1 - Unit 4 - Lesson 1

October 19, 2023

The creation of the lessons in this unit relied heavily on the existing lessons created by Mrs. Fitz-Zaland as well as the **lecture series** produced by Dr. Milaan Parmar.

The single most important skill for a computer scientist is problem solving. Problem solving means the ability to formulate problems, think creatively about solutions, and express a solution clearly and accurately. As it turns out, the process of learning to program is an excellent opportunity to practice problem-solving skills. On one hand, you will be learning to program, a useful skill by itself. On another hand, you will use programming as a means to an end.

1 Python Programming

Python is a powerful multipurpose programming language created by *Guido van Rossum*.

It has a simple and easy-to-use syntax, making it a popular first-choice programming language for beginners.

Many tech companies write programs using Python. It's used by Intel, IBM, NASA, Pixar, Netflix, Facebook, and Spotify. It's one of the four main languages used by Google. Python was used to write YouTube, Reddit, Pinterest, and Instagram. It's used heavily in academic research, particularly in biology, mathematics, and physics. Python is also the standard introductory language for many university computer science programs.

If you started using Python in IT8 or IT9, some of the information in the first two lessons may be a review. Don't worry, we'll be covering this information in more depth than before and you'll be writing more complex code in no time.

In this lesson, you'll learn about the history of programming languages and you'll start using Python.

1.1 History of Programming

Before we dive into writing programs in Python, you need to know a little more about the history of programming. The first programming language was created in 1843 before computers even existed. Ada Lovelace, a female mathematician, developed an algorithm that she wrote down on paper to remember since there were no computers. Needless to say, programming languages have evolved tremendously.

Video #1: Watch this video.

Video #2: Watch this video.

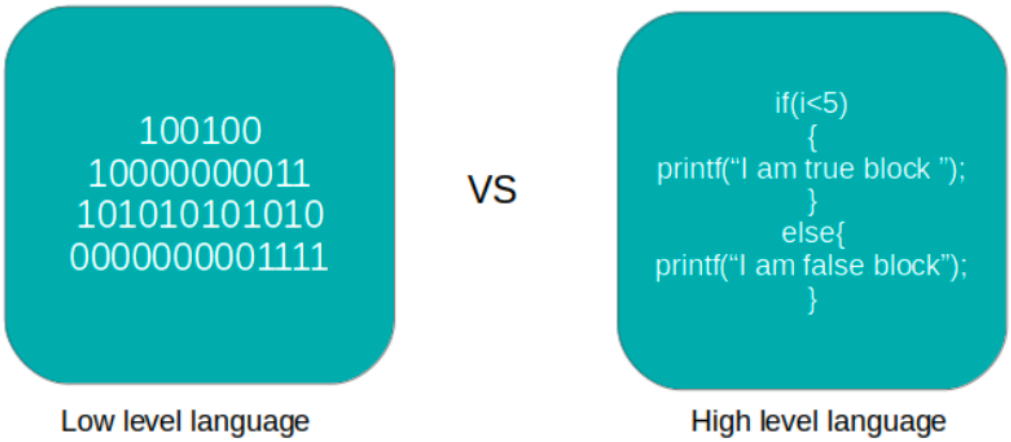
(You can also find these videos in the Lesson folder)

1.2 The Python Programming Language

There are hundreds of programming languages, each with their own unique strengths and applications. Ultimately, the best programming language for you depends on what you're looking to achieve.

As I mentioned earlier, the programming language you will learn in this unit is Python. Python is an example of a high-level language; other high-level languages you might have heard of are C, C++, Perl, and Java.

There are also low-level languages, sometimes referred to as "machine languages" or "assembly languages." Loosely speaking, computers can only run programs written in low level languages. So, programs written in a high-level language must be processed before they can run. This extra processing takes some time, which is a small disadvantage of high- level languages.



