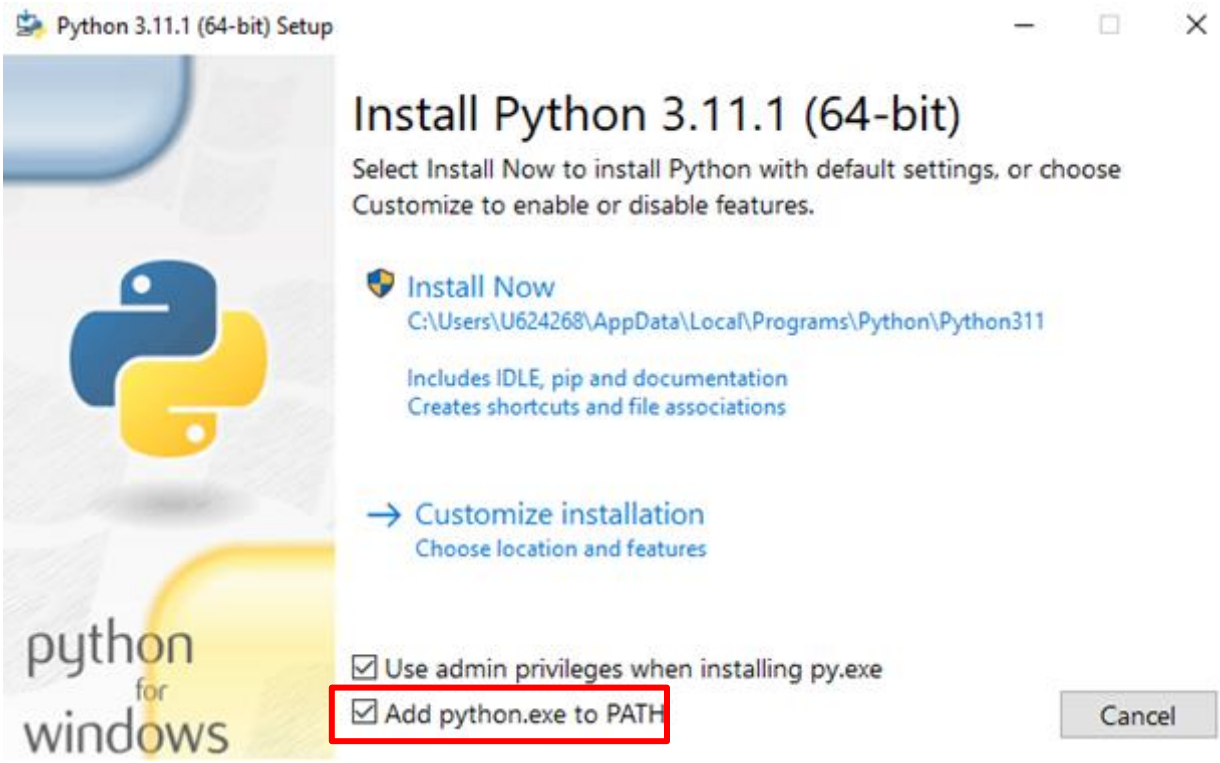


Click on the link for the latest stable release of **Python 3.11**. Earlier versions are acceptable if supported by PyVISA.

2. At the bottom of the page under **Files**, download either **Windows installer (64-bit)** for a 64-bit system, or **Windows installer (32-bit)** for a 32-bit system. The 64-bit version was selected for this example. If Windows asks you to **Run** or **Save** the program, select **Run**.



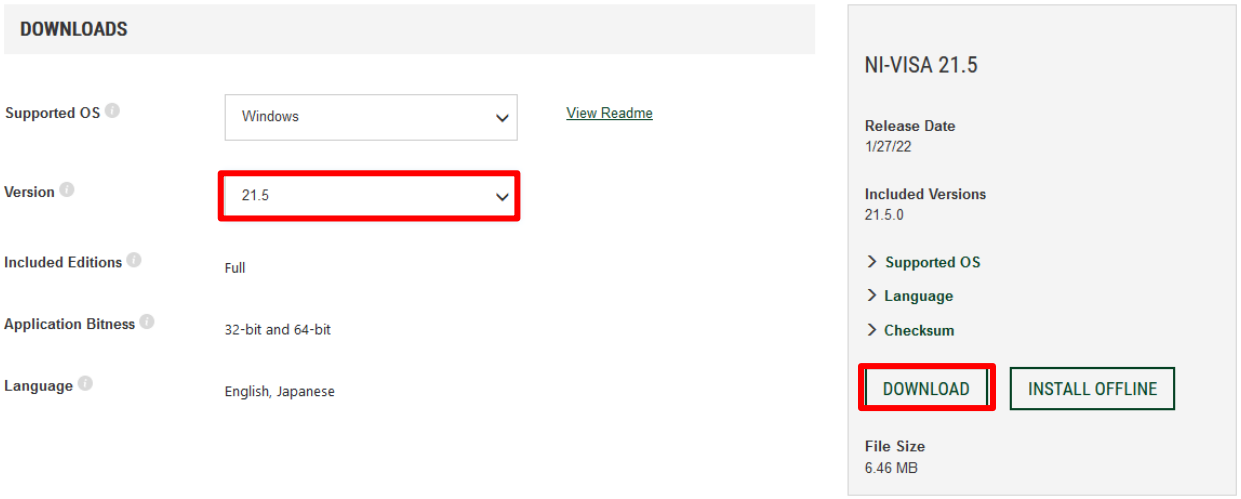
3. Run the downloaded executable and allow it to make changes to your PC. The Python Installer will open. If this is the first time Python is installed on the system or you intend to primarily use this version of Python, check the box at the bottom to **Add Python 3.11** (or your installed version) **to PATH**. Click **Install Now** to start the installation process.



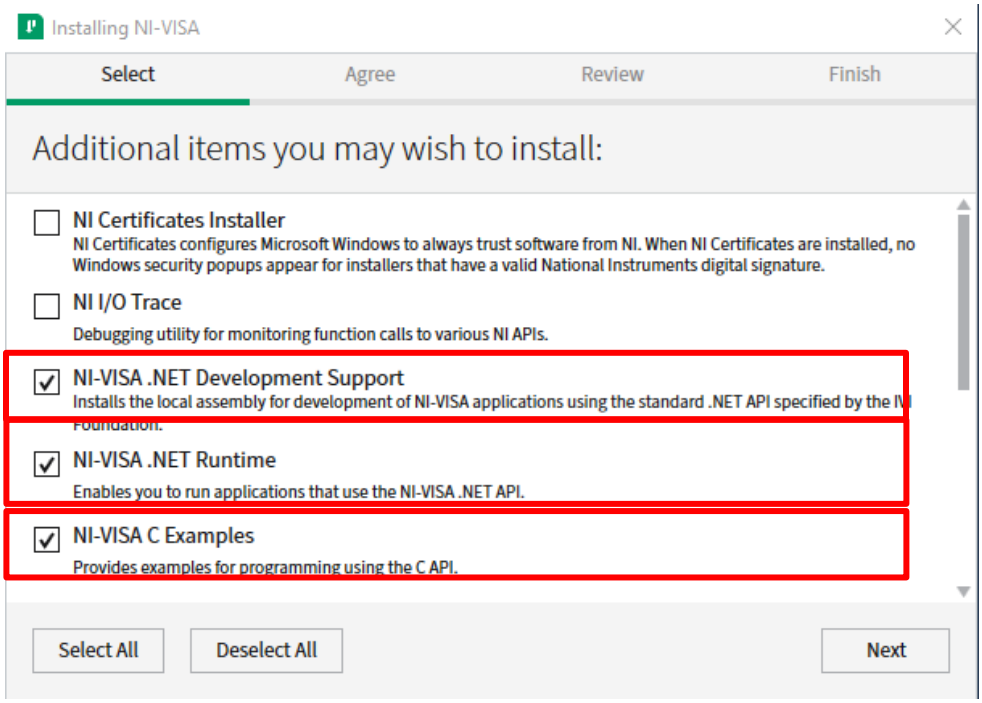
4. Once the process is successful, close the installer. If you intend to install other versions of Python, install them without adding to PATH unless they will replace the current version as your primary Python interpreter.

