Python Workshop Notes

**1. Introduction to Python**

Python is a general purpose programming language that can be used for a variety of tasks like web development, automation, data science, artificial intelligence, and more. It's very popular because it's designed to be easy to learn, with a simple and readable syntax. This makes Python a great choice for beginners who are just starting their journey in programming.

Python is also an **interpreted language**, meaning it executes the code line by line.

Another key point about Python is its large community and ecosystem. There are countless libraries and frameworks available for Python, which help you perform tasks more easily. For example, web development becomes simpler with Django, and data science becomes more powerful with libraries like Pandas and NumPy. Making it a popular choice among programmers.

**Comparison with Other Languages**

Python is often compared with other programming languages like Java and C++:

* Java has a more complex syntax and is often used in largescale systems like enterprise applications. It is fast and reliable but can be harder for beginners due to the strict rules in its syntax.
* C++ is commonly used for system programming (like operating systems) and video game development. It provides a lot of control over hardware but is more difficult to learn because of its complex structure.
* Python, on the other hand, focuses on being beginner friendly while still being powerful enough for advanced users.

**2. Data Types in Python**

A data type defines what kind of value a variable can store. Python supports various data types, and it’s important to understand these as they are the building blocks of programming.

**Numeric Data Types**

* **Integer**: This represents whole numbers, like 5 or 100. It is used when you need to work with numbers without decimal points.
* **Float**: This represents numbers that have decimal points, such as 3.14 or 0.5. Floats are used when you need more precision.
* **Complex Numbers**: These are numbers that include both real and imaginary parts. They are not commonly used by beginners but are useful in advanced mathematical computations.

**Boolean Data Type**:

* A Boolean can hold only two possible values: : True or False. Booleans are essential in decision making. For example, a program might ask, "Is this number greater than 10?" The answer will be either True or False

**Strings**:

* A string is a collection of characters like letters, numbers, and symbols. Strings are used to store text information.

**3**. **Operators in Python**

Operators are special symbols in Python that perform operations on variables and values. They are essential for carrying out calculations, making decisions, and handling logic.

* **Arithmetic Operators**: These are used to perform basic mathematical calculations, such as adding, subtracting, multiplying, or dividing numbers. They are fundamental when working with numeric data.
* **Logical Operators**: These help you work with multiple conditions at once. For example, you might want to check if both conditions are true (using the and operator), or if at least one of them is true (using the or operator). Logical operators are particularly useful in if statements when making complex decisions. There are 4 types of logical operators in python namely

 **and**: This operator returns **True** if both conditions are true.  
Example: Checking if a number is both greater than 5 **and** less than 10.

 **or**: This operator returns **True** if at least one of the conditions is true.  
Example: Checking if a person is either older than 18 **or** has a permission slip.

 **not**: This operator returns the opposite Boolean value of the condition.  
Example: If a user is **not** an admin, they cannot access certain features.

 **is**: This operator checks if two variables refer to the same object (compares identity).  
Example: Verifying if two variables point to the exact same object in memory.

**4.** **Control Flow: Conditions and Loops**

Control flow in Python refers to the order in which the code is executed. Python provides tools to control the flow of a program, like decision making structures and loops.

* **If Statements**: These are used to run a block of code only if a specific condition is true. If the condition is false, the code block will be skipped. This allows programs to make decisions. For example, you might use an if statement to check if a user has entered the correct password before allowing them to proceed.
* **If else Statements**: Sometimes, you may want to provide an alternative action if the condition in the `if` statement is false. This is where the `else` clause comes in. With `if else` statements, your program can choose between two different blocks of code based on whether a condition is true or false.
* **Elif Statements**: In cases where you have more than two possible outcomes, you can use `elif` (elseif) to check multiple conditions in sequence. The first condition that evaluates to true will trigger its corresponding code block.
* **Nested If-Else Statements**: In some situations, you may need to evaluate multiple layers of conditions. A **nested if-else** structure allows you to place one if or else statement inside another, creating a hierarchy of conditions. This is useful when you need to make further decisions based on a previous result. For example, after verifying if a person is eligible for a discount, you might further check if they qualify for an additional promotion. While nested structures add complexity, they enable more granular decision-making within your program.

**Strings:** A “string” is a [data type in Python](https://www.simplilearn.com/tutorials/python-tutorial/data-types-in-python), composed of a collection of characters which can include letters, numbers, and even special characters.For instance, a person’s name, a sentence, can be stored as a string. Working with strings is common in programs that interact with users, like those that display messages or take input.

* Strings have a lot of built-in functions in Python, making it easy to manipulate text. You can change the case of the text, check if it contains certain characters, or combine multiple strings. Some of the inbuilt functions are
*  **upper()**: This function converts all characters in a string to uppercase. It's useful when you need to standardize text for comparisons or formatting.
*  **lower()**: This function converts all characters in a string to lowercase. It's commonly used when you need to ignore case sensitivity in comparisons or store text in a uniform format.
*  **capitalize()**: This function capitalizes the first letter of a string while converting all other characters to lowercase. It's often used for formatting titles or names.
*  **count()**: This function returns the number of occurrences of a substring within a string. It's helpful when you want to know how many times a particular word or character appears in a text.
*  **find()**: This function searches for a substring within a string and returns the index of the first occurrence. If the substring isn't found, it returns -1. It’s useful for locating specific text within larger strings.
*  **split()**: This function splits a string into a list of substrings based on a specified delimiter (such as a space or comma). It's useful for breaking down text into individual components, such as words or data fields
* Understanding these data types is important because different types of data are handled differently in Python, and using the right type makes your program efficient and accurate.

