

Lab Exercises 1: C/C++ and Python Development

Setup

1. Introduction

This guide introduces how to configure a basic Python development environment on your own computer, and also cover advanced topics such as:

- (1) Using virtual environments for projects
- (2) Installing third-party packages
- (3) Managing multiple Python versions.

2. Objectives

- (1) Install Python
- (2) Install an editor
- (3) Create and use virtual environments
- (4) Manage Python Packages
- (5) Use Vscode

3. Installation

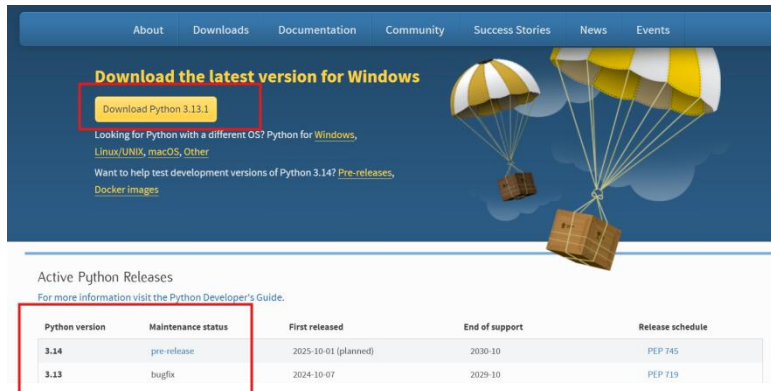
The following is a basic installation tutorial for Python. On macOS and Windows systems, the operations are similar.

NOTE: You can use any of the following configurations for Python development:

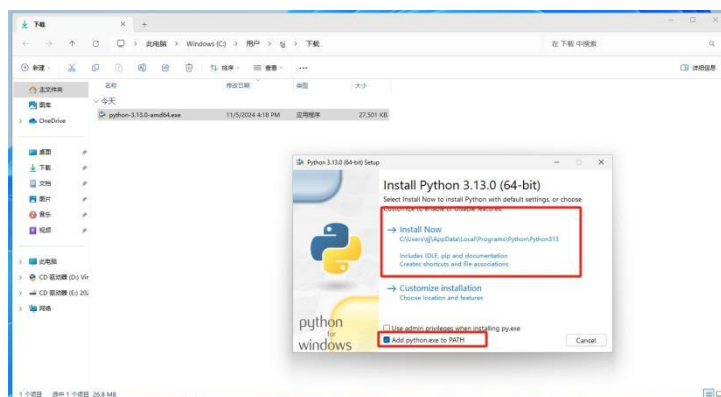
- (1) Official Python
- (2) Vscode + Anaconda (recommended)

3.1 Install Official Python

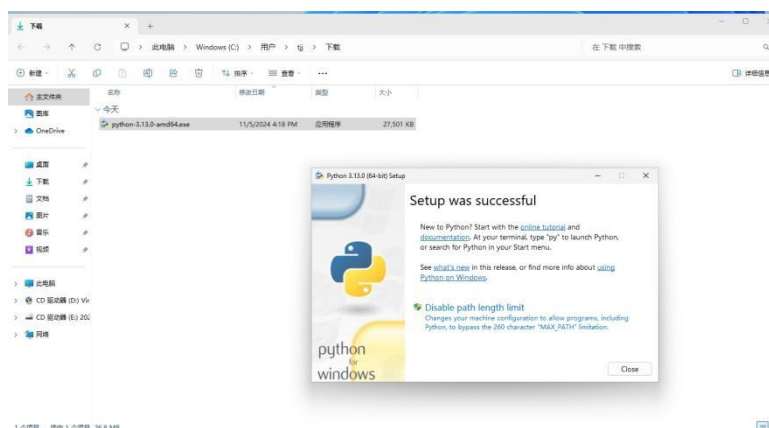
1. Visit the official website (<https://www.python.org/downloads/>) to download the latest stable release(e.g., Python3.13 (<https://www.python.org/ftp/python/3.13.0/python-3.13.0-amd64.exe>)).



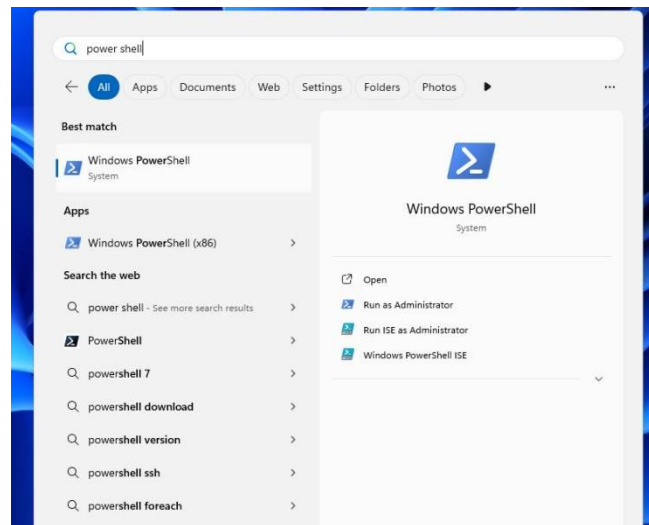
2. Double-click the installer to run it. On the first screen, Check the box labeled **Add python.exe to PATH** at the bottom. Click on **Customize installation** if you want to choose specific features or installation locations. Otherwise, click **Install Now** for default settings.



3. Wait for the installation process to complete. Once done, click **Close** to exit the installer.



4. Click on the **Start** button or press the **Windows** key on your keyboard. Type **PowerShell** in the search bar. Click on **Windows PowerShell** from the search results to open it.



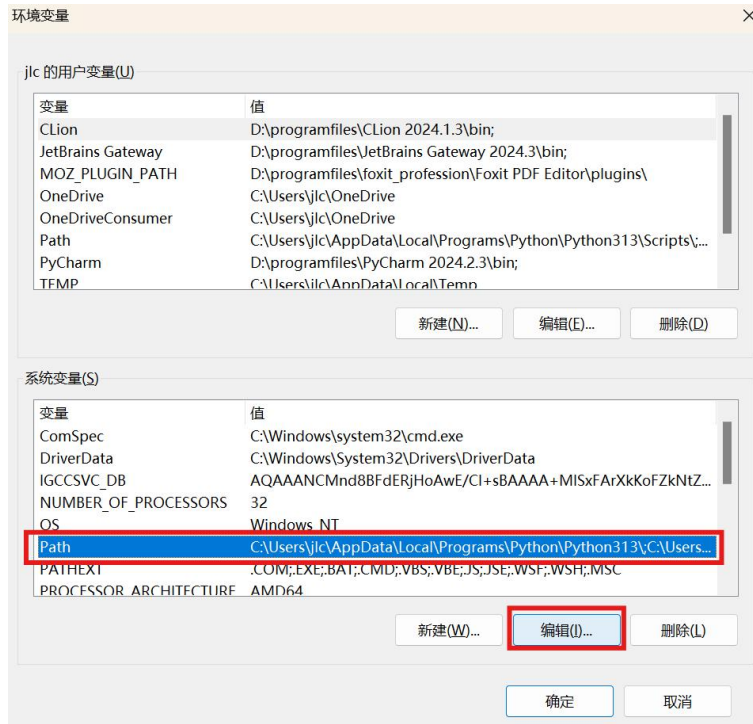
5. Type **python --version** and press Enter. You should see the installed Python version number.

```
PS C:\Users\tjj> python --version
Python 3.13.0
```