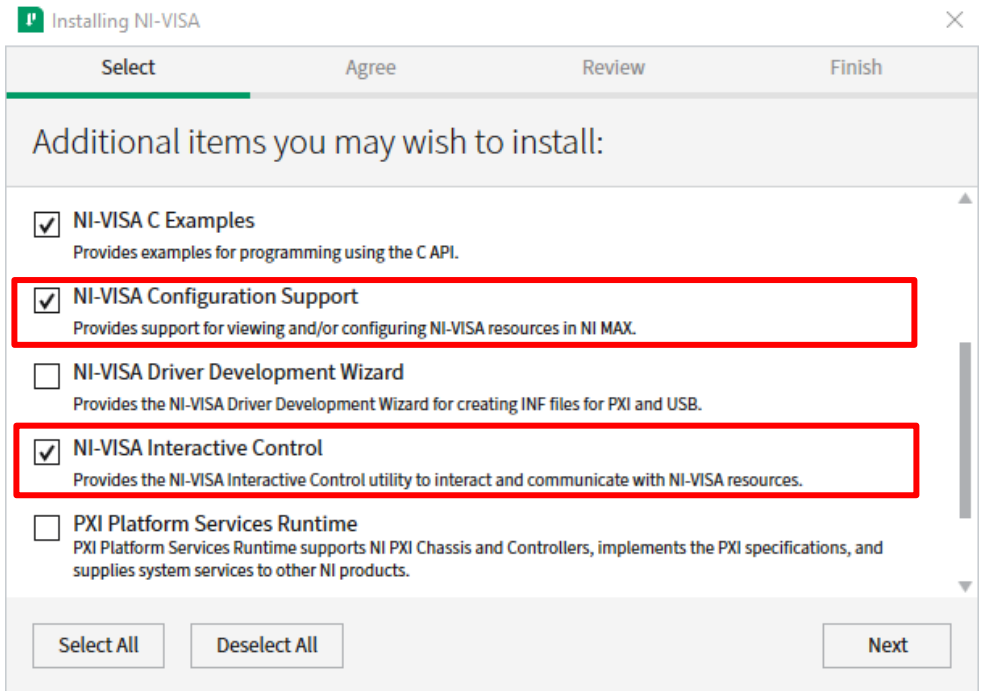
Installing National Instruments VISA

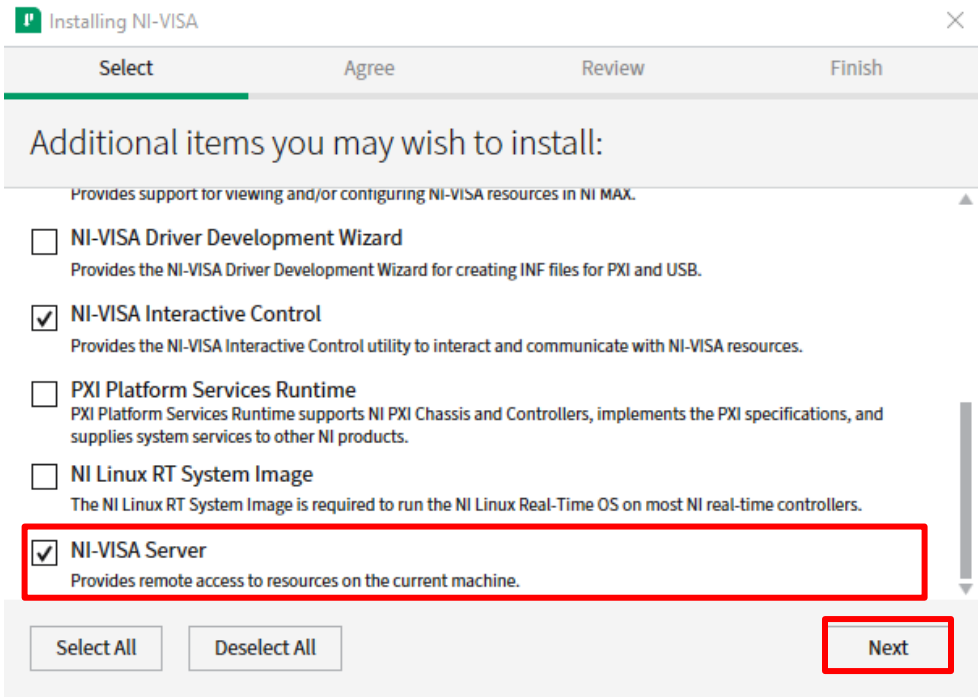
1. Go to https://www.ni.com/en-us/support/downloads/drivers/download.ni-visa.html#442805.
2. Select your operating system and version 21.5 (there are other versions available, but currently Keithley is recommending Version 21.5). Click **Download**.



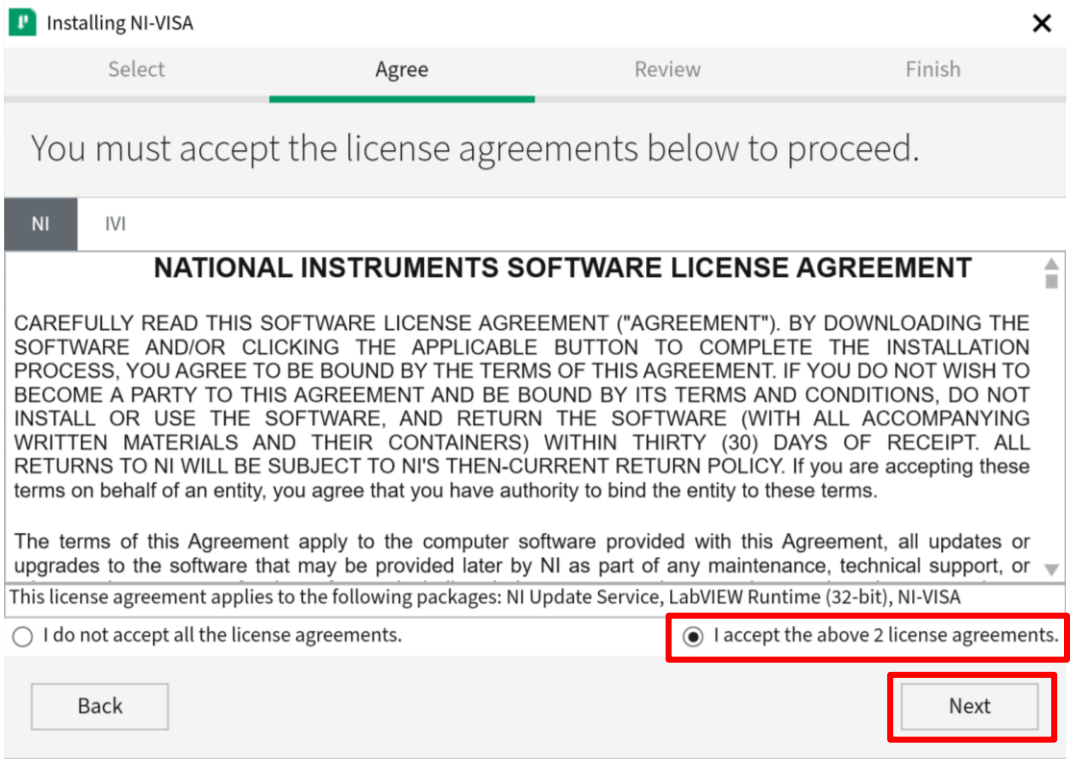
3. If Windows asks you to Open or Save the file, select **Save** or **Save As**. The file will download as an "exe" file. Find the file "**ni-visa_21.5_online.exe**" in your downloads folder, or the location to which you saved it, and click the file to run the installer.
4. If NI software has not been previously installed on the system, the installer may begin installing the NI Package Manager first. Agree to the terms, review, and install this first, then you will reach the next step.
5. Allow the application to make changes. Once the installer opens, select the following additional items to install: **NI-VIS.NET Development Support, NI-VISA.NET Runtime, NI-VISA C Examples, NI-VISA Configuration Support, NI-VISA Interactive Control, NI-VISA Server**. Click **Next**.

