Introduction to Computer Science - Coding in Python

Sophia Jin

January 2025

1 About this Program

The STEMPact: Youth to Youth high school mentorship program has been created by the North Liberty Youth Council. The purpose of this program is to introduce middle and upper-elementary school students to areas of STEM. Oftentimes, kids are daunted by STEM, but the areas within STEM are in fact quite approachable and interesting. We aim to give a gentle introduction to the areas of Computer Science, Chemistry, Mathematics, Physics, and Biology in a series of five lectures held over the span of January 2025 through May 2025. The Youth Council members who have worked on this project are listed below:

1. Sophia Jin (team lead): Lecture 1 - Computer Science
2. Luqmaan Khan: Lecture 2 - Chemistry
3. Jiwon Kim: Lecture 3 - Biology.
4. Anaya Patil: Lecture 4 - Math
5. Asher Bonner: Lecture 5 - Physics
6. Vincent Ni: Helped with set-up

2 Lecture Preview

Today we will be talking about the basics of Python coding. The topics included in today's lecture are print statements, escape characters, math in print statements, commenting, variables, conditionals, and user input. Python is usually the most approachable coding language for beginners. It is simpler than other languages such as Java and C++, but for our purposes, it will work just fine. Python is one of the most commonly used coding languages. For example, it is often used in research settings and software development.

We will also work on some practice problems after the lecture, and I will go

over some CS-related courses that you can take in high school as well as some resources for additional coding exploration.

Remark: Some of you might have heard about or used Scratch, which is a type of block coding. Scratch is a very informative and fun starting tool for young learners, but it is also a very oversimplified "language". It is not applicable in most contexts. This is why we'll focus on Python, a text-based coding language. Python is easy to learn compared to Java and C++, and it is very relevant in the scientific world, as it is widely applied in many areas from astrophysics to machine learning.

3 Coding Software

Usually, I use the VS Code IDE, but it is quite hard to install, and I'm not sure if it works on Chromebooks. Therefore, we will use a website called Replit, which functions similarly to VS Code, for our purposes.
The following are steps for creating a Replit account and using Replit:

1. Please go to replit.com and click the button "Log In"
2. Click "Continue with Google"
3. Click your email
4. If prompted, click "Continue"
5. Once you are at your home screen, click "Create Repl" and then "Python." Give your project a name if you like and then click "Create Repl." This will bring you to a new project.
6. You will write your code in main.py and the code will run when you click the green "Run" button. The results of the code will appear in the console.

4 Print Statements

Print statements simply print text onto the screen.
Example 1:

```
print("Hello World!")
```

The output of this command is:

```
Hello World
```

Notice that the print statement requires quotation marks to print the text. The quotation marks are not present in the final output. The type of quotation mark does not matter too much. You can use single quotation marks ‘ or double quotation marks “. However, you should be consistent (you cannot start a

print statement with a single quotation mark and end it with a double quotation mark).
Additionally, print statements insert a new line after the output, although it's not that obvious from the above example which only executes one print statement.
Example 2a:

```
print("Welcome to Session 1!")
print("I hope you enjoy the lecture!")
```

The output is:

```
Welcome to Session 1!
I hope you enjoy the lecture!
```

Notice that after the first statement is executed, the output of the second statement is on a new line. If you wanted to combine both into one line, you could simply include both sentences in the same print statement, like this:
Example 2b:

```
print("Welcome to Session 1! I hope you enjoy the lecture!")
```

This just prints

```
Welcome to Session 1! I hope you enjoy the lecture!
```

5 Escape Characters

Some symbols, such as the quotation mark, are "illegal" in strings. For example, the quotation mark is needed to enclose the string, so if you had a quotation mark in the string itself, the computer wouldn't know what the real string you wanted to type was. The computer would then generate an error. In order to type special symbols, you need to type an escape character. An escape character is a backslash \followed by the character you want to insert. The entire escape sequence is treated as one whole character by the computer.

1. To output a single quotation mark, type \'
2. To output a double quotation mark, type \"
3. To create a new line without typing a new print statement, type \n
4. To output a backslash \character, type \\

Remark: We will talk about strings soon. It's basically just a sequence of characters and the string is enclosed by quotation marks which denote the string (the quotation marks themselves aren't included in the string).
Example 3:

```
print("\'")
print("\"")
print("Hi\nBye")
print("\\")
```

Output:

```
'
"
Hi
Bye
\
```

Notice that the third print statement prints Hi and Bye on two lines even though they are both included in the same print statement. The escape character \n saves us time because we doesn't need to type a new print statement to print out Bye. Personally, though, I think it's better to just type out two print statements, as the code looks better to understand.

