Computer Science

Summer Springboard

SUMMER SPRINGBOARD
Look Inward. Go Upward

Day 2: Basics

SUMMER
SPRINGBOARD
Look Inward. Go Upward.

First up: Syntax vs Semantics

- Just like any other language, spoken or otherwise, Python has a set of rules that need to be followed in order to be understood by others
- First, we will talk about Python's syntax (the "grammar" of Python) and then semantics ("definitions")

Syntax

- Syntax refers to the set of rules that dictate how Python code should be structured and written.
- It deals with the proper arrangement of symbols, keywords, and punctuation in the code.
- Syntax errors occur when code violates these rules, making it invalid and preventing the code from being run.

Semantics

- Semantics refers to the meaning and interpretation of Python code.
- It focuses on how statements and expressions behave and what they do when executed.
- Semantics errors (logical errors) occur when code does not produce the expected or desired results, even if the syntax is correct.

Let’s move on

- We will see examples of good / bad syntax and semantics as we go along

Variables

Variables

- The first Python concept we will learn about today is... variables!
- A “variable” is a symbolic name or identifier used to store and manipulate data in Python programs
- They act as containers that hold values such as numbers, text, or objects

This may sound a bit abstract...

- If this definition doesn't make sense to you, just think of a variable as a box that can hold a specific type of information.
    - For example, maybe you have a box that can hold a number with a decimal, another box that can hold sentences, and another box that can hold either "true" or "false"

SUMMER SPRINGBOARD
Look Inward. Go Upward

Anatomy of a Variable

- A variable in Python has a few special parts:
  - Name - this should be lowercase. If there are multiple words in the name, they should be separated by underscores
  - Value - the object that the variable is holding
  - Type - the category of data that the variable falls into
  - Address - a place that your computer stores the variable in its memory

Let’s make a variable!

- To make a variable, first determine what it’s name and initial value will be.
- Then, put the name on the left side of an equal sign and the value on the right side of the equal sign.
  - For example, **my_first_var = 3.14** creates a variable with name **my_first_var** and an initial value of 3.14
    - The computer will remember this as long as the variable is “in scope” (more on this later)

How do we use variables?

- One way that we can use variables is in mathematical or logical operations.
- Last lesson, we covered mathematical operations in Python, so let's add some variables into our mathematical expressions.

```
1  my_first_var = 3.14
2  my_first_var = my_first_var * 2
3  # my_first_var will now contain the value 6.28
4
5  my_second_var = 8
6  my_second_var /= 4
7  # my_second_var will now contain the value 2
```

Note that for a mathematical binary operation *, **some_var = some_var * other_var** is equivalent to **some_var *= other_var**

SUMMER SPRINGBOARD
Look Inward. Go Upward

Data Types in Python

- We’ve mentioned data types, but have not really gotten into what they are yet.
- Data types define the type of data a variable can hold, such as numbers, text, or more complex types.
- Python's dynamic typing means the type is set at runtime (when the program runs the line of code assigning the variable), not in advance.

Common Data Types

- Integer (int): Represents whole numbers without a decimal. Used for counting, indexing, and more. Example: **age = 30**.
- Floating Point (float): Handles numbers with decimals. Useful for precision in calculations. Example: **temperature = 98.6**
- String (str): A sequence of characters, used for storing text. Can include letters, numbers, symbols. Example: **name = "Alice"**
  - Combine strings using + (this is called concatenation)
    - For example, **“Alice” + “Bob”** becomes **“AliceBob”**
- Boolean (bool): Represents truth values. Only two possible values: True or False. Often used in conditionals. Example: **is_valid = True**

- There are many other types, but this is all we will focus on for now.

Type Conversion

- Changing the type of a variable is common in Python.
  - Example: Converting a string to an integer with **int("10")** means that **“10”** becomes **10** (an integer instead of a string)
- Other commonly used type conversion functions are:
  - **float()** - converts a value to a number with a decimal
  - **str()** - converts a value to a string
  - **bool()** - converts a value to a boolean

Tips for Working with Data Types

- Choose the right data type based on what operations or methods will be performed.
- Be aware of type-related errors: trying to add a number and a string, for instance, will result in an error.
- Use built-in functions like **type()** to check a variable's data type if unsure.

Let’s do some practice!

- Open today’s Google Colab notebook and follow the instructions up until the line that says “***PAUSE***”

SUMMER SPRINGBOARD
Look Inward. Go Upward

Booleans

Introduction to Booleans

- A boolean is a data type that can only have one of two values: **“True”** or **“False”**
- The name “boolean” comes from George Boole, who was an English mathematician.