If it fails, you need to manually check your environment variable settings and add the following 2 lines into System variables.

```
C:\YourPath\Python\Python313\
C:\YourPath\Python\Python313\Scripts\
```





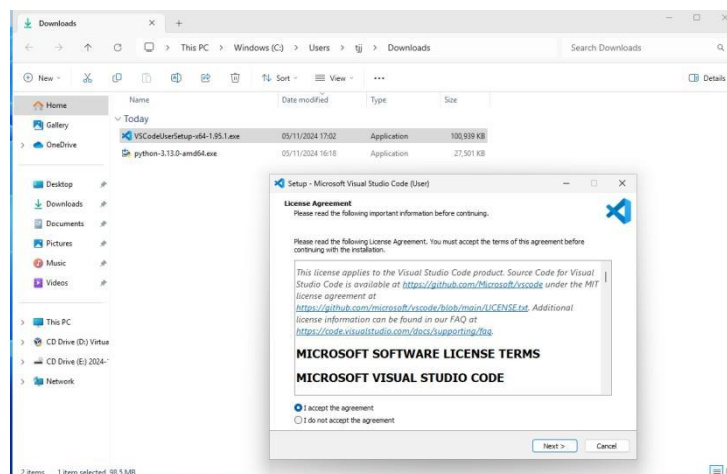
3.2 Install Vscode and Anaconda

3.2.1 Visual Studio Code (recommended)

Visual Studio Code (VS Code) is a free, open-source code editor developed by Microsoft. It is widely used by developers for its versatility and extensive customization options.

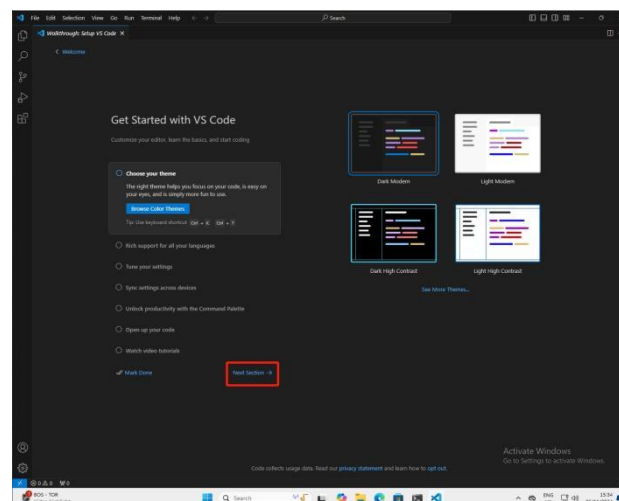
1. Install VS Code

Go to the VS Code website (<https://code.visualstudio.com/Download>) and click [Download](#) for your operating system (Windows, macOS, or Linux). Once downloaded, open the installer and follow the on-screen instructions.



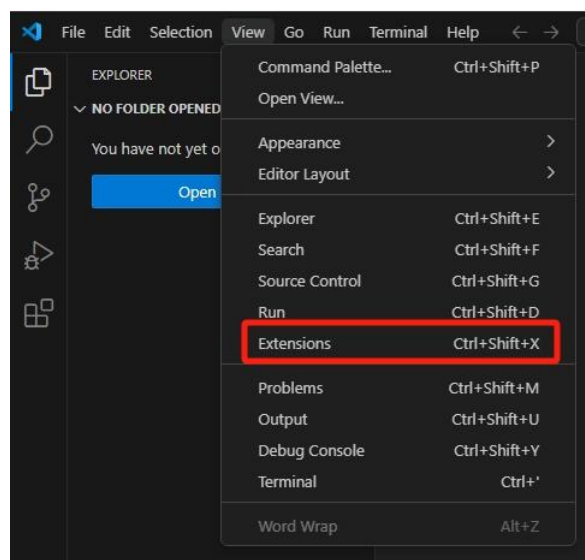


After installation, open VS Code.



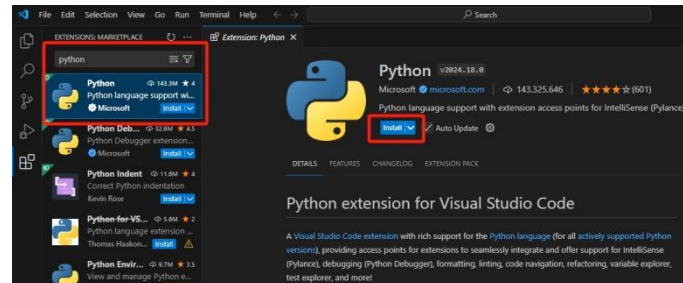
2. Install Python Extension

(1) In VS Code, select **View > Extensions** to open the Extensions view.



(2) Filter the list of available extensions by entering **python** in the search box at the top of the Extensions view.

(3) Select the Python extension published by Microsoft. The details about that extension appear in a tabbed panel on the right. In either the Extensions panel, or in the main panel, select **Install**.



When the installation is complete, the **Install** button changes to a **Settings** icon in the Extensions view or two buttons, **Disable** and **Uninstall** in the main panel. This message lets you know that you've successfully installed the Python extension for Windows.



3.2.2 Pycharm (optional)

PyCharm (<https://www.jetbrains.com/pycharm/download/>) is an integrated development environment (IDE) specifically designed for Python programming. Developed by JetBrains, PyCharm offers a wide range of features that streamline the coding process, making it a popular choice among Python developers.

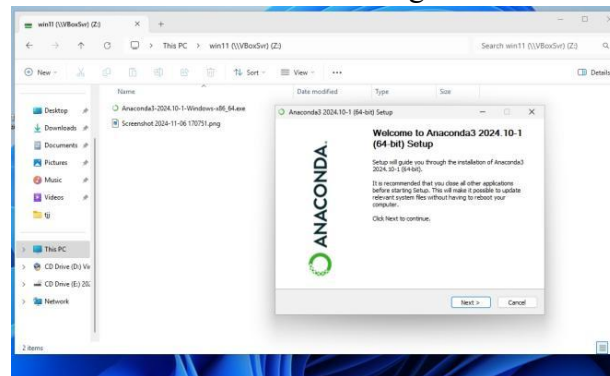
(1) PyCharm Professional Edition requires a subscription or a one-time purchase. It includes advanced features that are particularly useful for professional developers. JetBrains offers a special program (<https://www.jetbrains.com/student/>) for students to access their Professional Edition tools, including PyCharm, for free.

