Getting Started with Python for DSP

The goal of this preliminary tutorial is to get you set up with a functional Python environment. I will be starting with a fresh installation of Windows 10, but the steps should be similar, if not identical, if you are using Windows 11. If you are running macOS, please follow along as best as possible. Finally, if you are running Linux, depending on your distribution, you may have Python installed by default.

Regardless of your system, if you run into any issues at all during the setup process, please reach out to Aidan Sharpe via email:

sharpe23@students.rowan.edu subject line: **DSP Help - FULL NAME**.

The Python Language

Recall your Computer Science & Programming and Introduction to Embedded Systems classes. You are familiar with C and C++, where by compiling a `.c` or `.cpp` file, you generate an executable `.exe` file. Both C and C++ are considered *compiled languages* for this reason. You can email the `.exe` file to a friend, and without any code (or even a compiler) they can run your file on their machine. When this file is executed, it runs in its own *process* on the operating system, which you can see by opening your task manager while the program is still running.

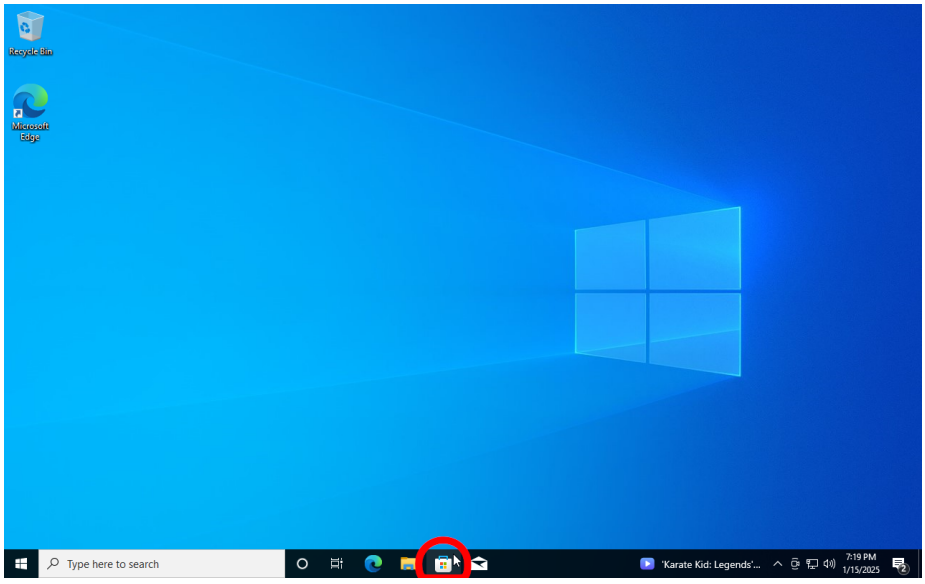
Python is *not* a compiled language. Instead, it is what we call an interpreted language. Rather than creating an executable file, the code is run line-by-line by a program called an interpreter. In the next section, we will install the Python Interpreter. Importantly, since no executable file is created, anyone who wants to run your code will have to run it with their own Python Interpreter. Additionally, unlike a compiled language, the program does not have its own OS process. In the case of Python 3, your program will run in the `python` process.

Installing Python

Before we begin writing any code, we need to install the Python Interpreter. These installation instructions are, as noted prviously, targetted at Windows 10/11 users. If you happen to be running Linux, you probably already have a Python interpreter installed, especially if you are running a "just works" distribution such as Ubuntu or Fedora. If for whatever reason your Linux installation does not have Python, simply install it from your package manager. If you happen to be running macOS, please follow the installation instructions on python.org.