However, **the advantages of high-level languages are enormous**:

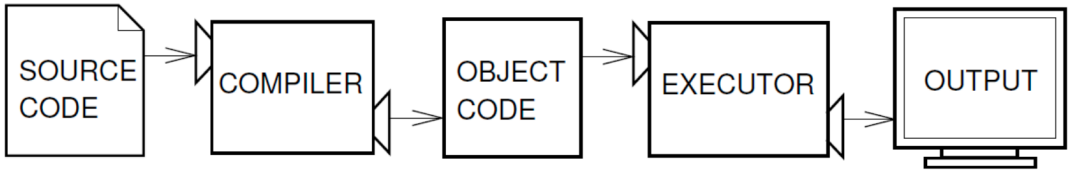
- First, it is much easier to program in a high-level language. Programs written in a high- level language take less time to write, they are shorter and easier to read, and they are more likely to be correct.
- Second, high-level languages are portable, meaning that they can run on different kinds of computers with few or no modifications. Low-level programs can run on only one kind of computer and have to be rewritten to run on another. Due to these advantages, almost all programs are written in high-level languages. Low level languages are used only for a few specialized applications. Two kinds of programs process high- level languages into low-level languages: **interpreters** and **compilers**.

1.2.1 Interpreters and Compilers

An **interpreter** reads a high-level program and executes it, meaning that it does what the program says. It processes the program a little at a time, alternately reading lines and performing

computations.

A **compiler** reads the program and translates it completely before the program starts running. In this context, the high-level program is called the **source code**, and the translated program is called the **object code** or the executable. Once a program is compiled, you can execute it repeatedly without further translation.



1.2.2 Interactive and Script mode

Python is considered an interpreted language because Python programs are executed by an interpreter. There are two ways to use the interpreter: **interactive mode** and **script mode**.

In interactive mode, you type Python code, and the interpreter displays the result:

```
[1]: 1+1
```

```
[1]: 2
```

Working in interactive mode is convenient for testing small pieces of code because you can type and execute them immediately. But if you want to write code to be used again and again, you'll probably want to save your code as a **script** so you can modify and execute it in the future. By convention, Python scripts have names that end with the file extension **.py**.

1.2.3 What is a Program?

A **program** is a sequence of instructions that specifies how to perform a computation. The computation might be something mathematical, such as solving a system of equations or finding the roots of a polynomial, but it can also be a symbolic computation, such as searching and replacing text in a document or something graphical, like processing an image or playing a video.

The details look different in different languages, but a few basic instructions appear in just about every language:

- **Input:** Get data from the keyboard, a file, the network, or some other device.
- **Output:** Display data on the screen, save it in a file, send it over the network, etc.
- **Math:** Perform basic mathematical operations like addition and multiplication.
- **Conditional Execution:** Check for certain conditions and run the appropriate code.
- **Repetition:** Perform some action repeatedly, usually with some variation.

Believe it or not, that's pretty much all there is to programming. Every program you've ever used, no matter how complicated, is made up of instructions that look pretty much like these.

You can think of programming as the process of breaking a large, complex task into smaller and smaller subtasks until the subtasks are simple enough to be performed with one of these basic instructions.

1.3 Interactive Development Environment

When you write a program, in any programming language, you usually type your code into an Interactive Development Environment or IDE for short. An IDE is a little bit like a word processor. An IDE will typically allow you to:

- write, edit, and save your code
- build and run your code (via an interpreter)
- debug your code

IDEs may also include other desirable features and tools. Choosing between IDEs is often a matter of preference, but the unique features of an IDE often play a role in decision making.

The first difference between my approach and Mrs. FitzZaland's is that you will be working with a cloud-based version of JupyterLab in my class, rather than IDLE.

There are a few reasons why I have chosen JupyterLab rather than IDLE:

- Allows for more interactive and exploratory coding
- Provides live data visualization and analysis
- Makes it easy to collaborate and share your work
- It is used extensively by the python community, especially in the sciences