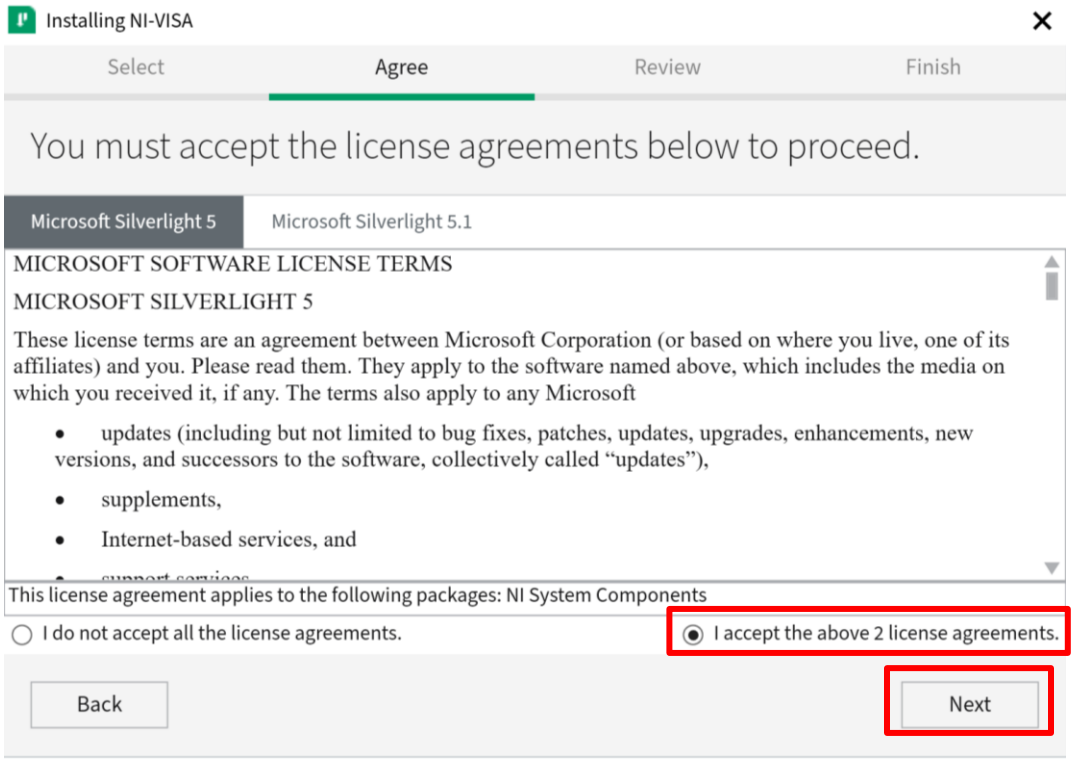

Installing NI-VISA
Select
Agree
Review
Finish
Additional items you may wish to install:
NI Certificates Installer
NI Certificates configures Microsoft Windows to always trust software from NI. When NI Certificates are installed, no Windows security popups appear for installers that have a valid National Instruments digital signature.
NI I/O Trace
Debugging utility for monitoring function calls to various NI APIs.
NI-VISA .NET Development Support
Installs the local assembly for development of NI-VISA applications using the standard .NET API specified by the IVI Foundation.
NI-VISA .NET Runtime
Enables you to run applications that use the NI-VISA .NET API.
NI-VISA C Examples
Provides examples for programming using the C API.
Select All
Deselect All
Next


Installing NI-VISA
Select
Agree
Review
Finish
Additional items you may wish to install:
NI-VISA C Examples
Provides examples for programming using the C API.
NI-VISA Configuration Support
Provides support for viewing and/or configuring NI-VISA resources in NI MAX.
NI-VISA Driver Development Wizard
Provides the NI-VISA Driver Development Wizard for creating INF files for PXI and USB.
NI-VISA Interactive Control
Provides the NI-VISA Interactive Control utility to interact and communicate with NI-VISA resources.
PXI Platform Services Runtime
PXI Platform Services Runtime supports NI PXI Chassis and Controllers, implements the PXI specifications, and supplies system services to other NI products.
Select All
Deselect All
Next

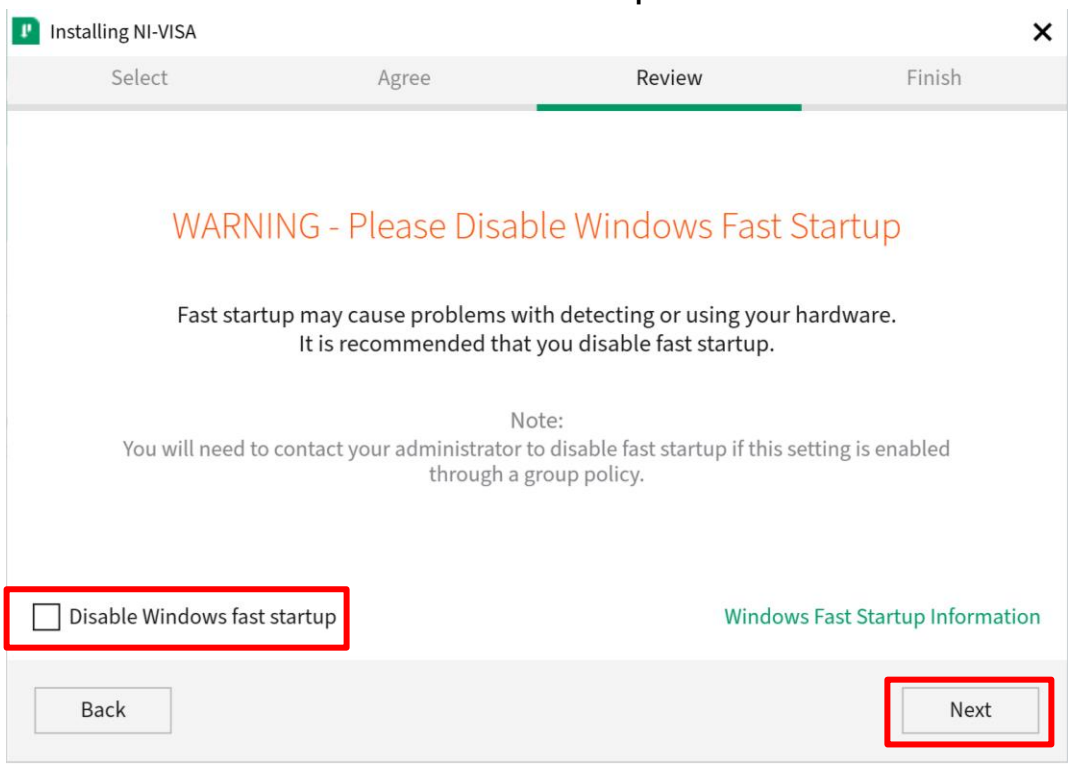


6. Accept all licenses. Click **Next** through both license windows.

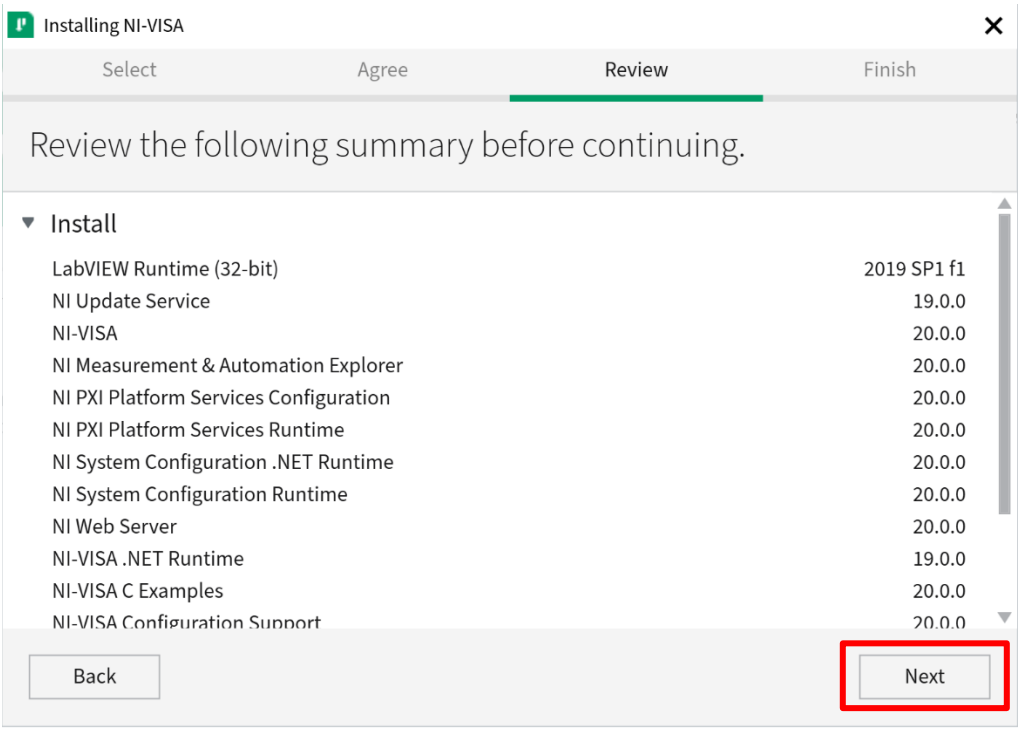




7. Uncheck the box labeled "**Disable Windows Fast Startup**." Click **Next**.

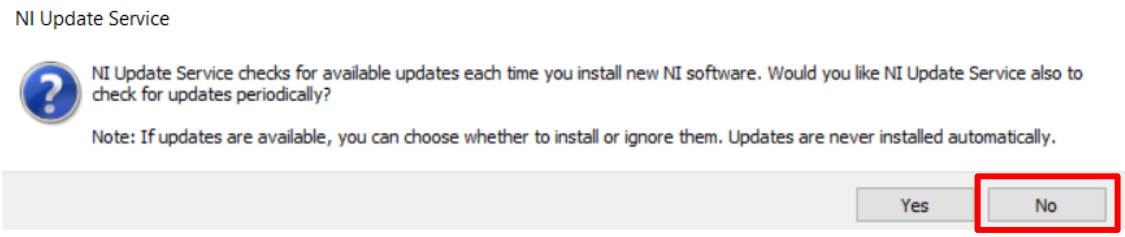


8. Accept all licenses. Click **Next** through both license windows.
9. Review software being installed. Click **Next**.



10. In the “Finish” tab, click **Next**. Click **No** in the update service dialogue box that appears to begin the installation.