(2) PyCharm Community Edition is open-source and available for free to all users. It is suitable for students, hobbyists, and developers who need basic Python development capabilities.

3.2.3 Anaconda

If you're in the data science community, you might already be using Anaconda (or Miniconda). Anaconda is a sort of one-stop shop for data science software that supports more than just Python.

You can download Anaconda from the official website (<https://www.anaconda.com/download/success>) and install it with default settings.



4. Create and use virtual environments

4.1 Conda Source

We recommend using Tsinghua Tuna Mirror (<https://mirrors.tuna.tsinghua.edu.cn/help/anaconda/>) to download packages of conda.

First, you can find `.condarc` file:

- (1) macOS: `${HOME}/.condarc`
- (2) Windows: `C:\Users\<YourUserName>\.condarc`

For windows users, you can execute `conda config --set show_channel_urls yes` in your terminal if you can not find `.condarc` file.

For macOS users, you can enter `HOME` directory and execute `open -e .condarc` to open the file.

After that, you can modify the contents of `.condarc` to use tuna mirror (copy the following content and paste it):

```
channels:
  - defaults
show_channel_urls: true
default_channels:
  - https://mirrors.tuna.tsinghua.edu.cn/anaconda/pkgs/main
  - https://mirrors.tuna.tsinghua.edu.cn/anaconda/pkgs/r
  - https://mirrors.tuna.tsinghua.edu.cn/anaconda/pkgs/msys2
custom_channels:
  conda-forge: https://mirrors.tuna.tsinghua.edu.cn/anaconda/cloud
  pytorch: https://mirrors.tuna.tsinghua.edu.cn/anaconda/cloud
```

4.2 Common conda commands

Managing Conda Itself

1. Check Conda Version: `conda --version`
2. Update Conda: `conda update conda`

Managing Environments

1. Create a New Environment: `conda create -n your_env_name python=3.xx`
2. Activate an Environment: `conda activate your_env_name`
3. Deactivate an Environment: `conda deactivate`
4. List All Environments: `conda env list`
5. Remove an Environment: `conda remove -n your_env_name --all`
6. Export an Environment: `conda env export --name env_name > env_name.yml`
7. Create an Environment from a File: `conda env create -f env_name.yml`

Managing Packages

1. List Installed Packages: `conda list`
2. Install a Package: `conda install package_name`
3. Install a Specific Version of a Package: `conda install package_name=version`
4. Update a Package: `conda update package_name`
5. Remove a Package: `conda uninstall package_name`

DO NOT MODIFY YOUR BASE ENVIRONMENT.

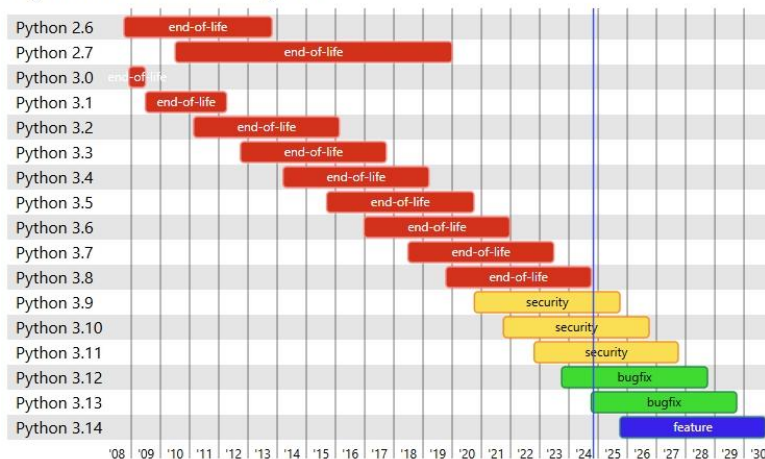
4.3 Create a virtual environment with conda

A Python virtual environment is an isolated environment that allows you to manage dependencies for a specific Python project without affecting other projects. It essentially creates a self-contained directory that contains a Python interpreter and copies of any necessary libraries.

0. Python version Management

Python has multiple versions, with the most significant division being between Python 2 and Python 3, which are not compatible with each other. Certain projects may require specific Python versions, so managing different versions is essential. Here, we introduce the conda tool, which helps in installing, switching, and using different Python versions.

Python release cycle



1. Open Terminal or Command Prompt

Start by opening your terminal or command prompt where you have Conda installed. To create a new virtual environment, use the following command:

```
conda create --name your_env_name python=3.12
```

Replace `your_env_name` with the desired name for your new virtual environment.

2. Activate the Virtual Environment

Once the environment is created, activate it using the command:

```
conda activate your_env_name
```

This command switches your current Python environment to the newly created one.

3. Install Packages

You can now install packages into your virtual environment using Conda. For example, to install numpy, use:

```
conda install numpy
```

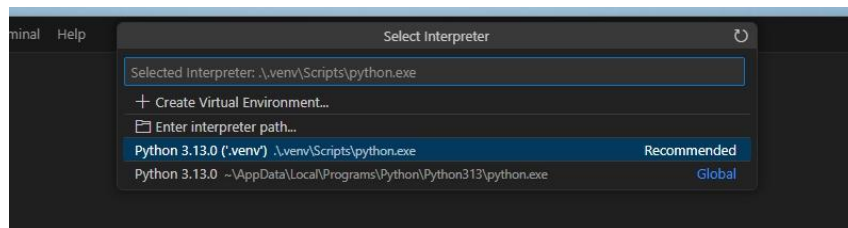
Wait a moment the new package will be installed.

5. Usage of Vscode

5.1 Select a virtual environment

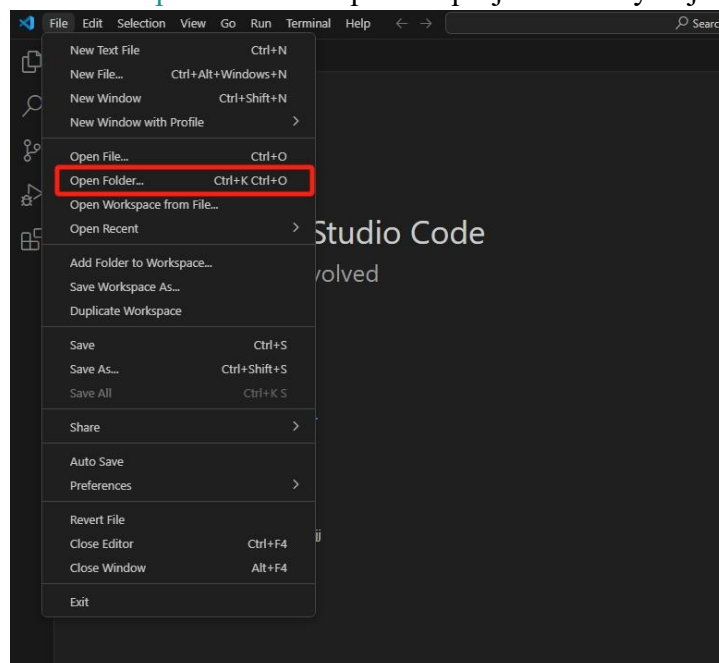
In VS Code, press **Ctrl+Shift+P** to open the Command Palette.