Using Booleans in Python

- Booleans are often used in conditional statements for decision-making
  - For example, **is_adult = age >= 18** will result in True if age is at least 18 and False otherwise

Boolean Expressions

- Comparisons like **==**, **!=**, **<**, and **>** result in boolean values (either True or False)
  - **==** checks if two values are equal to each other
    - Note: This is different from **=** which assigns a value to a variable
  - **!=** checks if two values are NOT equal to each other
  - **<** checks if a value is less than another
  - **>** checks if a value is greater than another

Logical Operators with Booleans

- Logical operators take two boolean arguments and returns either True or False.
- **and**: both conditions must be true
- **or**: at least one condition must be true
- **not**: takes in only one boolean argument and inverts it
    - **not True** results in **False**
    - **not False** results in **True**

And Truth Table

This table shows what the output result of **and** is, depending on the inputs.

Input 1	Input 2	Output
True	True	True
True	False	False
False	True	False
False	False	False

Or Truth Table

This table shows what the output result of **or** is, depending on the inputs.

Input 1	Input 2	Output
True	True	True
True	False	True
False	True	True
False	False	False

Not Truth Table

This table shows what the output result of **and** is, depending on the inputs.

Input	Output
True	False
False	True

Real-Life Examples

- We can think of logical operators and boolean expressions using real-life examples!
- For example: if it is a weekday AND it is not a holiday, you should go to school
- You can have a movie night if you have a movie you want to watch AND it's not too late in the day to start it
- Go grocery shopping if the fridge is empty OR you are out of essentials like milk or bread
- You can sleep in if it's not a weekday OR you're on vacation

- Think of some other examples on your own or with a group!

More Practice!

- Go back to the Google Colab notebook and work through to the next “***PAUSE***

SUMMER SPRINGBOARD
Look Inward. Go Upward

Conditionals

Conditionals

- Conditionals are constructs in programming that allow the execution of certain pieces of code depending on whether a specific condition (or set of conditions) is met.
- In simple terms, they let your program make decisions.
- In English, we see this in sentences like: **If** (some condition) **then** (do this thing) **else** (do this other thing)
    - For example, **If** it is cold outside, **then** wear a jacket. **Else**, don't wear a jacket.

If

- The **if** statement is the most basic form of conditional. It executes a block of code if a condition (boolean expression) is True.
  - Make sure to indent the line of code that you want to be run if the condition is met
  - Put a colon at the end of the line to let the computer know that the block you want to run if the condition is met is coming up next

```
11
12  temperature = 75
13  if temperature > 70:
14      print("It's warm outside.")
```

SUMMER SPRINGBOARD
Look Inward. Go Upward

Elif

- The **elif** statement stands for “else if.” This is used to check if multiple conditions, one after another.
- You can have as many **elif** statements as you want, but there must be an **if** statement before them.
- Put a colon at the end of the line to let the computer know that the block you want to run if the condition is met is coming up next

```
18  if temperature > 80:
19      print("It's hot outside.")
20  elif temperature < 60:
21      print("It's cold outside.")
```

SUMMER SPRINGBOARD
Look Inward. Go Upward

Else

- The **else** statement catches anything which isn't caught by the preceding conditions.
- There is no condition checked in the **else** block.
- You do not need **elif** statements before the **else** block, but you at least need an **if** statement.
- There should be no other conditional statements after **else**.
- Put a colon at the end of the line to let the computer know that the block you want to run if the condition is met is coming up next

```
25  if temperature > 80:
26      print("It's hot outside.")
27  elif temperature < 60:
28      print("It's cold outside.")
29  else:
30      print("It's a pleasant day.")
```

Anatomy of an if block

So, the overall syntax for an if block is as follows:

```
if (condition):
    do_something
elif (other_condition):
    do_something_else
else:
    do_this_instead
```

- Make special note of where the colons are, the indentation, and the order of **if**, **elif**, and **else**.

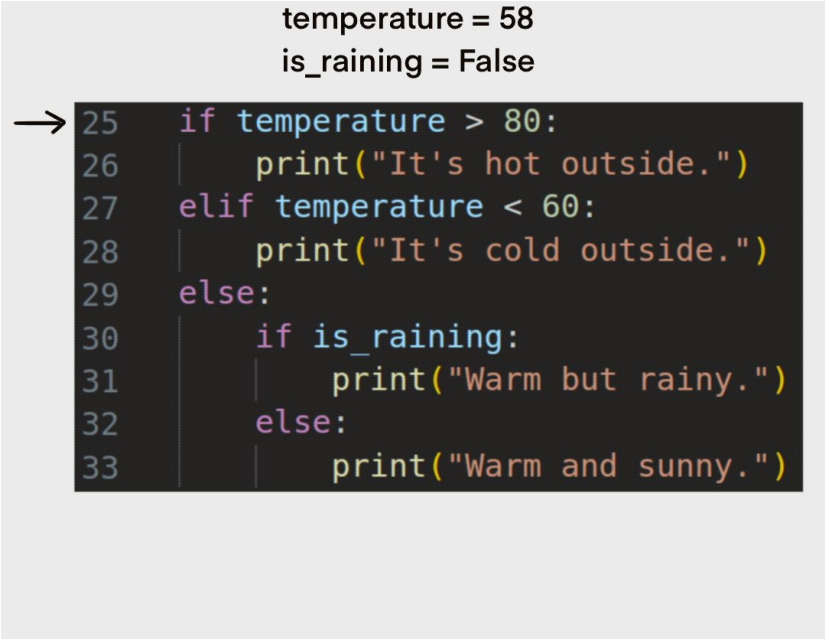
Nested Conditionals

- Conditionals can be nested within each other for more complex logic.

```
37  if temperature > 70:
38      if is_raining:
39          print("Warm but rainy.")
40      else:
41          print("Warm and sunny.")
```