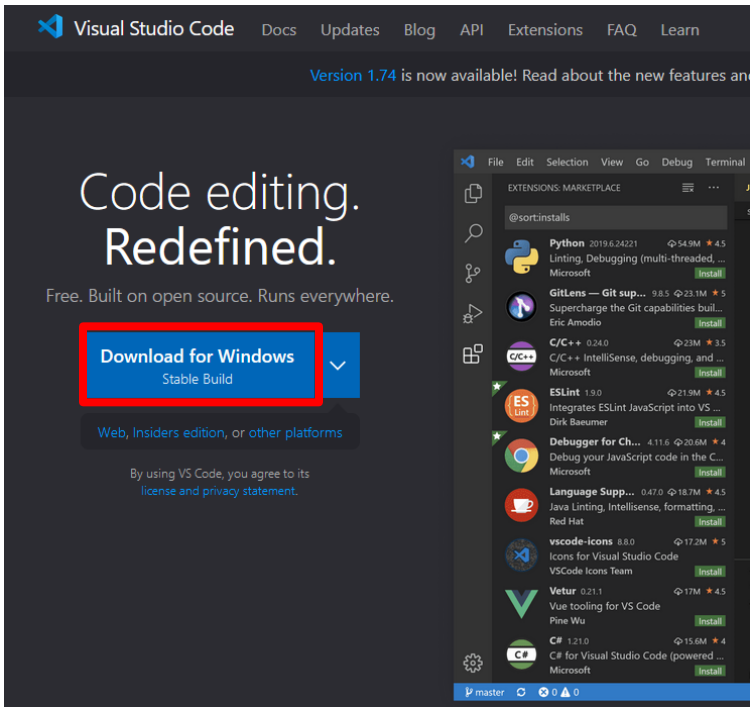
    NOTE: *The installation may take a while to finish.*



11. After the software is finished installing, allow your computer to restart to finish the process.

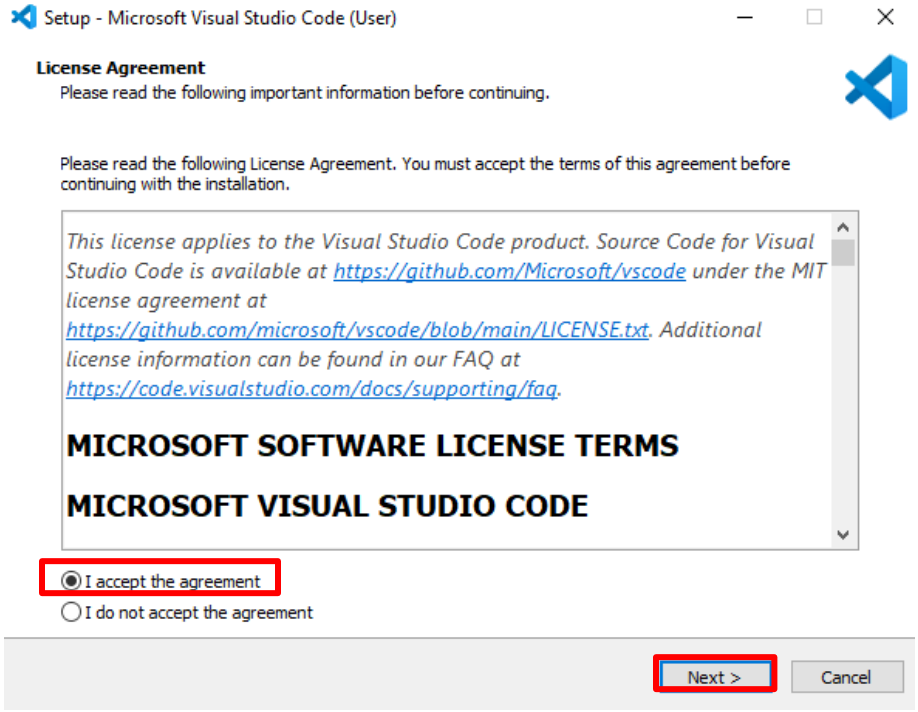
Installing Visual Studio Code

1. Go to https://code.visualstudio.com/. Click **Download for Windows** in the center of the screen.

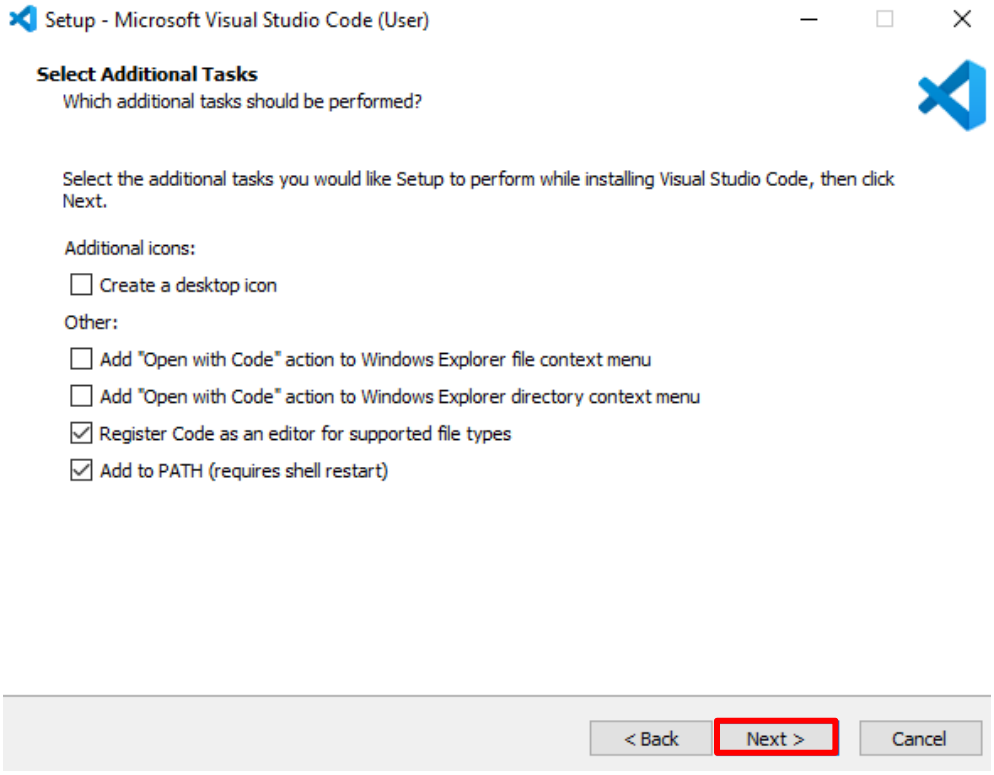


2. If Windows asks you to Run or Save the program, select Run. Allow it to make changes to your device.

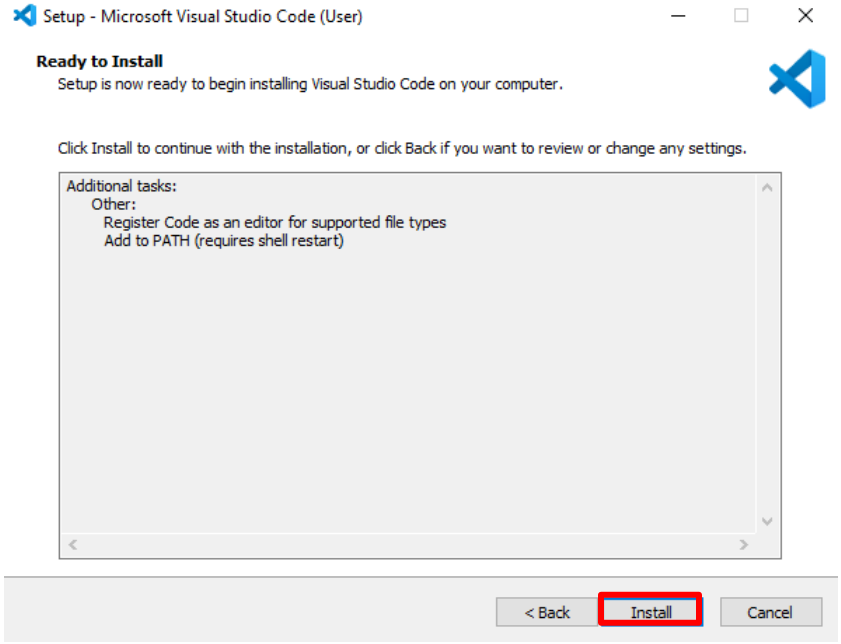
3. Once the Setup panel is open, accept the license agreement and click **Next** to continue the installation.