Type and select **Python: Select Interpreter**. Ensure your new environment is selected. It should look something like `\\.venv\\Scripts\\python.exe`. After this, you should see the selected interpreter in the bottom right corner.

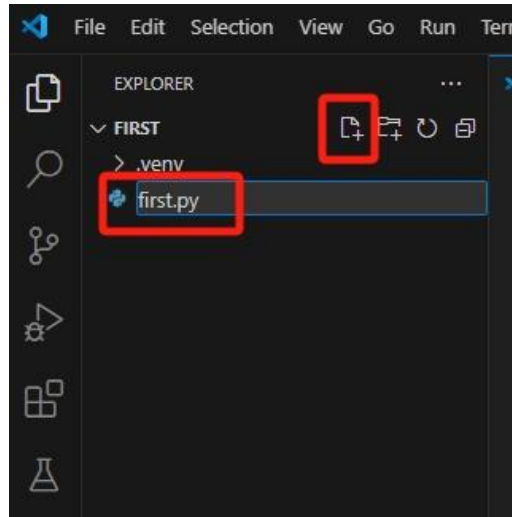


5.2 Create Python scripts

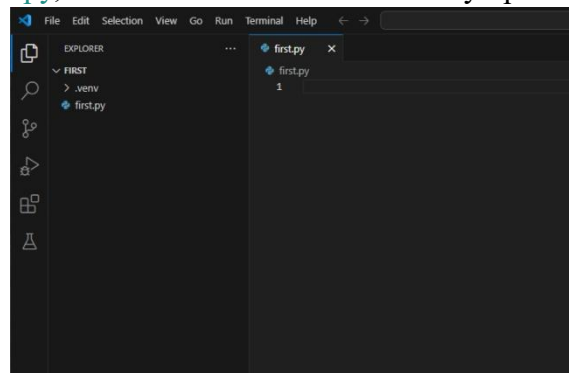
1. Create a new project folder in the File Explorer.
2. Use VS Code's **File > Open Folder** to open the project folder you just created.



4. From the File Explorer toolbar, select the New File button to create your `.py` file:



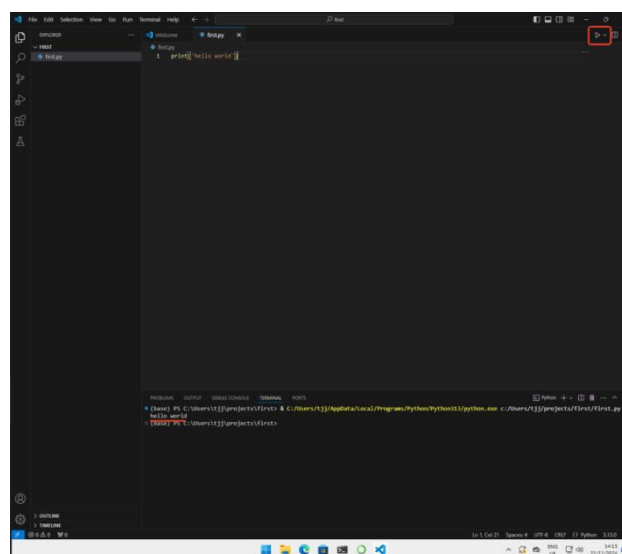
5. Name the file `first.py`, and VS Code will automatically open it in the editor:



6. Write Python code

```
print('hello world!')
```

7. Click the Run Python File play button in the top-right side of the editor. The button opens a terminal panel in which your Python interpreter is automatically activated, then runs `python3 first.py` (macOS/Linux) or `python first.py` (Windows):



8. (Optional) Press **Ctrl+`** to open the Terminal. Find your current working path and press **cd your_path** to enter the working directory. In your terminal, press **python first.py** to execute your python script.

Congrats, you just ran your first Python code in Visual Studio Code!

5.3 Create a Jupyter Notebook

1. Install Required Extensions.

Open your terminal, activate your own virtual environment, and install a package:

```
pip install ipykernel
```

or you can use

```
conda install ipykernel
```

```
(lwc_py312) jaden@JadenMac lab1 % conda install ipykernel
Channels:
- defaults
- conda-forge
- pytorch
Platform: osx-arm64
Collecting package metadata (repodata.json): done
Solving environment: done

# All requested packages already installed.

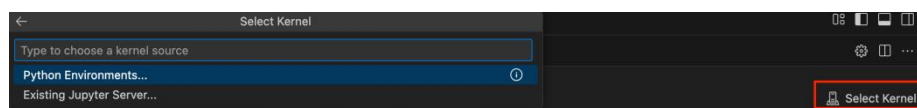
(lwc_py312) jaden@JadenMac lab1 %
```

2. Create a New Jupyter Notebook

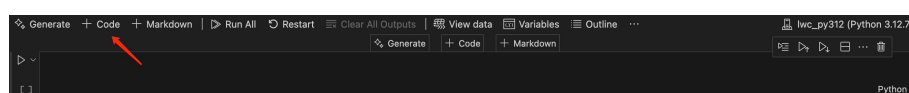
Click on the "File" menu and select **New File**. Save the new file with a **.ipynb** extension, for example, **my_notebook.ipynb**. You can now start writing your Jupyter notebook code in this file.

3. Write Code in Cells

(1) Open **my_notebook.ipynb**, click **Select Kernel** and select **Python Environment**. Chosse your own virtual environment.

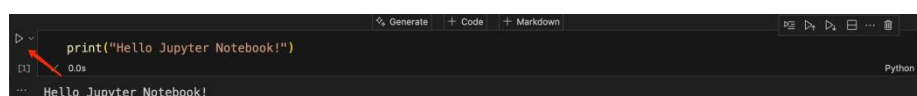


(2) To add a new code cell, click on the **+code** button in the toolbar or press **Alt + Enter**.



(3) Write your Python code in the cell.

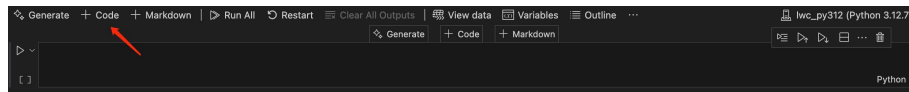
(4) To execute the code in a cell, click on the **Run** button or press **Shift + Enter**



(5) To add an extra cell, press **Shift+Enter** or Click **right button -> Insert Cell -> Insert Code Cell Below** (or Above).