Let’s see an animation!

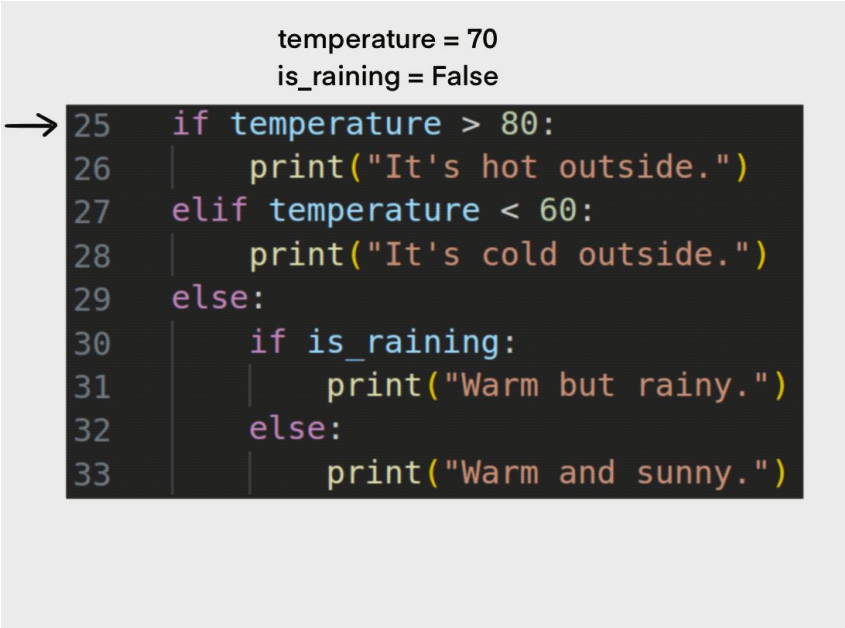
- Check out this animation to see how the computer runs conditionals. The arrow represents the line of code that the computer is currently running.



SUMMER SPRINGBOARD
Look Inward. Go Upward

Animation with Different Values

- Here is an animation showing the same code running, but with different values.

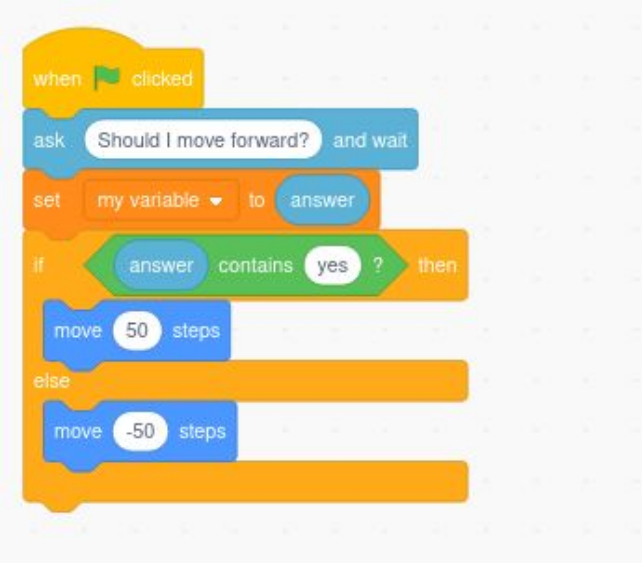


Scratch

- To give us further insight, let's use Scratch to visualize what our code will look like.
- Scratch is a cool tool that allows us to create code using code blocks; you can drag and drop blocks to make different things happen and everything is color-coordinated.
    - Using this website can make coding easier because it gives you a better visual idea of what code is doing.
- Go to: https://scratch.mit.edu/projects/editor/
- Drag and drop blocks so that you have the same "code" as the picture on the next slide

Scratch (Cont'd)

- This gives us a simple visual example of a conditional; Once you press the green flag on the top right, if you type in "yes," our cat character moves forward 50 steps. Otherwise, if you enter anything else, the cat character moves backwards 50 steps.



SUMMER SPRINGBOARD
Look Inward. Go Upward

To the Notebook!

- For the last part of today, let's move onto writing code in our Google Colab notebook. Finish the rest of the exercises in the notebook.