Getting started with Python

Welcome to the University of Arizona Applied Math program! This document will provide instructions and tips on how to get set up with GitHub and Python when using a Windows machine.

WSL

Python originated in a Unix-based (Mac OS, Linux) environment, and so many of its tools and libraries are optimized for Unix-based systems. While Python is designed to be cross-platform and can run on Windows, it is common for those running Python on Windows to encounter compatibility issues and bugs that don't arise on Unix-based systems.

To avoid these issues, we recommend that you install WSL (Windows Subsystem for Linux). WSL is a compatability layer that allows you to run Linux natively on Windows 10 and 11. It is simple to get running, and we'll go through the steps to do so here. Don't worry if you have no experience on Linux; it is very easy to learn all the commands you need to write and run Python scripts.

1. *Enabling WSL*
   (a) Open the Start menu and search for "Turn Windows features on or off".
   (b) Click on it.
   (c) In the "Windows Features" window, scroll down until you see "Windows Subsystem for Linux" in the list.
   (d) Check the box next to it to enable the feature.
   (e) You should also enable "Virtual Machine Platform."
   (f) Click "OK" to save your changes.
2. *Installing Ubuntu (a Linux operating system)*: Now that WSL is enabled, you can install Ubuntu on your computer. Note that this won't overwrite your Windows operating system, but will install Ubuntu as an app that you can open from inside Windows.
   (a) Open the Microsoft Store app on your computer.
   (b) Search for "Ubuntu" in the search bar and select "Ubuntu" from the search results.

   (c) Click the "Get" buton to download and install Ubuntu on your computer.
   (d) Wait for the installation to complete.

3. *Launching Ubuntu:* Now that Ubuntu is installed, we can open it and use it.
   (a) Press the Windows key and type "Ubuntu" in the search bar.
   (b) Select "Ubuntu" from the search results to launch it.
   (c) Wait for Ubuntu to start up and create a new user account with a username and password when prompted.

4. *Install necessary Linux updates*:
   (a) In the Ubuntu terminal, type the command `sudo apt update` and then press `Enter` to update the package list. Typing a command and hitting enter in the command line is called "running" the command. This command updates the list of packages that come with Ubuntu.
   (b) Run the command `sudo apt upgrade`. This command installs any necessary updates.

Finding files in WSL

WSL has its own independent home and file system apart from Windows, but you can still access files in Windows from your Ubuntu terminal by navigating through the `mnt` directory. For example, if you want to navigate to your `c` drive (which is where your regular Windows files are stored) you can use the command `cd` (change directory) as follows:

```
cd /mnt/c
```

to change to your standard head directory. If you want to access your desktop, you can find it at `/mnt/c/Users/<your username>/Desktop`, and would navigate to it in Ubuntu terminal with

```
cd /mnt/c/Users/<your username>/Desktop.
```

This is a little annoying and slightly counterintuitive, but it will ultimately make your life much easier by helping you avoid the headaches that come from running Python natively on Windows.

Additionally, you can change the default directory that your Ubuntu terminal boots into. We recommend you do so, as it will make your life simpler.

1. Open your Ubuntu terminal
2. Type `cd ~`. This will take you to the home directory set up for you by Ubuntu.
3. Type `sudo nano .profile` to open your profile file. This file controls many of the default Ubuntu terminal settings.

4. Add the path of your desired default boot directory to the end of the file. For example, if you wanted to change teh default boot directory to be your Desktop, you would add this line to the end of the file:

   ```
   cd /mnt/c/Users/<your username>/Desktop
   ```

Git

Git is a *version control* tool that helps you keep track of changes to your files. It lets you save different versions of your work, switch between them, and easily revert to previous versions if needed.

Github is a website where you can store your git projects online. It makes it easy to share your work and collaborate on projects, because it allows multiple people to work on the same files at the same time and keeps track of everyone's changes. It provides methods to easily integrate those changes together into the main version of the files. It's a widely-used tool in applied math and computational sciences, and many researchers publish their code bases on Github.

In Math 589, you will use Github to submit programming assignments and receive feedback, so it's important to familiarize yourself with it. You will *push* your code from your local machine to your online Github repository, and then the most current version of your code will be *pulled* by the instructor from your online repository to their local machines for grading.

Setting up GitHub

1. *Sign up for GitHub.* If you already have a GitHub account, sign in to it and continue to the next step. If not, create a GitHub account at `https://github.com/`.

2. *Set up your Git credentials.* You will need to set up your Git `user.name` and `user.email`. Your `user.name` should be your actual name, and `user.email` must be the email associated with your GitHub account. To do this, open Ubuntu terminal and run the following:

   ```
   git config --global user.name "your name"
   git config --global user.email "your email"
   ```

3. *Create an SSH key.* This step only needs to be done once on each computer you want to use to acces your GitHub repositories. If you have multiple repositories on the same computer, you do *not* need to repeat this for each one. To create an SSH key, enter the following command in your Ubuntu terminal:

   ```
   ssh-keygen -t ecdsa -b 256
   ```

   Press the `Enter` key to accept the default file location. It will then prompt you to enter a Password, but you can press `Enter` again to skip this step if you don't want a Password. The key will then be created. The file for the key will be placed in the `/home/<your username>/.ssh` directory.