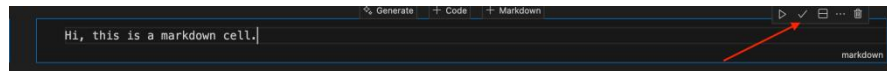
4. Write Markdown in Cells

(1) To add a new markdown cell, click on the **+Markdown** button in the toolbar.



(2) Write your markdown code in the cell.

To execute the code in a cell, click on the **Run** button.



5. Save and Export

Save your Jupyter notebook by clicking on the save icon or using **Ctrl + S**.

You can export your notebook to various formats using the "File" menu.

Exercises

1. Create your own environment

Create a virtual conda environment named `<yourname>_2043`, i.e. `jaden_2043`.

Install the following packages:

```
numpy
matplotlib
scikit-learn
seaborn
pandas
scipy
```

Uninstall the following packages:

```
pandas
scipy
```

2. Two sum

Write a Python program, define two integer variables, and output the sum of the two numbers.

baseline:

```
x = 20
y = 43
z = x + y
print("x + y=", z)
```

2. Generating random numbers

Write a Python program using the NumPy library to generate an array containing 10 random integers.

baseline:

```
import numpy as np
result = np.random.randint(low=0, high=2043, size=10)
print(result)
```

3. Plotting a figure

Write a Python program using the Matplotlib library to plot a simple line graph showing the temperature variation for each day of a week.

baseline:

```
import matplotlib.pyplot as plot
temperature_list = [13, 15, 19, 16, 20, 15, 14]
plt.plot(temperature_list)
plt.show()
```

4. Using Notebook

Create two Jupyter Notebooks to implement Exercise 2 and Exercise 3 respectively.

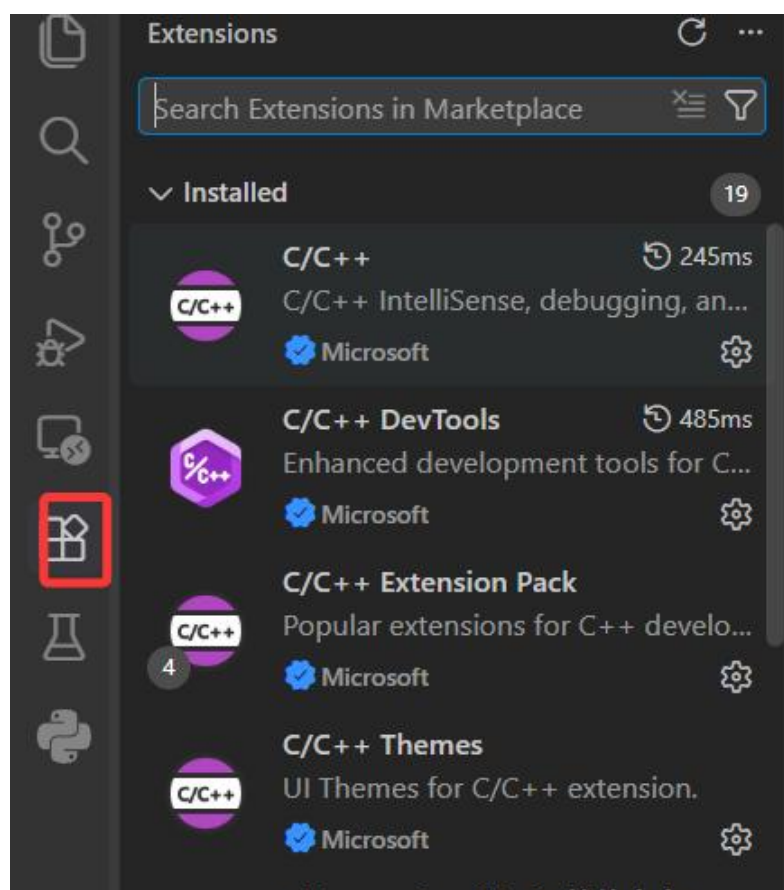
6. C/C++ Development Setup in VS Code (Windows)

This section configures a native 64-bit C/C++ development environment for VS Code on Windows. It uses the Microsoft C/C++ extension and w64devkit, a portable MinGW-w64 toolchain that includes GCC, G++, GDB, Make, and related command-line tools.

NOTE: The screenshots use `C:\Users\bamboosir920\Downloads\w64devkit\bin`. On another computer, replace the user name and extraction folder with the actual path. Choose a permanent folder because moving w64devkit later requires updating every configured path.

6.1 Install the Microsoft C/C++ Extension

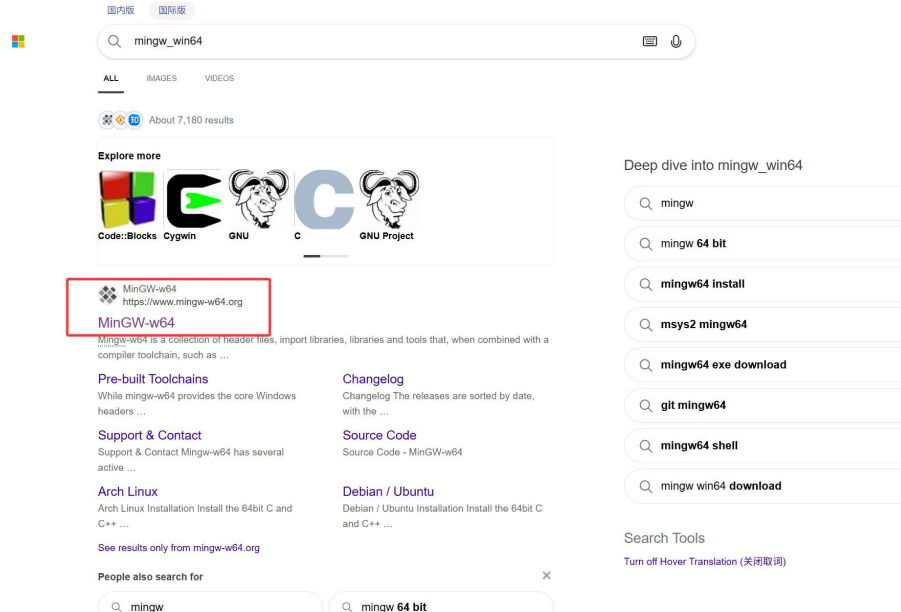
1. Open VS Code. Select **View > Extensions** or press **Ctrl+Shift+X**.
2. Search for **C/C++**. Select the extension published by Microsoft and click **Install**. If it is already installed, VS Code shows a gear icon instead of the Install button.



