Background Information

What is programming?

Programming means writing code (which is just fancy text that does something useful), then starting the code and hoping it works. Often times, it won’t work (especially on first try), and programmers need to check what the error is and try to fix it. Python is a very popular programming language. Here is an example of some Python code.

```
print("Hello world!")
```

When this one line of code is started (or “run”), all it does is show the text “Hello world!”.

What is a “variable” in programming?

Think of a variable as a box or a storage space where you can keep information. You give this box a specific name so you can easily find and use the information later.

For example, imagine you have a box named "shoes" where you keep your shoes. In programming, if you had information like the number "5", you might store it in a "box" named "age" to remember someone's age.

Types of variables in Python

There are many types of information you can store in Python, but for now, let's focus on three main types.

1. Strings (str): This is just text. It can be a single letter, a word, a sentence, or even longer. Strings are put between quotes.

   ```
   name = "Donny"
   ```

2. Dictionaries (Dict): Think of this as a special kind of box with compartments separated by commas. Each compartment has a label (called a "key") and information inside it (called a "value"). The value can be of any type: number, string, and even dictionary!

   ```
   my_pet = { "age": 4, "name": "Buddy", "info": { "weight": 284, "unit": "kg" } }
   ```

   In this case, I have a “my_pet” variable, with the “age” being 4, the “name” being “Buddy”, and the “info” being another dictionary. To access a dictionary’s value in

Python, we use square brackets, with the key name inside the brackets, like so:

```
print(my_pet["name"])
```

This will print “Buddy”.

3. Lists (List): These hold a list of values. Values can be of any type: string, dictionary, etc.

```
pizza_toppings = ["Pineapple", "Banana", "Wasabi", "Eggs"]

my_pets = [{ "age": 4, "name": "Buddy" }, { "age": 2, "name": "Daisy" }]
```

To access a dictionary’s value in Python, we use square brackets, with the numerical
order of the value we want to access inside the brackets (starting from 0), like so:

```
print(pizza_toppings[0])
```

This will print “Pineapple”

You can also add elements to an array by using the *.append()* function. For example:

```
# Initialize an array of strings
pizza_toppings = ["Pineapple", "Banana", "Wasabi", "Eggs"]

# Add a string to the array
pizza_toppings.append("Sour Patch Kids")

# Output the array after the new item is added
print(pizza_toppings)
```

This will print `["Pineapple", "Banana", "Wasabi", "Eggs", "Sour Patch Kids"]`

What is an “if statement” in Python?

An “if statement” in Python allows us to do something conditionally. For example, we could print the string “Could I get extra Wasabi on my pizza?”, but only if a number is equal to 4.

```
num = 4

if num == 4:
    print("Could I get extra Wasabi on my pizza?")
```

Since num is in fact equal to 4, it will output the provided text.

What are operators in Python?

In Python, comparison operators are used to compare two values. They return either **True** or **False** depending on whether the comparison is satisfied. Here are a few primary ones:

1. ==: **Equal to**
   Checks if the values of two operands are equal.
   Example: **x == y** returns **True** if **x** is equal to **y**.

2. **!=**: **Not equal to**
   Checks if the values of two operands are not equal.
   Example: **x != y** returns **True** if **x** is not equal to **y**.

3. <: **Less than**
   Checks if the value of the left operand is less than the value of the right operand.
   Example: **x < y** returns **True** if **x** is less than **y**.

4. >: **Greater than**
   Checks if the value of the left operand is greater than the value of the right operand.
   Example: **x > y** returns **True** if **x** is greater than **y**.

5. <=: **Less than or equal to**
   Checks if the value on the left is less than or equal to the value on the right.
   Example: **x <= y** returns **True** if **x** is less than or equal to **y**.

6. >=: **Greater than or equal to**
   Checks if the value on the left is greater than or equal to the value on the right.
   Example: **x >= y** returns **True** if **x** is greater than or equal to **y**.

These comparison operators are fundamental for building **if** statements mentioned earlier.

What is a loop in Python?