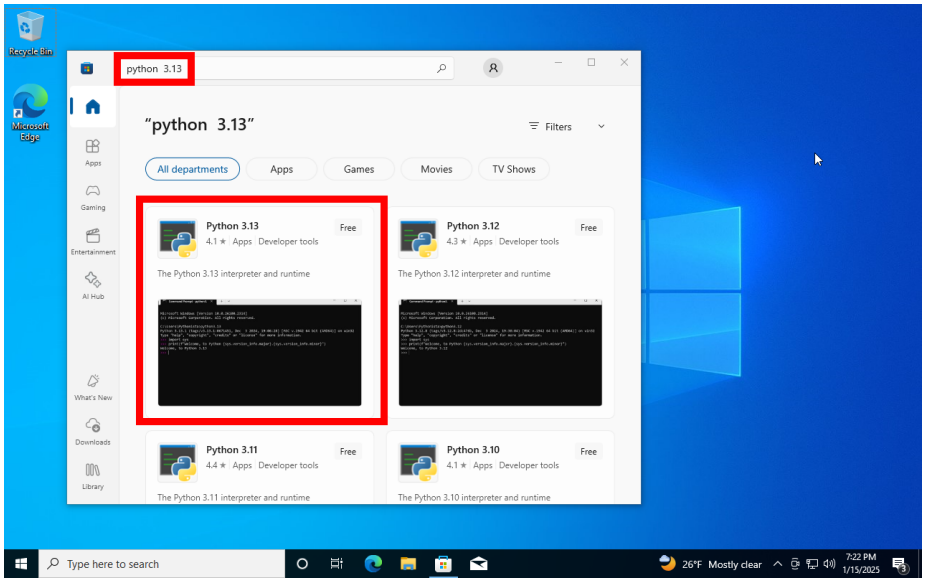
1. Open the Microsoft Store

On you taskbar or “Start” menu, open the “Microsoft Store” application.



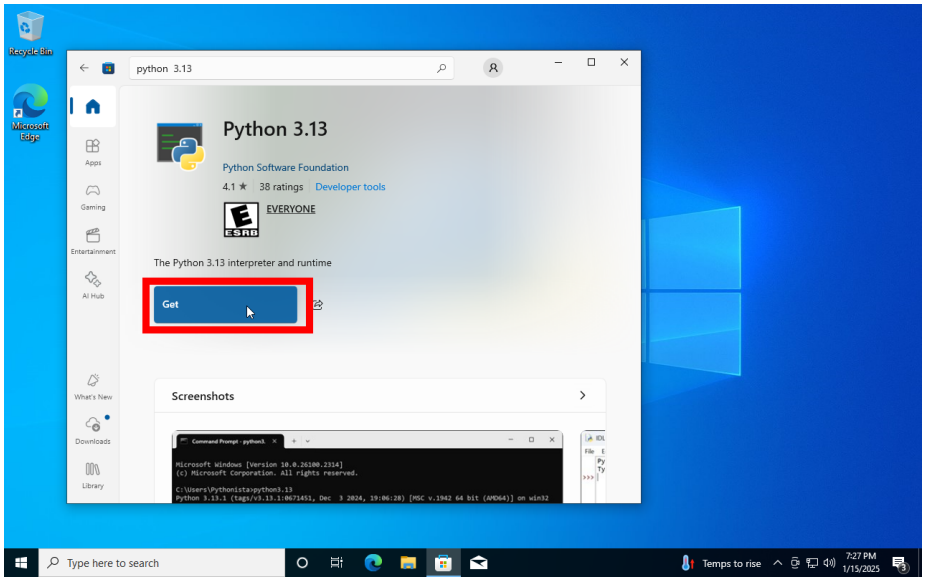
2. Search for “python 3.13”

In the Microsoft Store, search for “python 3.13” and click the result titled “Python 3.13”. This is the interpreter and runtime we will be using.



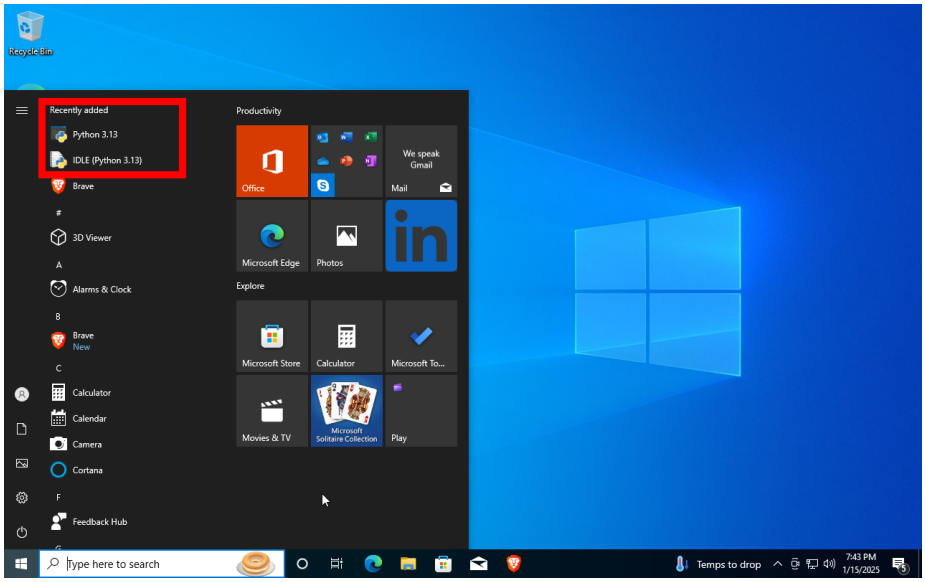
4. Click “Get”

Click the “Get” button to install the application. Wait for the installation to complete.