**5. Data Structures**

Python provides several built-in data structures to store and organize data efficiently.

* **Lists**: A list is a collection of items that can hold multiple values at once. Lists are useful for storing related data, like the names of students in a class. Lists are also flexible because you can add or remove items, and even change the values in a list.
* **Tuples**: A tuple is similar to a list, but once a tuple is created, you cannot change it. This makes tuples useful for data that should not be altered. For example, the dimensions of a rectangle (width, height) might be stored in a tuple since these values are usually fixed.
* **Sets**: A set is an unordered collection of unique items. Unlike lists or tuples, sets do not allow duplicate values. They are useful for storing data where you only care about unique entries. For example, a set could store all unique words from a text.
* **Dictionaries**: A dictionary stores data in keyvalue pairs. Each key is unique, and it maps to a corresponding value. Dictionaries are extremely useful for looking up information quickly. For instance, you can use a dictionary to store a person’s name as a key and their phone number as the value. They also have inbuilt functions like
* **update(**): Merges the contents of another dictionary or iterable of key-value pairs into the dictionary.
* **setdefault(**): Returns the value of a specified key; if the key does not exist, it inserts the key with a specified default value.
* **clear()**: Removes all items from the dictionary, resulting in an empty dictionary.
* **get(**): Retrieves the value for a specified key; if the key is not found, it returns None or a default value if provided.
* **pop()**: Removes and returns the value associated with a specified key; raises a KeyError if the key is not found unless a default value is provided.
* **keys():** Returns a view object of all the keys in the dictionary.
* **values()**: Returns a view object of all the values in the dictionary.
* **items()**: Returns a view object of all key-value pairs in the dictionary as tuples.
* These data structures allow you to organize and store data in an efficient way, and they provide various methods to access and manipulate the data.

**Loops:**

* **For Loops**: These are used to repeat a block of code for each item in a collection (like a list or string). It’s useful when you know in advance how many times you want to repeat the code.
* **While Loops**: These repeat a block of code as long as a specific condition remains true. They are useful when you don’t know in advance how many times the loop should run. The loop continues until the condition becomes false.
* Loops are essential for automating repetitive tasks in your programs. For instance, a loop can go through a list of student names and print them out one by one.

**6. Functions**

A function is a block of reusable code that performs a specific task. Functions help to make your code more organized and manageable by allowing you to group related instructions into a single unit.

* Functions can take parameters, which are inputs that modify the way the function behaves. For example, a function that calculates the area of a rectangle might take the rectangle’s width and height as parameters.
* Functions can also return values, allowing you to get results back after performing calculations or processing data.

Using functions is essential for writing clean and efficient code. By breaking down complex tasks into smaller, reusable functions, you make your programs easier to read, debug, and maintain.

**7. Object Oriented Programming (OOP)**

Python is also an object-oriented programming (OOP) language and everything in python is an object. OOP is a programming paradigm that revolves around the concept of objects, which represent real world entities. Each object is created from a class, which is like a blueprint that defines its properties and behaviour.

* Classes: A class defines the structure of an object. It includes attributes (data) and methods (functions) that define what the object can do. For example, in a program that simulates a library, you could have a class called `Book` that stores information about a book’s title, author, and ISBN number.
* Objects: An object is an instance of a class. You can create many objects from the same class, each with its own set of attributes.
* Inheritance: This is a feature of OOP that allows one class to inherit the properties and methods of another class. It promotes code reusability, as you can create new classes that extend the functionality of existing ones. For example, if you have a `Vehicle` class, you could create a `Car` class that inherits from `Vehicle` but adds more specific attributes, like the number of doors or the fuel type.

**8. Libraries and Modules**

One of Python's strengths is its extensive collection of libraries and modules. These are prewritten pieces of code that you can import into your programs to save time and effort.

* **NumPy**: This is a powerful library used for numerical computing. It makes working with large arrays and matrices much easier and more efficient. NumPy also provides mathematical functions for performing complex calculations, making it an essential tool in data science and machine learning.
* **Vectorization**: This is a technique used in NumPy that allows you to apply operations to entire arrays of data without writing explicit loops. This makes your code not only simpler but also faster, as vectorized operations are optimized for performance.
* **SQLite3**: It is a lightweight, disk-based database engine that doesn’t require a separate server process. It’s part of the Python Standard Library, allowing easy integration of SQL-based data management into Python applications. With SQLite3, you can create, read, update, and delete database records using SQL commands. The library provides a simple interface to interact with SQLite databases through Python's `sqlite3` module, which supports executing SQL queries, managing transactions, and handling database connections. This makes SQLite3 ideal for lightweight, embedded applications or as a temporary database for development and testing.
* **File handling in Python**: It can be efficiently managed using `with open()` and `writelines()`. The `with open()` statement serves as a context manager that opens a file and ensures it is properly closed after the block of code is executed, even if an error occurs, thereby simplifying resource management. Within this context, `writelines()` is used to write a list of strings to a file in one operation. Unlike `write()`, which handles single strings, `writelines()` writes multiple lines or strings from a list to the file sequentially without adding extra newlines. This approach is particularly useful for writing large amounts of text data efficiently.
* **Pandas**: It is a powerful data manipulation library that builds on file handling by providing high-level data structures and functions specifically designed for data analysis. It supports reading from and writing to various file formats, such as CSV, Excel, and SQL databases, using functions like read\_csv(), read\_excel(), to\_csv(), and to\_sql(). Pandas makes it easy to clean, transform, and analyse data using its DataFrame and Series objects, streamlining the data analysis process and enabling sophisticated data operations with minimal code.