6 Math in Print Statements

If you want your program to print out the result of $25 * 25$, you would write the print statement like so:
Example 4:

```
print(25 * 25)
```

Output:

```
625
```

We must take note of one key thing here: **The print statement for math operations does not include quotation marks. This is very important! Strings**, which are sequences of characters (letters, numbers, symbols, etc.) are enclosed by quotation marks in print statements. The strings don't have to be just one word; they can be several words or several sentences, as shown above in **Example 2b**. Strings are usually printed out as they are written.

Another important thing of note is that arithmetic operations in print statements follow the PEMDAS rule as in algebra.
Example 5:
Input:

```
print(10 + 10)
print("10 + 10")
```

Output:

```
20
10 + 10
```

From this, we see that without the quotation marks, the information inside the print statement is just treated like a normal math operation, and the computer executes the sum 10 + 10 which gives 20. However, in the second line of code, the computer does not calculate 10 + 10. It treats everything inside the print statement as a string, and it just prints the string as is, *without any reference to the contents of the string.*
Here are some other examples of arithmetic in print statements that all four operators (addition, subtraction, multiplication, division).
Example 6:
Input:

```
print(4 + 5)
print(4 - 5)
print(-1 - 7)
print(2 * 3)
print(1.5 * 11)
print(5 / 4)
print(2 / 3)
```

Output:

```
9
-1
-8
6
16.5
1.25
0.66666666666666
```

Note: Sometimes there may a slight round-off error. For example, if the answer to a question were 4, the output might appear as 3.9999999.

There are a few other operations that are useful to know about. The first is the modulus operator, which gives the remainder of a division equation. The modulus operator is denoted by %
Example 7:
Input:

```
print(5 % 3)
print(7 % 4)
print(8 % 9)
print(1.5 % 1)
print(0.8 % 0.6)
```

Output:

```
2
```

```
3
8
0.5
0.20000000000007
```

Notice that the last output has a slight round-off error.
Another operator is the exponentiation operator, denoted by **
Example 8:

```
print(2 ** 3)
print(0.5 ** 2)
```

Output:

```
8
0.25
```

The final operator is the floor division operator. This basically takes away everything behind the decimal point as the result of a division equation. Floor division is denoted by the symbol // (two forward slashes).
Example 9:

```
print(3 // 5)
print(9 // 7)
```

Output:

```
0
1
```

7 Commenting

Commenting is a pretty simple concept. When you "comment out" a line of code using the symbol #, the computer ignores that line of code and does not execute it. The computer just continues to the next line of code if there is another line of code. A few things to note here:

1. The # symbol comments out everything behind it. Thus, coders usually comment out a line of code by putting the # symbol at the beginning of the line or behind all the relevant code in the line.

2. The # symbol only comments out a line of code, not a block of code.

To comment out a block of code, we use quotation marks: you can use ' (single quotation marks) or " (double quotation marks). Either will suffice just like in print statements. Type three quotation marks at the beginning of the block that you want to comment out and type three quotation marks at the end of said block. In this way, the whole block will be commented out and thus ignored by the computer upon execution.

Remark: Comments are a good way to "save" lines of code that you might need later. Additionally, they can be used to write notes. For example, if a coder were working on a team project, they might leave some comments behind explaining the code so that the next team member who looks at the code can quickly understand what is going on. Writing comments also helps the coders themselves.
Example 10:

```
print("This is the first sentence")
#print("This is the second sentence")
print("This is the third sentence")
```

Output:

```
This is the first sentence
This is the third sentence
```

After printing the first statement, the computer ignores the second statement which is commented out, and the computer proceeds to print out the third statement.

8 Variables

In programming, we use variables to store information. Datatypes are the classification or categorization of data items. A datatype represents the kind of value that tells what operations can be performed on a particular datum. Some common datatypes include int (which means integers), float (numbers with a decimal point), str (an abbreviation of string which is a sequence of characters), and bool (an abbreviation of boolean, which represents truth values). You set a variable by writing the name of the variable, then an equal sign, and finally whatever you want to set the variable to. See the example below.
Example 11: Setting Variables

```
x = 8
school = "Liberty High School"
num = 0.76
eligibility_for_job = False
print(x)
print(school)
print(num)
print(eligibility_for_job)
```

Output:

```
8
Liberty High School
0.76
False
```

Notice that setting the variables to some values does not result in anything being printed. Think of it like work done behind-the-scenes that doesn't show up on the output screen. You can print out the variables using print statements, though. Additionally, notice that we don't use quotation marks to print out the variables.
The first variable, x, is set to 8, an integer number. Thus, the datatype of this variable is int. The second variable, school, is set to "Liberty High School." Thus, the datatype of this variable is str. The third variable, num, is set to 0.76, a decimal number. Thus, the datatype of this variable is float. The final variable, eligibility_for_job, is set to False. Thus, the datatype of this variable is bool.