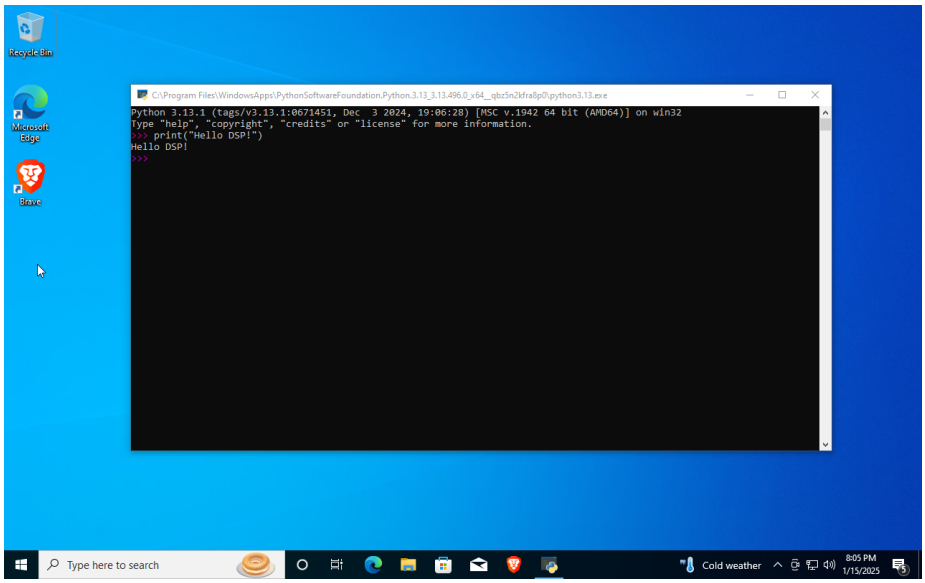
5. Verify installation

Open your “Start” menu and see if Python 3.13 and IDLE are shown.



Hello World

Start by opening "Python 3.13". This is the *Python Interpreter* we mentioned earlier. When run directly as an application, we are met with the *Python Shell*. Here, we can type Python code, and it will be executed as we go. For example, we can write a one-line "Hello, World" style program simply by typing `print("Hello, World!")` and hitting the enter key. The text "Hello, World!" will be printed, and we are prompted again on the following line. We can change the text inside the quotes to whatever we want, and that text will be printed out as well. Congratulations, you have run your first Python code!



Basic Concepts - Variables, Functions, and Logic

You are likely used to declaring a variable in the following manner:

```
int x;
```

where you specify a data type and a variable name. In languages like C and C++, the variable x will always be an integer within the scope that it was declared as one. This system of declaration is called *static typing*, which simply means that the data type of a variable—integer in this case—cannot change. Python is what is called a *dynamically typed* language, meaning that variables can change their data type at any time. For example, the following C code would generate an error

```
int x = 5;
x = "hello";
```

but the following Python code would be valid:

```
x = 5
x = "hello"
```

Two things should stand out when comparing the C and Python samples above. First, the Python code does not have semi-colons since statements in Python are terminated by a new line. Second, the Python code should stand out in that no type is explicitly stated. That is not to say that the variable does not have a type, but rather the type can change based on the value being stored. In the case of our example, x is initially an integer, but then changed to a string.

Rarely, do we see Python code standing alone outside a function like this, however. This is because functions allow us to re-use complex code segments. We create a function by using the `def` keyword, followed by the function name, a parameter list in parenthesis, and the line ends with a colon (:). The following line starts the body of the function, and it *must* be tabbed in once more than the line with the definition.

IMPORTANT: In Python, changing the tabbing can change the behavior of the code, so be careful. Be patient, building the habit of paying attention to tabbing can take some time.

Below is a simple function that adds two variables (a and b) and returns the result.

```
def add(a, b):
    return a+b
```

The final basic concept is control logic. This will allow our program to make decisions based on the current program state. The simplest example is the `if-elif-else` statement. For example, if we wanted to find the maximum value between two variables a and b we might write the following:

```
def max(a,b):
    if a > b:
        return a
    else:
        return b
```

With this simple case, either a is greater than b and a is the max, or a is not greater than b and b is the max. It doesn't matter what we return if the two values are equal, because the result will be indistinguishable.

Say we want to calculate our letter grade based on our number grade. We may write a function:

```
def letter_grade(number_grade):
    if number_grade >= 90:
        return 'A'
    elif number_grade >= 80:
        return 'B'
    elif number_grade >= 70:
        return 'C'
    elif number_grade >= 60:
        return 'D'
    else:
        return 'F'
```

Here, we exhibit two new Python concepts: the `elif` keyword and "snake case". The `elif` keyword is Python's version of the `else if` you may be familiar with in other programming languages. Snake case (often stylized as snake_case) simply means all lower case with underscores between words. It is recommended that for Python code all variable and function names be written using snake_case as opposed to something like camelCase or PascalCase. According to PEP 8 - the style guide for Python, snake case is correct Python style.