A loop in Python allows us to perform the same operation many times. We can loop over a list in Python by using the "for … in …" syntax. For example, if I wanted to take all non-negative numbers from one array and put them in another array, I could do the following.

```
nums = [1, -4, 2, 3, -8]
```

```
positive_nums = []

for num in nums:
    if num > 0:
        positive_nums.append(num)

print(positive_nums)
```

This will print [1, 2, 3]

What is a function in Python?

You can think of a function like a piece of code that's bundled under a name. It's much like a variable, except instead of holding information, it holds actual code. A function can have inputs (called "parameters"), and a single output (called a "return value").

For example, you could write a function called "add", which takes two numbers as inputs, and outputs the sum of those two numbers. In python, functions are created by using the "def" keyword, a name for the function, then a colon (:), and the function outputs (or "returns") by using the "return" keyword.

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

When you create a function, the code inside it doesn't get automatically run, it needs to be called using its name, like the following:

```
sum = add(a=1, b=2)

print(sum)
```

The code above will store the result of "add(1, 2)" into the "sum" variable, then call "print" (which is a function that Python gives us) to show the result when the code is run. Therefore, the code above would print "3".

What are import statements in Python?

In Python, you can think of import statements as a way to borrow tools or pieces of code from other places, so you don't have to write everything from scratch every time. For example, instead of making an add function, we could import it from somewhere else:

```python
from otherfile import add
print(add(1, 2))
```

This would print “3” if we created the “add” function in another file called otherfile.py.

What are “comments” in code?

In programming, comments are lines in your code that the computer ignores when running the program. They're like little notes or explanations you (or other developers) write directly in the code to clarify what's happening, provide context, or explain why certain decisions were made. Think of them as the "behind-the-scenes" messages that help guide anyone reading the code. In Python, lines that start with “#” are comments:

```python
# This is a comment! It won't be run by the computer.
```

In the exercise code, you’ll find many comments to help you complete the tasks.

What is indentation in Python? Why is it important?

In Python, **indentation** means using spaces (usually four) at the beginning of a line to show which lines of code belong together. Think of it like creating levels in a video game: each level goes deeper with more spaces.

1. **It groups code**: Just like paragraphs in an essay, indentation shows which lines of code work together.
2. **Python requires it**: If you don't indent correctly, your code won't run, and you'll get an error.
3. **Makes code easy to read**: Properly spaced code is easier for both you and others to understand.

For example, in the **if** statement earlier, everything running inside the **if** statement if the condition is true has to be indented. This is how Python knows to execute the correct block of code if the condition is true. It’s the same for functions: all the code belonging to a function must be indented after the “def” statement:

```python
def example_function():
   print("This line is indented, so it only runs when the function is called.")
   print("This line is also indented and part of the function.")

print("This line is not indented, so it is not a part of the function.")
```

GPT: Types of Messages
In this lab, we will be creating a chatbot using Python and GPT. You may have already tried ChatGPT, which is a website that allows you to talk to GPT. In this lab, you'll also be talking to GPT, but by writing Python instead. When sending messages to GPT using Python code, there are 3 types of messages you can send and receive:

1. **User messages**

This is the type of message that humans send to GPT. This could be asking a question, asking for information, or anything else sent to GPT by a person. A user message will trigger a response by GPT.

2. **Assistant messages**

This is the type of message that GPT sends back to humans. This could be an answer to a question, information, or anything else GPT sends back to a person.

3. **System messages**

Most non-developers don't know this type of message, but it can play a crucial role in setting the stage for a conversation. A system message is sent to GPT by a human, like a user message. However, unlike a user message, it will not trigger a response by GPT. Instead, system messages are often used to tell GPT what it's role is in a conversation before the conversation begins. It can also give GPT context or instructions for how to respond to user messages.

For Example

System message (Human): You are a tow truck, you start every message with "Vroom!".

User message (Human): What is the capital of France?

Assistant message (GPT): Vroom! The capital of France is Paris.

User message (Human): Thanks!

Assistant message (GPT): Vroom! You're welcome!