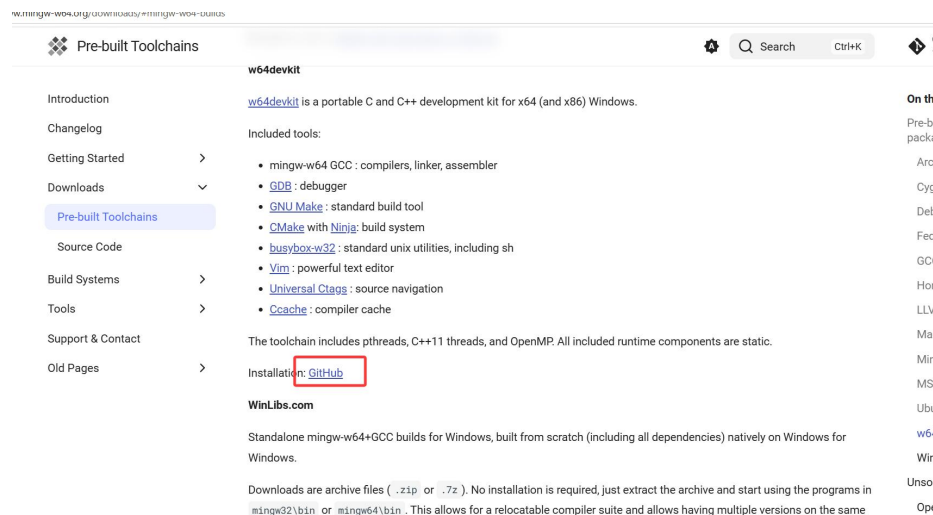
3. Only the Microsoft **C/C++** extension is required for this tutorial. C/C++ DevTools, C/C++ Extension Pack, and C/C++ Themes are optional.

6.2 Download and Extract w64devkit

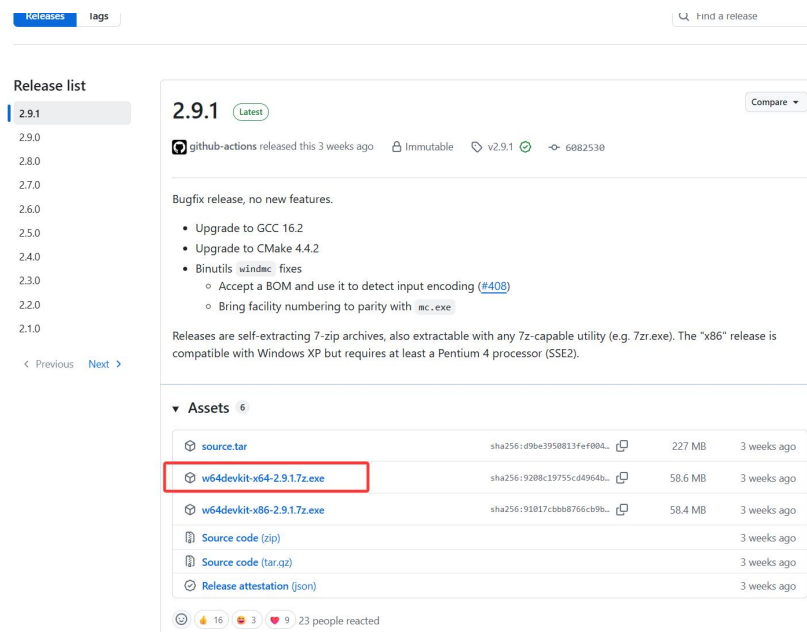
1. Open a browser and go directly to <https://www.mingw-w64.org/>. You can also search for MinGW-w64 and open the official site.



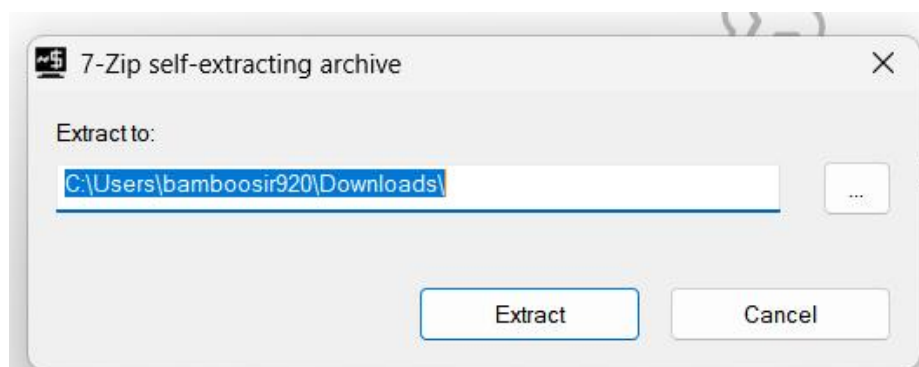
2. On the MinGW-w64 website, open [Downloads > Pre-built Toolchains](#). Scroll to the [w64devkit](#) entry and click its GitHub link.



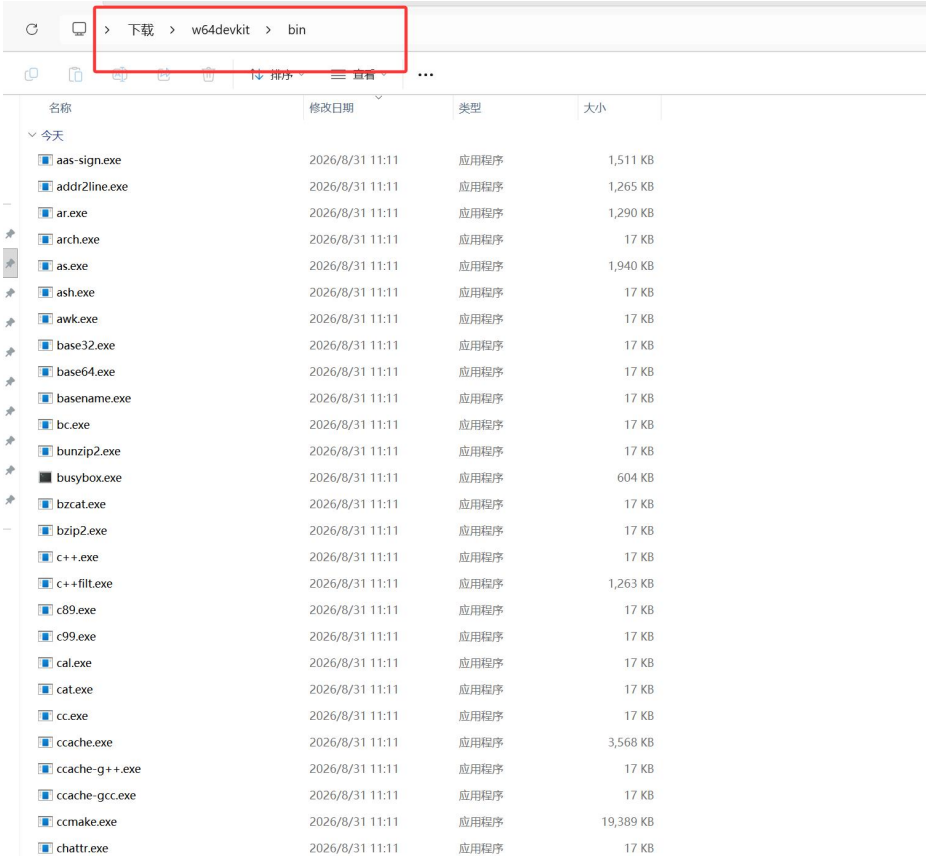
3. On the GitHub Releases page, expand [Assets](#) for the latest release and download [w64devkit-x64-<version>.7z.exe](#) for a 64-bit Windows computer. The version number will change over time; do not select the x86 package unless your computer is 32-bit.