You can also change the values stored in the variables. See the below example.
Example 12:

```
y = 7
print(y)
y = 9
print(y)
y = "Hello"
print(y)
```

Output:

```
7
9
Hello
```

These are the steps for the code execution:

1. y is set to 7
2. the variable y is printed. Notice that the letter y isn't printed but rather the value stored in y is printed.
3. y is set to 9
4. the variable y is printed
5. y is set to the string "Hello". Notice that the datatype of y changes from int to str.
6. the variable y is printed

Example 13: (from MoocFi)

```
number1 = 100
number2 = "100"
print(number1)
print(number2)
```

Output:

```
100
100
```

In this example, the variable number1 is set to 100, so number1 is of an int datatype. number2 is set to "100", so number2 is of a string datatype. Even though when both variables are printed, the output looks the same, one of them is an integer number and the other is a string. It would be best to rename the variables to something better, for example intnumber and strnumber.

8.1 Naming Variables

Notice how the variable named eligibility_for_job has underscore characters between words. When we write names of variables in Python, there are a few conventions to follow:

1. Name variables according to what they describe. Let's say that we want to create a variable that stores a book title. We could name the variable book or title but it would not be very good to name the variable x or y. If we named the variable descriptively, it'll make it easier later on to remember what the variable stores and what the variable's function is.

2. A variable name should begin with a letter. The variable name can contain letters, numbers, and underscores only.

3. For letters, only lowercase letters can be used. If a variable name has multiple words, use an underscore instead of a space to connect them.

9 Conditionals

The final topic we will talk about is conditionals. Conditional statements are also sometimes called "if" statements. A conditional starts with the word if and then follows with a condition and then a colon. After that is the block of code that is part of the conditional statement. If the condition is true, the block of code inside the if statement is executed. What makes a block of code "inside" the if statement? The code must be indented by pressing the tab button on your computer. Replit automatically indents for you. See the following example. **Example 14:**

```
x = 10
if x > 9:
    print("Double digit positive integer")
```

Output:

```
Double digit positive integer
```

Here are common operators used in conditional statements:

1. == (equal to)
2. != (not equal to)
3. > (greater than)
4. >= (greater than or equal to)
5. < (less than)
6. <= (less than or equal to)

You can also expand your if statement into an if-else or if-elif-else conditional. The next examples illustrate these conditionalsas well as the operators.
Example 15:

```
x = 7
if x == 9:
    print("x equals 9")
else:
    print("x does not equal 9")
```

Output:

```
x does not equal 9
```

The computer goes executes the first line and sets the variable x to 7. Then, it goes to the if statement. Since x does not equal 9, it goes to the else part of the conditional and executes that, thus printing the output above.
Example 16:

```
placement = "third"
if placement == "first":
    print("Gold medal!")
elif placement == "second":
    print("Silver medal!")
elif placement == "third":
    print("Bronze medal!")
else:
    print("No medal.")
```

Output:

```
Bronze medal!
```

This is a more complicated example. Lets walk through the steps:

1. The computer sets the variable placement to the string “third”
2. The computer goes to the conditional and checks the first condition which is placement == “first”. Since placement, which is set as “third”, is not the same as “first”, the computer goes to the second condition. Note that this isn’t exactly an if else statement as it contains multiple layers.