2 Deepnote

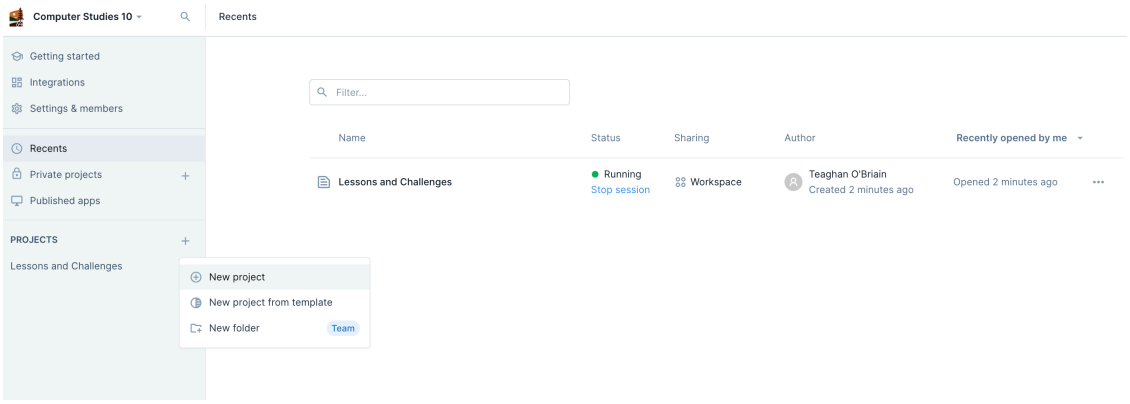
Deepnote is a cloud-based workspace that allows you to create *notebooks*, python script files, and run those script files from a terminal - all in the cloud! This means that you can get started with Python without downloading anything and you can seemlessly move between your school computer and your computer at home.

2.1 Getting Started with Deepnote

First things first, provide your teacher with your email address and they will send you an invite to join the class Workspace on Deepnote. This will prompt you to create an account with your personal or school email address. When creating your account, please **do not use your actual name for your Deepnote profile**. Instead, you can stick with the same notation as your school

email. For example, if your email is tea1010@sd64.org, you can use "Tea" as your first name and "1010" as your last name.

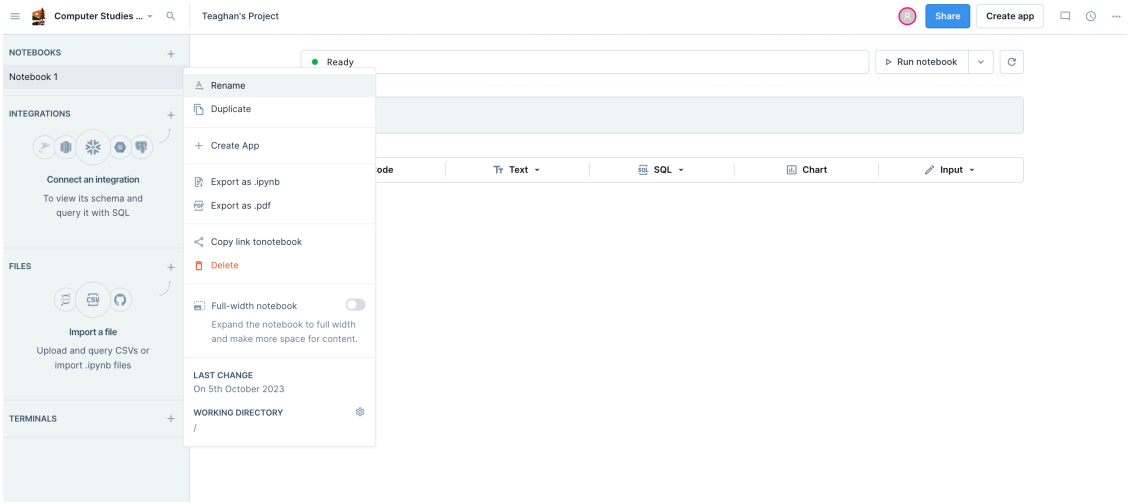
Once you have joined the workspace, you can **create your own project** by selecting `+ >Projects > New Project` within the left-hand panel:



1. **Try creating your own project and please check with your teacher to make sure it is set-up correctly.** Again, please do not use your full name when naming your project.

Once you have created your project, it will automatically start you in a new notebook, which should look similar to the screenshot below.

2. **Rename this notebook with the name** `My First Notebook`



Within a notebook like this, you can write python code, write "normal text" alongside your code, and save your work to share!