4. Run the downloaded self-extracting archive, and click [Extract](#). This extracts a portable toolchain; it is not a traditional installer.



5. Open the extracted `w64devkit\bin` folder. Press `Ctrl+L` in File Explorer and copy the complete path from the address bar.

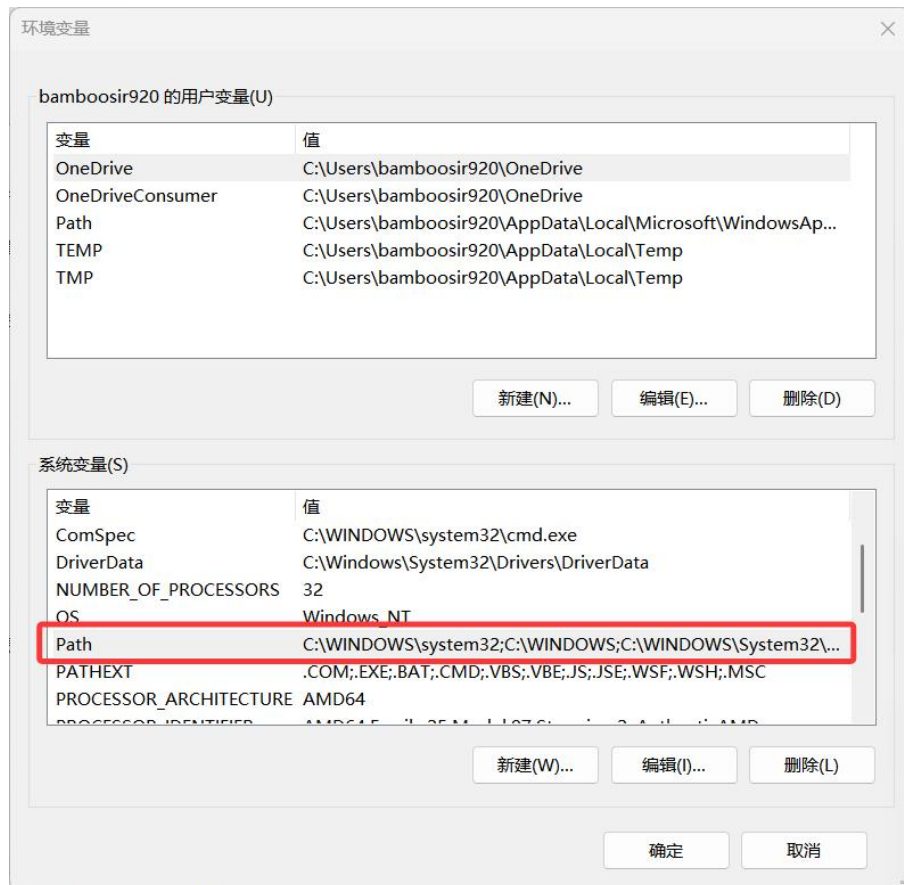


6.3 Add w64devkit to the Path Environment Variable

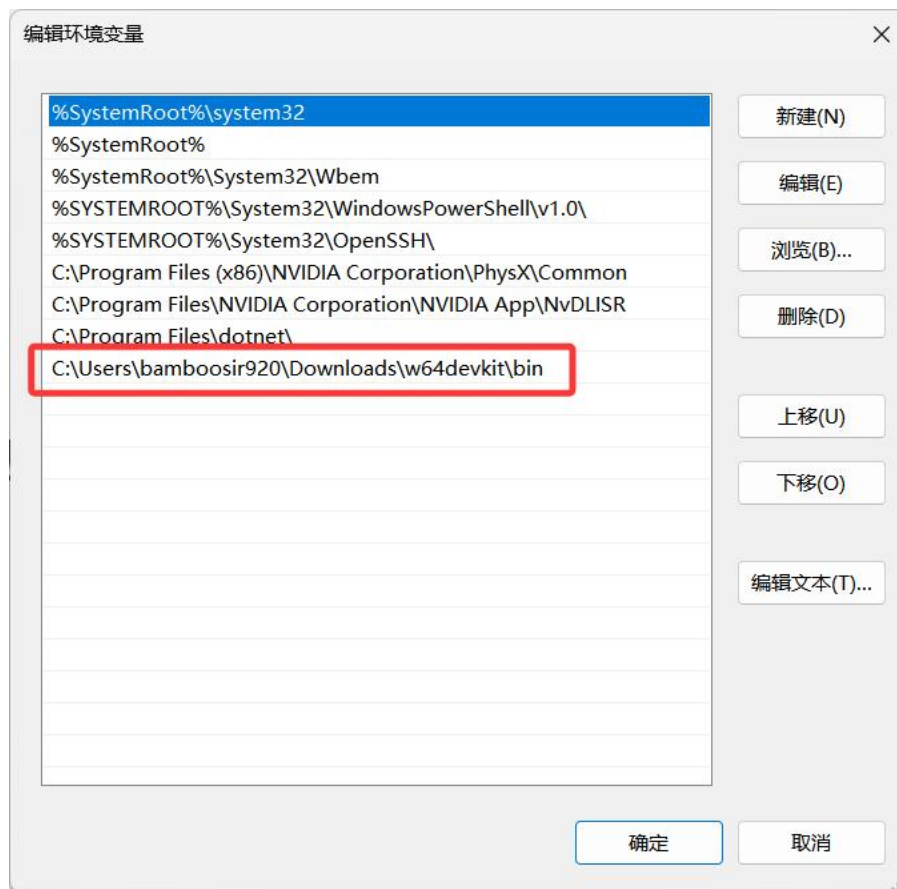
1. Search Windows for [Edit the system environment variables](#). In System Properties, open the [Advanced](#) tab and click [Environment Variables](#).



2. Under **User variables**, select **Path** and click **Edit**. Editing User Path is normally sufficient and does not require administrator permission. The screenshot below highlights System Path; use it only when the toolchain must be available to every user and you have administrator permission.