4. *Add your SSH key to your GitHub account*:
   (a) From Github, click on your profile icon in the upper right corner.
   (b) Click on “Settings” toward the bottom of the menu.
   (c) Click on “SSH and GPG keys” on the left.
   (d) Click the green “New SSH key” button.
   (e) Make a title for this key (it doesn’t matter what you put, but it may be a good idea to specify which machine this key belongs to)
   (f) Make sure **Key type** is set to `Authentication Key.`
   (g) Using the file explorer, nagivate to the `.ssh` directory. You can see your WSL files in the explorer by typing `\\wsl$` in the File Explorer address bar, and then clicking into the `Ubuntu` directory.
   (h) Open the *public key* file. This file should be called `id_ecdsa.pub`; do *NOT* use `id_ecdsa` (without the `.pub` extension). Copy the contents of this file and paste it into the `Key` field on GitHub.
   (i) If you’re having trouble navigating to the `.ssh` folder in explorer, you can print out the contents of the file by running this command in your terminal: `cat ~/.ssh/id_ecdsa.pub`
   (j) Once you’ve pasted the context of `id_ecdsa.pub` into GitHub, press the green **Add SSH key** button.
   (k) In your terminal, run
   ```
   ssh-add ~/.ssh_ecdsa
   ```
   If you get an error that says “Could not open a connection to your authentication agent,” then run
   ```
   eval $(ssh-agent)
   ssh-add ~/.ssh/id_ecdsa
   ```
   (l) To verify that this worked, you should see your new ssh key in the file `~/.ssh/known_hosts`.

CLoning GitHub repositories

A GitHub repository is a Git project that is stored on GitHub. You can clone a repository to your own machine, make changes to the files locally and, depending on the permission settings of the remote (online) repository, push changes back to the remote repository. We’ll practice this by cloning the repository set up for this workshop.

- Navigate to `https://github.com/jcallahan4/arizona_programming_workshop`

Python environments

What Are Python Environments?

A Python environment is a self-contained directory that contains a specific version of Python and a set of packages. When you install a Python package, it gets installed to the directory of your active environment. It helps you manage dependencies for different projects and avoid conflicts between them.

Why Use Python Environments?

1. Isolation Python environments create isolated spaces for your projects. This means each project can have its own set of dependencies, without interference from other projects. Without environments, all projects share the same global Python interpreter and packages, leading to conflicts when different projects require different versions of the same package.

2. Compatibility Different projects often depend on different versions of packages. For example, one project might require `numpy` version 1.18 while another requires 1.20. Environments allow you to install specific versions of packages for each project, ensuring that your code runs smoothly and consistently across different setups.

3. Reproducibility By using environments, you can precisely control the versions of Python and packages used in a project. This makes it easier to reproduce results, whether you're sharing your code with collaborators or running it on a different machine.

Using Python Environments

Managing Python environments involves creating, activating, and deactivating them. This section provides a step-by-step guide to these processes using both virtual environments and Conda environments.

Creating an Environment

Virtual Environments To create a virtual environment using the built-in `venv` module, use the following command:

```
python3 -m venv myenv
```

This command creates a directory named `myenv` containing the virtual environment.

Conda Environments To create a new environment using Conda, use the following command:

```
conda create --name myenv
```

This command creates an environment named `myenv`. You can also specify Python and package versions if needed, for example:

```
conda create --name myenv python=3.9
```

Activating an Environment

Virtual Environments To activate a virtual environment, use the following commands:

On macOS/Linux:

```
source myenv/bin/activate
```

On Windows:

```
myenv\Scripts\activate
```

Once activated, your command line prompt will typically change to include the environment name, indicating that you are now using the environment.

Conda Environments To activate a Conda environment, use the command:

```
conda activate myenv
```

This will also change your command line prompt to indicate the active environment.

Deactivating an Environment

Virtual Environments To deactivate a virtual environment, simply use the command:

```
deactivate
```

This will revert your shell back to the global Python environment.

Conda Environments To deactivate a Conda environment, use the command:

```
conda deactivate
```

This will also revert to the base Conda environment or the system's default Python installation.

Additional Tips

1. Don't install packages into your base environment (i.e., without activating a specific environment first). This will keep installed packages from modifying your system files and prevent conflicts.

2. It's a good practice to use a descriptive name for your environments, reflecting the project or the purpose of the environment.

3. Always activate the appropriate environment before installing new packages or running your code, to ensure that you're working in the correct context.

Additional Resources

1. Python Virtual Environments

2. Conda Environments