4. Select optional setup tasks if desired and click **Next**.

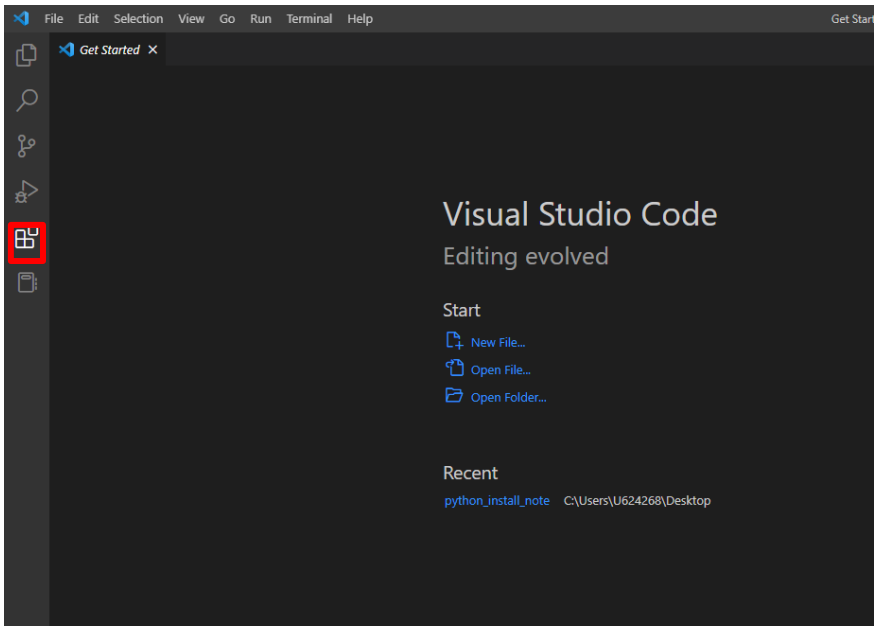


5. Click **Install** on the next window to start the installation.

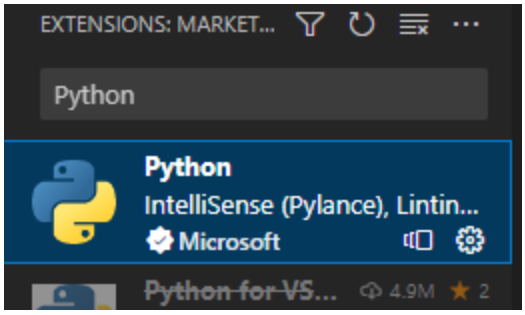


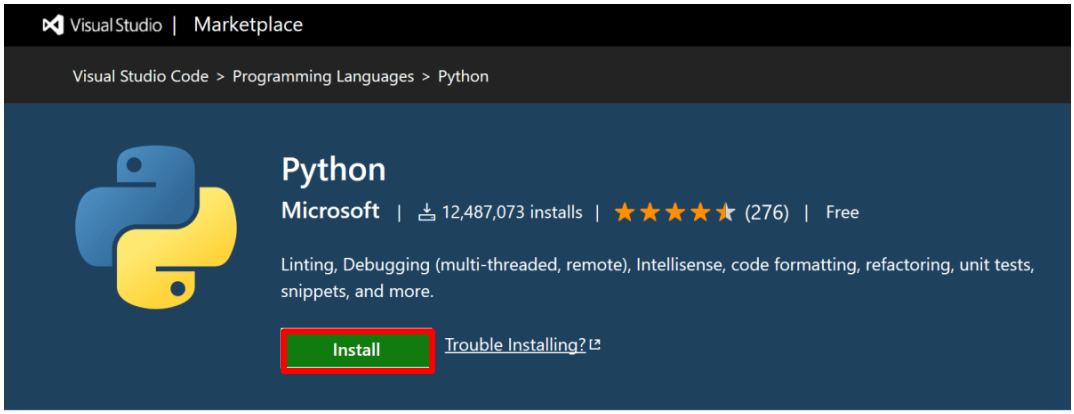
Install Visual Studio Code Extensions

1. Start Visual Studio Code from the Windows Start Button (or from the icon on your desktop if you selected to create a desktop shortcut during install). Click the **Extensions** button on the side panel.



2. Search for **Python** in the extensions marketplace and select it, then click **Install**.





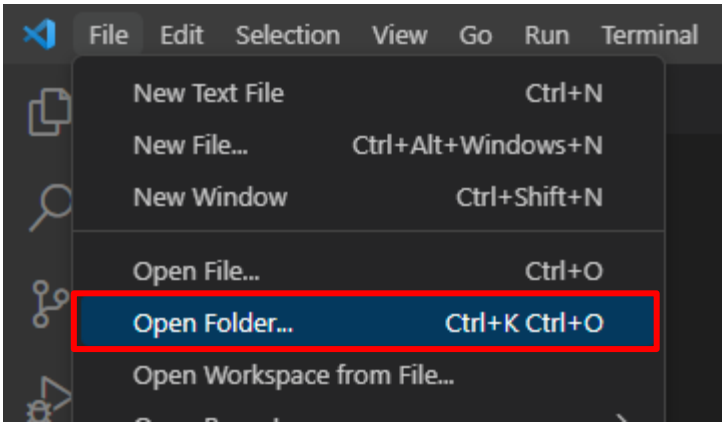
3. A list of additional useful extensions is included in **Appendix A.**

Create a Project Directory

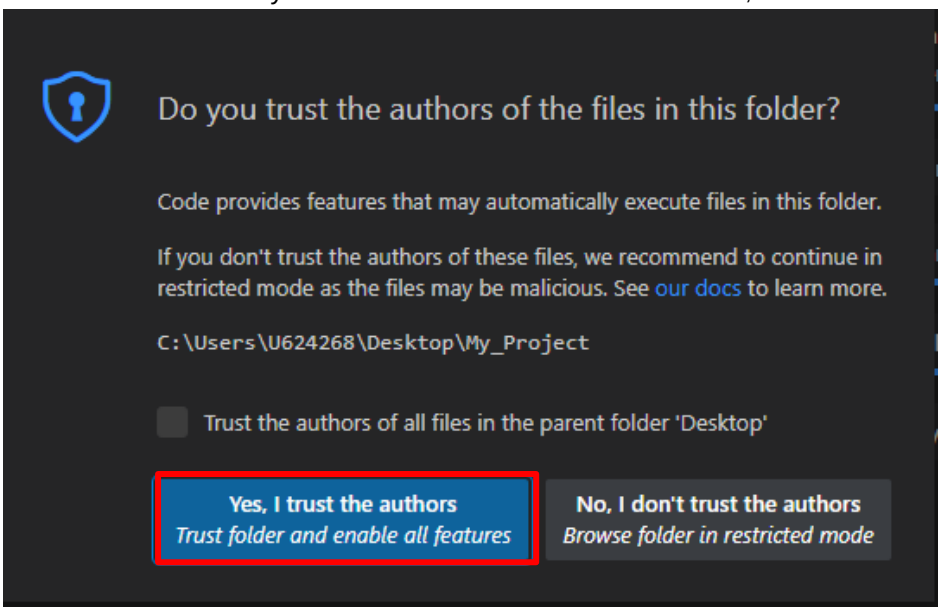
1. In Windows, create a directory/folder in the location where you want to store your project.

2. Start Visual Studio Code from the Windows Start Button (or from the icon on your desktop if you selected to create a desktop shortcut during install) if it isn't already started.

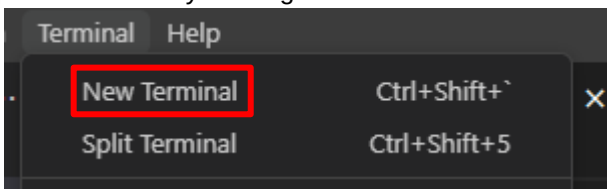
3. In Visual Studio Code, click **File** → **Open Folder**. Navigate to the project folder you created and click **Select Folder**.



4. If asked whether you trust the authors of files in the folder, click **Yes**.



5. Open a terminal by clicking **Terminal** → **New Terminal.**



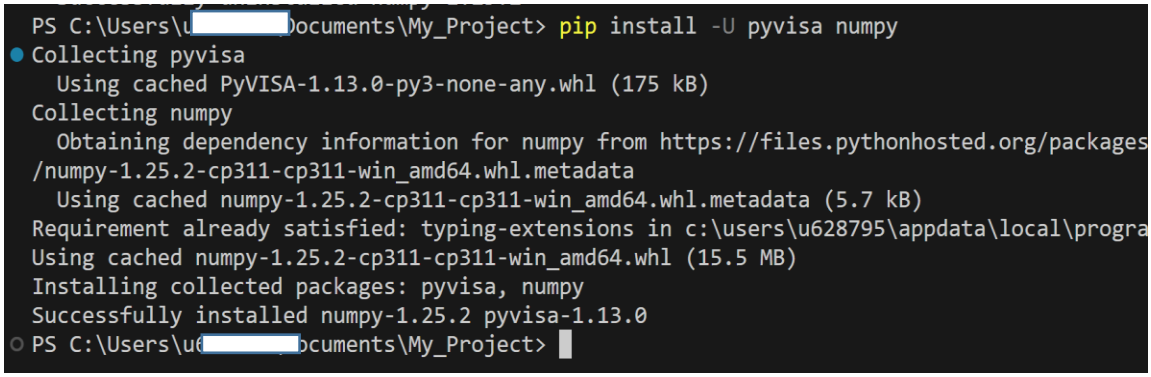
Adding Packages to Python with Pip

1. Pip is the package manager default to Python. We're going to use it to install two packages: PyVISA and NumPy.

   To install the packages, type `pip install -U pyvisa numpy` into the terminal and hit enter. The `-U` flag upgrades packages if they are already installed.