2.2 Running code in a notebook

3. **Enter the following text into a cell and then press Shift + Enter:**

```
[2]: print('Hello world!')
```

```
Hello world!
```

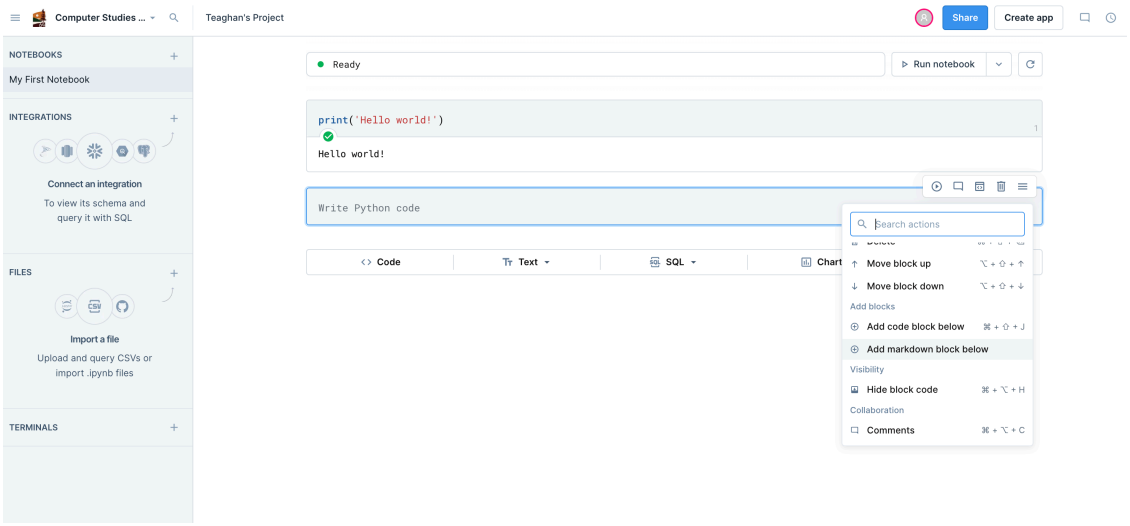
This is an example of a **print statement**, although it doesn't actually print anything on paper. It displays a result on the screen. You should get the response `Hello world!` on the next line. If you forget an apostrophe or bracket, you'll get an error message and have to retype it.

The quotation marks in the program mark the beginning and end of the text to be displayed; they don't appear in the result.

The parentheses indicate that **print** is a function. We'll get to **functions** a little later.

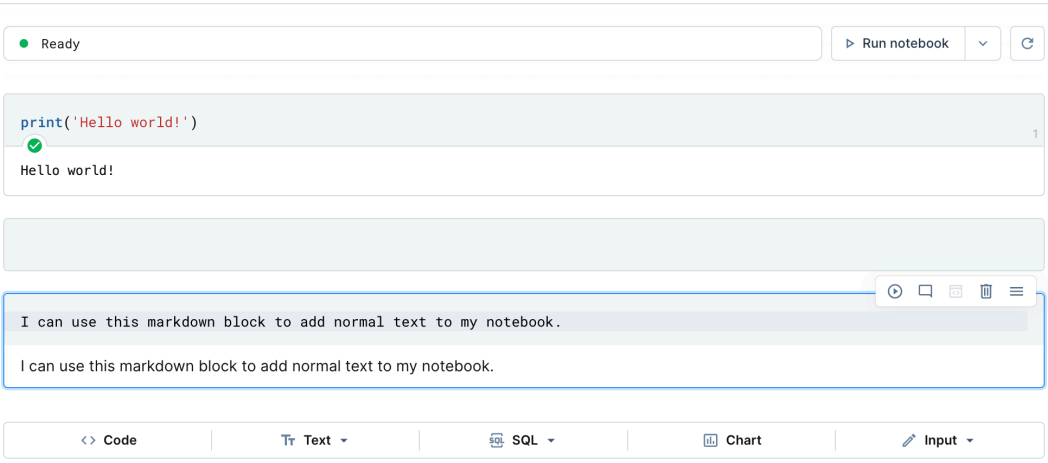
2.3 Markdown

After you execute the cell, the notebook will automatically create a new *code* cell for you to provide a new command. You can also create a **Markdown** cell, which allows you to format regular text (like what you are reading right now). There are multiple ways to accomplish this. One way is to select the 3 horizontal lines at the right of your current cell, and select `Add markdown block below`



4. **Create a new Markdown block and add some text like in the screenshot below then press Shift + Enter.**

Note that the next cell that is automatically created will also be Markdown now. If you want to create a `code` cell, you will have to create a new code cell. Feel free to experiment with these functions to get comfortable with creating and running different types of cells.