3. Click [New](#) and paste the copied path, for example [C:\Users\. Add it as a new entry; do not replace or delete the existing Path entries. Click OK in every open dialog.](#)



4. Fully exit all VS Code windows and reopen VS Code. A terminal that was already open may still have the old environment. Open a new PowerShell terminal and run:

```
where.exe gcc
gcc --version
g++ --version
gdb --version
```

The first command should show a path inside w64devkit\bin, and the remaining commands should print version information. If any command is not found, check that the Path entry points to the bin folder rather than the w64devkit root folder or to an individual .exe file.

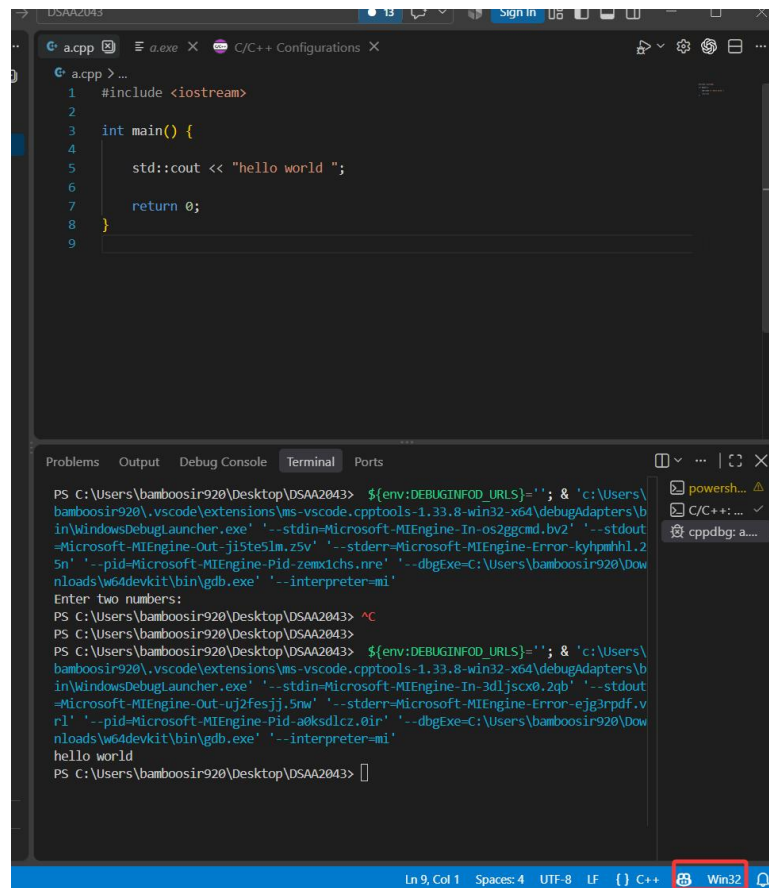
6.4 Configure C++ IntelliSense

1. Open a project folder in VS Code and create a file named `hello.cpp` with the following program:

```
#include <iostream>

int main() {
    std::cout << "Hello, World!\n";
    return 0;
}
```

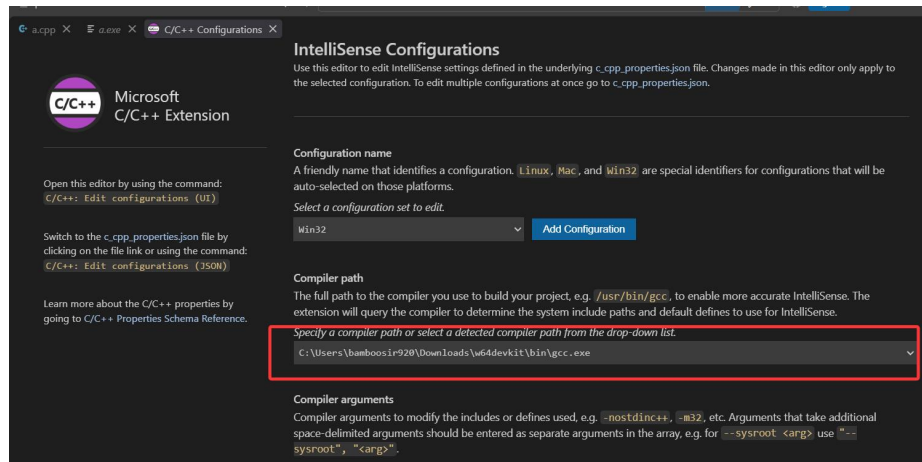
2. Click the C/C++ configuration indicator in the lower-right status bar (it may be named `Win32`). `Win32` is a configuration name; it does not mean that the selected compiler is 32-bit.



3. Select `Edit Configurations (UI)`.



4. In **Compiler path**, select or enter the full path to `w64devkit\bin\g++.exe`. The screenshot shows `gcc.exe`; for a `.cpp` file, choose `g++.exe` so IntelliSense matches the compiler used to link the C++ standard library.



5. Set **IntelliSense mode** to `windows-gcc-x64` and set the C++ standard to `C++17` unless your course specifies another standard. These settings are saved in `.vscode\c_cpp_properties.json`.

IMPORTANT: This page configures IntelliSense only. Build and debug commands must also use the `w64devkit` compiler and debugger.

6.5 Build, Run, and Debug

1. Open **Terminal > New Terminal** and compile the active C++ file:

```
g++ -std=c++17 -Wall -Wextra -g .\hello.cpp -o .\hello.exe  
.\hello.exe
```

The program should print:

```
Hello, World!
```

2. For one-key build and debugging, press `F5` or open Run and Debug. The first time, select **C/C++: g++.exe build and debug active file**. VS Code creates or updates `.vscode/tasks.json` and `.vscode/launch.json`.

Check that the generated settings agree with the IntelliSense configuration:

```
compilerPath: ...\\w64devkit\\bin\\g++.exe  
build task command: ...\\w64devkit\\bin\\g++.exe  
miDebuggerPath: ...\\w64devkit\\bin\\gdb.exe  
IntelliSense mode: windows-gcc-x64  
C++ standard: c++17
```

Add a breakpoint by clicking in the editor gutter, then press F5. A successful debug session builds hello.exe before starting GDB and stops at the breakpoint.

6.6 Troubleshooting Checklist

- 1. 'g++' is not recognized:** Confirm that Path contains the complete w64devkit\bin folder, then fully exit and reopen VS Code.
- 2. Red underlines remain although compilation works:** Reopen C/C++: Edit Configurations (UI) and set Compiler path to the same g++.exe used by the build task.
- 3. Debugging does not start:** Confirm that gdb.exe exists, miDebuggerPath points to it, and the build task successfully creates the .exe file.
- 4. w64devkit was moved:** Update Path, compilerPath, the build task command, and miDebuggerPath to the new location.
- 5. C versus C++:** Use gcc for .c files and g++ for .cpp files. The g++ driver automatically links the C++ standard library.

After completing these checks, VS Code provides C/C++ code completion, diagnostics, compilation, terminal execution, breakpoints, and GDB debugging with the same w64devkit toolchain.