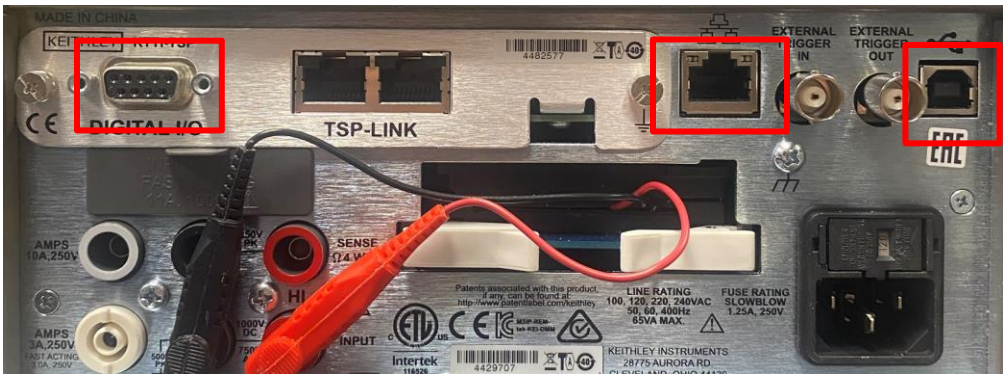
   NOTE: *If your terminal defaults to Windows PowerShell and you receive an error stating that running scripts is disabled, you can ask your system administrator to enable PowerShell scripts or you can switch your default terminal to the Command Prompt in Visual Studio Code by pressing Ctrl+Shift+P, searching for and selecting "Terminal: Select Default Profile", and selecting "Command Prompt."*

2. You should see something like this when it's done:



(RECOMMENDED) Refer to Appendices A-C for more information on configuring a virtual environment and installing other useful Python formatters and packages. These steps are optional but strongly recommended for anyone developing Python code.

Connection Guide

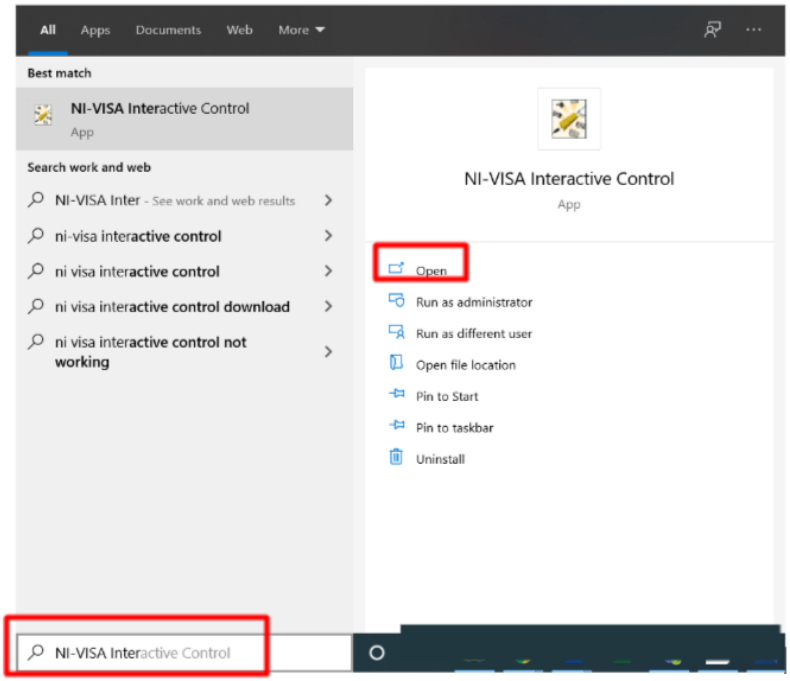


Example connection ports found on the rear panel of the Keithley DMM6500

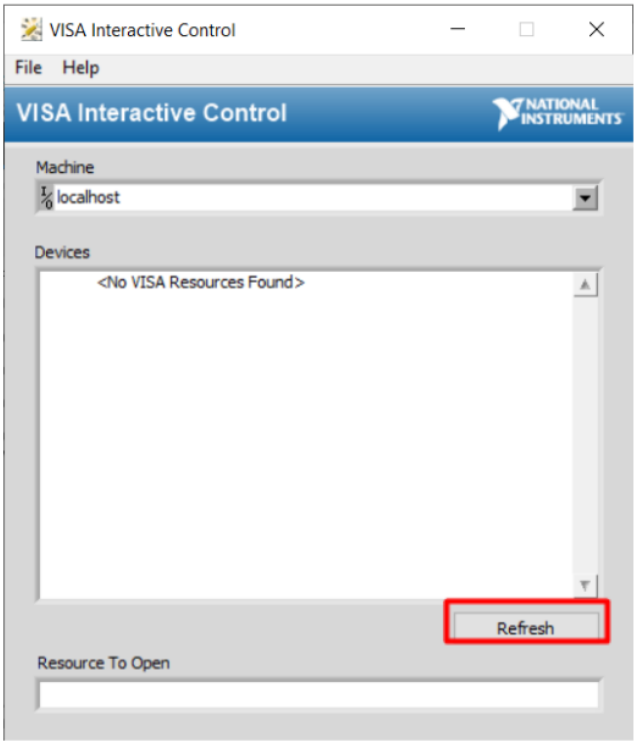
Start by connecting your instrument to an internet-connected computer using an ethernet cable, GPIB to USB cable, or a USB-B to A cable. After connecting your instrument, use the following steps to locate your device's instrument address. The instrument address tells your computer where to send the code and commands that you write.

Obtaining an Instrument Resource String via the VISA Interactive Control Utility

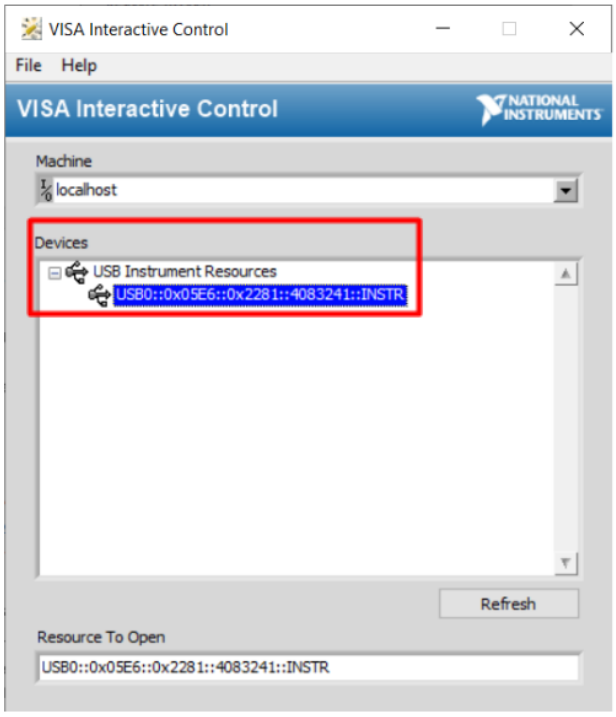
1. Connect the instrument to your PC via one of the interfaces on the rear panel of the instrument (e.g., USB, LAN, GPIB, or RS-232). Make sure the instrument is turned on.
2. Search for **NI-VISA Interactive Control** in the Windows search box and open the utility.



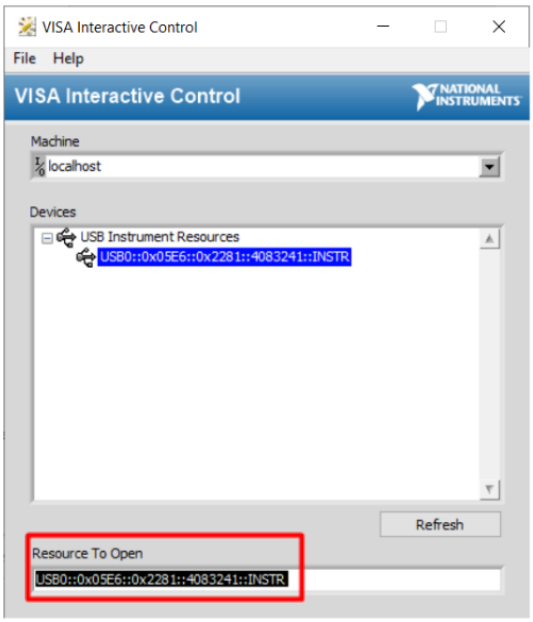
3. If the program does not automatically search for devices, click **Refresh** in the bottom right.