3. Since placement is not the same as "second", the computer goes to the third condition.

4. Since placement does store the string "third", the output is Bronze Medal!

10 User input

The command *input* reads in information typed in by the user. The computer can then use this information to do various things.
Example 17:

```
age = input("How old are you? ")
print(f"You are {age} years old.")
```

This program first displays the question "How old are you?" It then stores a number inputted by the user in the variable age. Finally, it prints a statement about the user's age.
Remark: If we simply type age instead of age in the print statement, the computer would output "You are age years old." To solve this problem, we write an f in front of the string and enclose the variable, age, in braces. This f-string format is generally what we use when we want to include a variable in a print statement.
Example 18:

```
name = input("What's your name? ")
school = input("What school do you go to? ")
print(f"{name} goes to {school}.")
```

Example 19:

```
number = input("Please input a number. ")
print(number + 1)
```

This program produces an error! We must take note of one thing. **The input function always returns a string.** In our case, a string is then stored in the variable number. Even though the user meant to type in a number, the computer thinks that the input is a string, because the input is typed in from our keyboard which contains characters. In order to force the computer to treat the input as an int, we must invoke casting.
Example 20:

```
num = int(input("Please input a number. "))
print(num + 1)
```

By enclosing our former input statement with the function int(), we force the computer to treat the user input as an integer and ensure that the variable num stores an int. Then, the code proceeds without error.
Similarly, you can cast use the float(), str(), and bool() functions to cast the user input to the respective datatype.

11 Practice Problems

These practice problems cover topics we've talked about today.
1. Write code that prints the following:

```
This is a practice problem.
```

2. Write code that prints the following:

```
\""\
```

3. Set x equal to 2 and y equal to 6. Write code that prints out the following quantity (not string): x raised to the yth power minus the product of x and y.
4. Determine the output:

```
number = -1.5
if (number > 0):
    print("Positive")
elif (number < 0):
    print("Negative")
else:
    print("Zero")
```

5. Determine the output:

```
grade = 8
if (9 <= grade <= 12):
    print("High school")
elif (6 <= grade <= 8):
    print("Middle school")
elif (1 <= grade <= 5):
    print("Elementary school")
else:
    print("Not a valid grade level")
```

6. Write code that takes in a user-inputted integer number and prints out "This integer is equal to 10" if the integer is equal to 10 or "This integer is not equal to 10" if the integer is not equal to 10.
7. Write code that takes in two **real numbers**, storing them each in a variable. Then write code that prints out the quantity that results from multiplying those two numbers. (Hint: real numbers do not have to be integers, so be wary of using int(); you may want to use a different function to cast the inputted string)
8. Determine the output:

```
x = 7
#x = 8
x = 9
print(x)
```

9. Write code that takes in an integer number and prints "even" if the number is even and "odd" otherwise.
10. Determine the output:

```
num1 = 5
num2 = 6
print(f"{num1} + {num2} = {num1 + num2}")
```

12 Solutions to Practice Problems

1. print("This is a practice problem.")
2. print("\\\"\"\\")
3. See code below.

```
x = 2
y = 6
print(x ** y - x * y)
```

4. Negative
5. Middle school
6. See code below:

```
number = int(input("Please type in an integer number: "))
if number == 10:
    print("This integer is equal to 10")
else:
    print("This integer is not equal to 10")
```

7. See code below:

```
num1 = float(input("Please type in a real number: "))
num2 = float(input("Please type in another real number: "))
print(num1 * num2)
```

8. 9
9. See code below:

```
num = int(input("Please input an integer number: "))
if num % 2 == 0:
    print("even")
else:
    print("odd")
```

10. $5 + 6 = 11$

13 Courses

1. Intro to Computer Science - simple class
2. AP Computer Science A (APCSA) - I strongly recommend this class! This class uses the Java programming language, which is a bit more difficult than Python, but once you know Python, it's easy to switch to Java.
3. AP Computer Science Principles (APCSP) - easier than APCSA. Teachers can choose what coding language they want to teach the class with. You will have to create a project at the end of the year as part of the AP Exam.
4. PLTW Computer Science Principles
5. PLTW Digital Electronics
6. PLTW Computer Integrated Manufacturing
7. Web Design
8. Cybersecurity

Remark: I'm not the best source of information for APCSP, since I didn't take that class, but I would consult with your school teachers. Also, AP exams are classes that you take to get college credit. They're generally harder than regular level classes.

14 Resources

1. MOOC Fi. These are free coding courses created by the University of Helsinki. The courses teach many different coding languages. I took the Python course to learn Python, and it is great! Even though the University of Helsinki is in the country Finland, most courses are in English and you won't need to learn Finnish, of course. I would strongly recommend trying out MOOC Fi if you're interested in diving deeper into coding. https://www.mooc.fi/en/
2. Harvard CS50 Course - introduces a lot of different types of coding languages and applications. This is good for getting a broad overview of coding.
3. Code.org
4. Youtube
5. Udemy
6. Codecademy

7. Khan Academy
8. Coursera
9. EdX
10. MIT OpenCourseWare