Now you will be able to include both code and regular text into one file, which will make it easy for you to organize and submit your programming work in this class.

If you're interested in learning how to format in Markdown, check out this page.

2.4 Arithmetic Operators

After "Hello World", the next step in learning to program is arithmetic. Python provides **operators**, which are special symbols that represent computations like addition and multiplication.

The operators `+`, `-`, `*`, `/` perform addition, subtraction, multiplication, and division as in the following examples:

[3]:
```
40 + 2
```

[3]: 42

[4]:
```
43 - 1
```

[4]: 42

[5]:
```
6 * 7
```

[5]: 42

[6]:
```
84 / 2
```

[6]: 42.0

2.5 Values and Types

A value is one of the basic things a program works with, like a letter or a number. Some values we've seen so far are 2, 42.0 and 'Hello World!'. These values belong to different **types**:

- 2 is an **integer**
- 42.0 is a **floating-point number**
- 'Hello World!' is a **string** (so-called because the letters it contains are strung together)

If you are not sure what type a value has, the Python shell can tell you.

5. **Enter the following code into individual code cells and run the cells separately to determine the type of each value.**

```
type(2)
```

```
type(42.0)
```

```
type('Hello World')
```

In the results, you should have gotten the category of values. Not surprisingly, integers belong to the type `int`, strings belong to `str`, and floating-point numbers belong to `float`.

What about values like `'2'` and `'42.0'`? They look like numbers, but they're in quotation marks like strings.

6. **Enter the following code into code cells and Shift + Enter enter after each line.**

```
type('2')
```

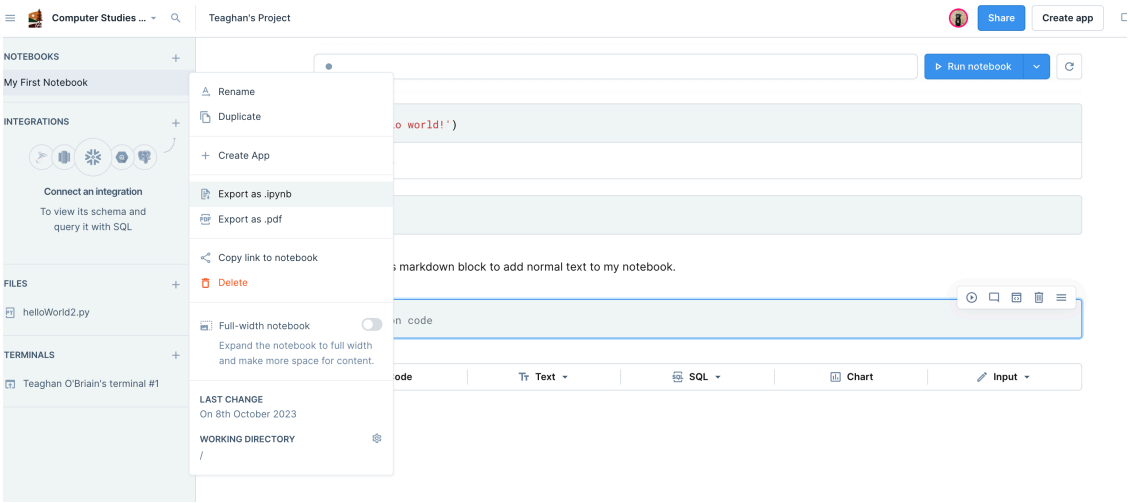
```
type('42.0')
```

Do they belong to `str` or `int` or `float`?

2.6 Saving

Your notebook will save automatically. You can also download the **.ipynb** file to share your notebook. Typically, you can export your notebook as a **.pdf**, which will format the code and markdown cells nicely. However, I have not found a way to do this nicely within Deepnote. Stay tuned!

6. **Try downloading your notebook as an `.ipynb` file.**



3 Formal and Natural Languages

Natural Languages are the languages people speak, such as English, Spanish, and French. They were not designed by people (although people try to impose some order on them); they evolved naturally.

Formal Languages are languages that are designed by people for specific applications. For example, the notation that mathematicians use is a formal language that is particularly good at denoting relationships among numbers and symbols.Chemists use a formal language to represent the chemical structure of molecules. And most importantly, programming languages are formal languages that have been designed to express computations.