4. Once your connected instrument appears, select it from the “Devices” menu.



5. Copy the instrument string from the “Resource to Open” box and paste it wherever necessary.

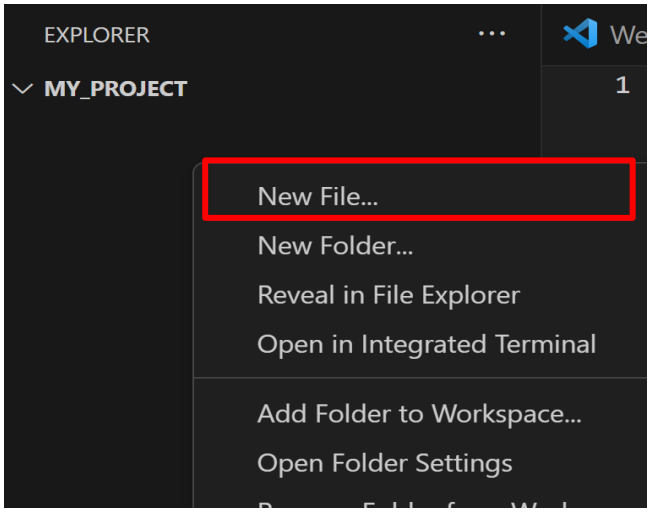


Writing a Program

NOTE: *More information on using Python and VISA together is available at https://pyvisa.readthedocs.io/en/latest/index.html. Here you can find the API Documentation (functions, constants, parameters and return values) for all the library functions available to you. We only cover the basics in this guide.*

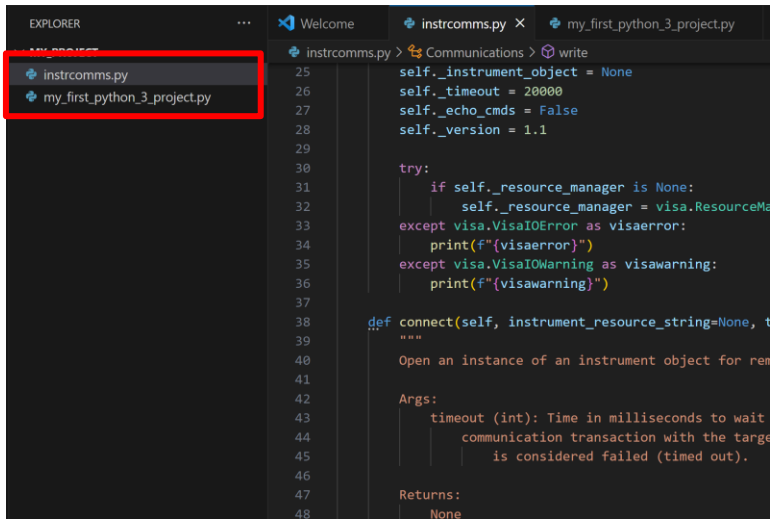
The complete example discussed in this section is posted on the Tektronix/Keithley Github here: https://github.com/tektronix/keithley/tree/main/Instrument_Examples/General/Instructabes/Get_Started_with_Instr_Control_Python

First, create a new file to write your program in. This can be done within Visual Studio Code by right clicking in the Explorer pane and selecting new file. Call the file “my_first_python_3_project.py” or something similar.



The new file should open, or if it does not then it can be opened by clicking on it in the explorer.

We then begin the new program by importing the libraries needed. In this example we need the Communications class from the instrcomms library, which is a file written by Keithley that provides wrapper functions for the pyvisa library and ensures ease of use with Keithley instruments. The instrcomms.py file can be downloaded from the Instrument Communication Resources folder on the Tektronix/Keithley GitHub (Link). The file should be placed in your project directory alongside your main file.



We also need the time library so we can measure time in our program. This is a standard library and should be importable without any additional files or package installs. Import both libraries using the `import` keyword at the top of your file.

```
import time
from instrcomms import Communications
```

Next, let's define a global constant that will be the resource string for our instrument. This string is the identification string of our instrument that VISA uses to connect to it. It defines the communication

protocol, the instrument address and the type of instrument being controlled. The instrument string can be found by launching the VISA Interactive Control Utility or by using the `list_resources()` method to return available resources directly in Visual Studio Code.

Start by taking an initial time reading. We will keep track of the elapsed time by capturing the beginning and end times and taking the difference. To capture the time, use the `time()` function from the time library.

```
start_time = time.time()                    # Record start time
```

Next, create an instance of the `Communications` class we imported from instrcomms. We instantiate the object passing in our instrument's resource string, then we connect to the instrument by calling the class's `connect` function.

```
my_instr = Communications(INST_RESOURCE_STR)
my_instr.connect()
```

Now reset the instrument settings to their default values and clear the reading buffers by using the `write()` function to write the **`*RST`** command to the device. This puts the instrument into a known state and makes it easy to configure the instrument in a repeatable way. If we had further settings we wanted to change, we would do so after resetting the instrument.

```
my_instr.write("*RST")
```

Next, we use a for loop to loop over the following three commands 10 times. In the loop, we again use instrument `write()`, this time to query the instrument's identification string with the **`*IDN`** command. The **`*IDN`** command tells the instrument to return its identification string, which we then read with the `read()` command. We also use the instrument `query()` function, which does the same thing as the previous two statements; it essentially calls `write()` to write the command we pass it as an argument, then follows it with `read()`.

```
for _ in range(10):
    my_instr.write("*IDN?")
    print(my_instr.read())
    print(my_instr.query("*IDN?"))  # query is the same as write + read
```

Finally, we call `disconnect()` to disconnect the instrument. We collect the time again with `time.time()`, and print a notification that the program has completed. We also print the difference between the stop and start times so the user knows how long the program was running.

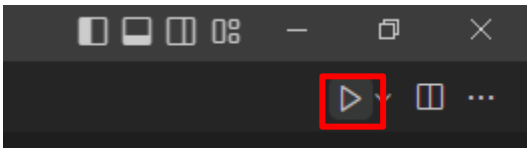
```
my_instr.disconnect()

stop_time = time.time()  # Record stop time

# Notify the user of completion and the data streaming rate achieved.
print("done")
print(f"Elapsed Time: {(stop_time-start_time):0.3f}s")
```

That marks the end of the example program! The full program follows:

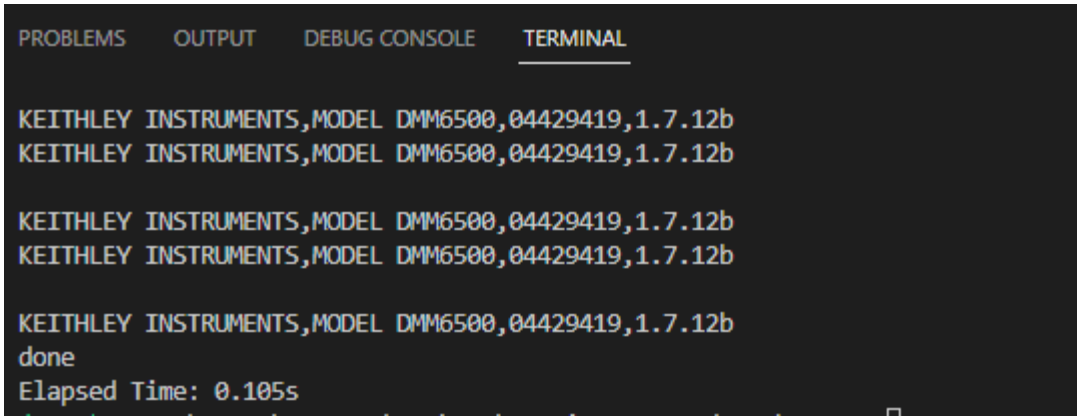
```python
import time
from instrcomms import Communications

INST_RESOURCE_STR = (
    "USB0::0x05E6::0x6500::04429707::INSTR"  # Get from VISA Interactive Control
)
start_time = time.time()  # Record start time

my_instr = Communications(INST_RESOURCE_STR)
my_instr.connect()

my_instr.write("*RST")
for _ in range(10):
    my_instr.write("*IDN?")
    print(my_instr.read())
    print(my_instr.query("*IDN?"))  # query is the same as write + read

my_instr.disconnect()
stop_time = time.time()  # Record stop time
# Notify the user of completion and the data streaming rate achieved.
print("done")
print(f"Elapsed Time: {(stop_time-start_time):0.3f}s")
```

Running the Program

There are multiple ways to run the program, including pressing Ctrl+F5, selecting **Run Without Debugging** from the Run tab at the top, or clicking the run button at the top right of the window.



When the program runs, the output is displayed in the terminal at the bottom of the screen.



Conclusion

Congrats, you controlled an external instrument using your PC! As noted earlier, there are many more features of the VISA library. See the API Documentation at https://pyvisa.readthedocs.io/en/latest/index.html for a complete list of all available methods. Visual Studio Code also has many debugging features, such as the ability to step through your programs line-by-line and view the state of program variables. More information on the debugging tools can be found here: https://code.visualstudio.com/docs/python/debugging. For further instrument control, your instrument's reference manual will provide a complete index of all commands along with their parameters and return values. We strongly encourage you to explore VISA, the Python language and Visual Studio Code's features to advance your automation skills. The complete program code discussed today (along with numerous other examples) can be found at https://github.com/tektronix/keithley/tree/main/Instrument_Examples/General/Instructables/Get_Started_with_Instr_Control_Python

.

Appendix A – Useful Visual Studio Code Extensions

- **Python:** Adds rich support for actively supported Python versions. https://marketplace.visualstudio.com/items?itemName=ms-python.python
- **isort**: Automatically sorts Python import statements https://marketplace.visualstudio.com/items?itemName=ms-python.isort
- **Black Formatter:** Adds black formatter natively to Visual Studio Code and used when black is not installed in the active environment. Planned to eventually replace black formatting functionality in the Python Extension. https://marketplace.visualstudio.com/items?itemName=ms-python.black-formatter

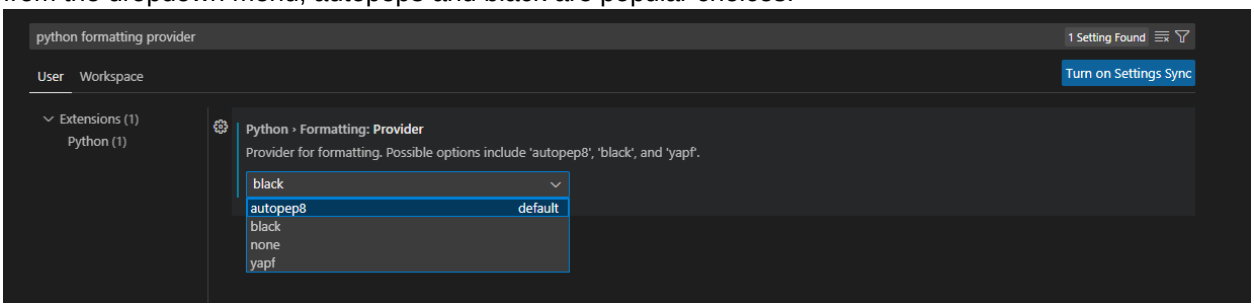
Appendix B – Installing a Formatter and Linter

Using Python Code Formatters and Linters in Visual Studio Code

In order to improve readability, maintain consistency, and reduce the chance for programmatic errors, it is often desirable to enforce some form of style checking upon code you write. One way to accomplish this is with a code formatter, linter, or both. Code formatters automatically adjust the style of your code as you work, while code linters will identify stylistically poor or ambiguous code. Visual Studio Code's Python extension includes a variety of code formatters and linters for Python.

Choose and Install a Formatter

To choose a code formatter, go to Visual Studio Code's setting by clicking **File** → **Preferences** → **Settings**. In the settings menu, search for "python formatting provider" and choose your desired formatter from the dropdown menu; autopep8 and black are popular choices.

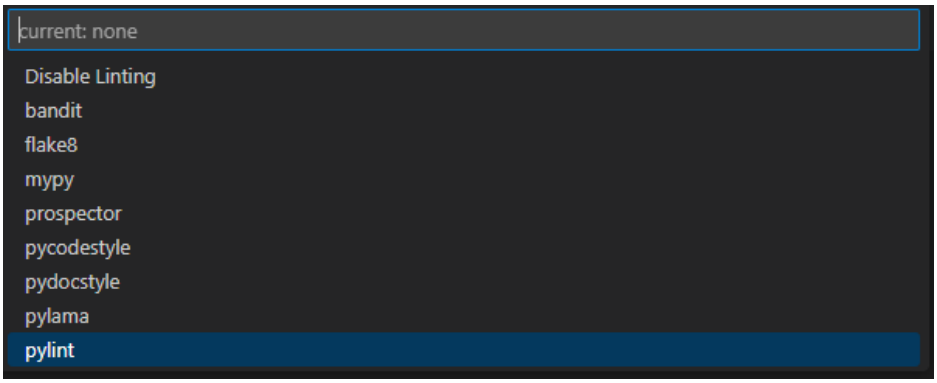


Next, search for "format on save" in the settings menu and enable the **Editor: Format on Save** option to automatically apply formatting when you save your file. If desired, you can also search for and enable **Editor: Format on Paste** and/or **Editor: Format on Type**.

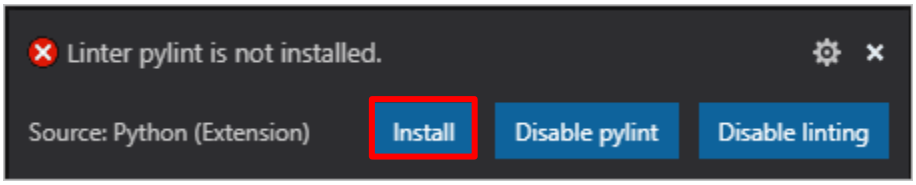
Visual Studio Code will prompt you to install the formatter you choose in your environment when it first tries to format your code. However, you can also install it with pip. For example, you would install black (the formatter used in this tutorial) by running the command `pip install black` in a terminal with your virtual environment activated.

Choose a Linter

To enable linters, open the command palette with the keyboard shortcut Ctrl+Shift+P. Search for and select **Python: Select Linter.** Then, select the linter you would like to use; pylint was used in this tutorial.



You will be prompted to install your linter. Click **Install**.



If you weren't prompted to install the linter automatically then you can install it with pip. For example, you can install pylint with `pip install pylint`. Linting is automatically run when your files are saved.

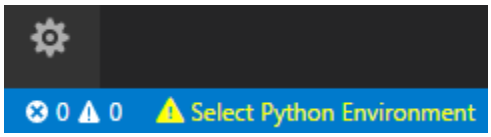
Appendix C – Using Virtual Environments in Visual Studio Code

In the following steps, we will create a virtual environment to isolate packages used by your project from your main install. **This is optional but recommended as general practice when managing Python projects**. This should be done each time a new python project is created, in order to separate its packages and environment from those of other projects.

Python versions 3.3+ come with venv, a simple tool for creating virtual environments (https://docs.python.org/3/library/venv.html). If you are using multiple versions of Python on the same machine and want to use a specific version for your projects, then invoking venv with that version of the Python interpreter will create a virtual environment running the same version. For older versions of Python, *virtualenv* is also a good tool: https://virtualenv.pypa.io/en/latest/.

1. In the terminal, type `python -m venv venv` and press enter. The first `venv` in the statement refers to the venv python module, while the second is the path to the new virtual environment. In this case, a directory called "venv" will be created in your project directory.

2. Visual Studio Code may notice the new environment and ask if you want to use it in the workspace; click **Yes**. If not, select the button on the bottom status bar labeled **Select Python Environment**. Then select the python interpreter you wish to use (in this case, the virtual environment we just created). There may already be one selected, in which case it will be shown on the bottom status bar instead. You can click this to select a different interpreter.



3. New terminals created in this workspace should now automatically activate the virtual environment. To activate the environment in the already open terminal, type `.\venv\Scripts\activate` in the terminal and press enter. Commands in the terminal will now use the virtual environment's interpreter, and pip commands executed in this terminal will install packages to the virtual environment instead of globally. The terminal should indicate that a virtual environment is in use with the name of the virtual environment in parentheses prior to the working directory:

```
(venv) PS C:\Users\Username\...\My_Project>
```

Revisions

Revision	Date	Authored by	Notes
1.0	2019-08-23	Josh Brown and Elizabeth Makley	
2.0	2021-08-18	Adam Billmaier	• Updated screenshots and steps for new versions of installs • Added sections on adding packages to PyCharm and obtaining an instrument's resource string
3.0	2023-01-13	Lukas Hazen-Bushbaker	• Updated code to be more Pythonic and use modern Python language features • Updated guide to use Visual Studio Code instead of PyCharm • Added steps showing how to use venv to create virtual environments and pip to install packages • Updated screenshots and steps for new versions of installs • Added section on Formatters and Linters
4.0	2023-09-1	Evan Murphy	• Added table of contents • Moved virtual environment/formatting sections to appendix. • Added section about getting instrument string • Added section about connection guide • Updated guiding code and images