Formal languages tend to have strict **syntax** rules that govern the structure of statements. For example, in mathematics the statement `3 + 3 = 6` has correct syntax, but `3 + = 3$6` does not. In chemistry H_20 is a syntactically correct formula, but ${}_2Zz$ is not.

Syntax rules come in two flavors, pertaining to **tokens** and **structure**.

Tokens are the basic elements of the language, such as words, numbers, and chemical elements. One of the problems with `3 + = 3$6` is that `$` is not a legal token in mathematics (at least as far as I know). Similarly, ${}_2Zz$ is not legal because there is no element with the abbreviation Zz.

*Ex: This is @ well-structured Engli$h sentence with invalid t*kens in it.*

The second type of syntax rule pertains to the **structure** of a statement; that is, the way the tokens are arranged. The statement `3+ = 3` is illegal because even though `+` and `=` are legal tokens, you can't have one right after the other. Similarly, in a chemical formula the subscript comes after the element name, not before.

Ex: This sentence all valid tokens has, but invalid structure with.

When you read a sentence in English or a statement in a formal language, you have to figure out what the structure of the sentence is (although in a natural language you do this subconsciously). This process is called **parsing**.

Although formal and natural languages have many features in common—tokens, structure, syntax, and semantics—there are some differences:

- Formal languages are designed to be nearly or completely unambiguous, which means that any statement has exactly one meaning, regardless of context.
- Formal languages are less redundant and more concise.
- Formal languages mean exactly what they say.
- Finally, the details matter. Small errors in spelling and punctuation, which you can get away with in natural languages, can make a big difference in a formal language.

4 Debugging

Programmers make mistakes. For whimsical reasons, programming errors are called **bugs** and the process of tracking them down is called **debugging**.

Programming, and especially debugging, sometimes brigs out strong emotions. If you're struggling with a difficult bug, you may feel angry, despondent, or embarrassed.

Preparing for these reactions might help you deal with them. Learning to debug can be frustrating, but it is a valuable skill that is useful for many activities beyond programming.

5 Glossary

In this lesson, I introduced a lot of new terms, and it can get confusing. This list may help you to understand these terms. Please read it carefully.

- **algorithm**: A general process for solving a category of problems.
- **bug**: An error in a program.
- **compile**: To translate a program written in a high-level language into a low-level language all at once, in preparation for later execution.
- **debugging**: The process of finding and removing any of the three kinds of programming errors.
- **executable**: Another name for object code that is ready to be executed.
- **formal language**: Any one of the languages that people have designed for specific purposes, such as representing mathematical ideas or computer programs; all programming languages are formal languages.
- **high-level language**: A programming language like Python that is designed to be easy for humans to read and write.
- **interactive mode**: A way of using the Python interpreter by typing commands and expressions at the prompt.
- **interpret**: To execute a program in a high-level language by translating it one line at a time.
- **low-level language**: A programming language that is designed to be easy for a computer to execute; also called "machine language" or "assembly language."
- **natural language**: Any one of the languages that people speak that evolved naturally.
- **object code**: The output of the compiler after it translates the program.
- **parse**: To examine a program and analyze the syntactic structure.
- **print**: A function that causes the Python interpreter to display a value on the screen.
- **problem solving**: The process of formulating a problem, finding a solution, and expressing the solution.
- **program**: A set of instructions that specifies a computation.
- **prompt**: Characters displayed by the interpreter to indicate that it is ready to take input from the user.
- **script**: A program stored in a file (usually one that will be interpreted).
- **script mode**: A way of using the Python interpreter to read and execute statements in a script.
- **semantics**: The meaning of a program.

- **semantic error**: An error in a program that makes it do something other than what the programmer intended.
- **source code**: A program in a high-level language before being compiled.
- **syntax**: The structure of a program.
- **syntax error**: An error in a program that makes it impossible to parse (and therefore impossible to interpret).
- **token**: One of the basic elements of the syntactic structure of a program, analogous to a word in a natural language.

5.0.1 (optional) Installing Python and JupyterLab locally

If you happen to want to install Python on your own computer (**which is not a requirement**), go to www.python.org and select the Download link. Here you should be able to download the latest Python version for your system. After the file downloads, run the installer.

To install JupyterLab (**also not a requirement**) on your own computer follow these instructions.

Next you should do Challenge 23.

[ ]: