* Python can distinguish among data types such as integers, floats, strings, and Booleans.
* Integers are whole numbers that can be positive or negative.
* Floats include integers as well as decimal numbers between the integers.
* You can convert integers to floats using typecasting, but you cannot convert a float to an integer.
* You can convert integers and floats to strings.
* You can convert an integer or float value to True (1) or False (0).
* Expressions in Python are a combination of values and operations used to produce a single result.
* Expressions perform mathematical operations such as addition, subtraction, multiplication, and so on.
* We use"//" to round off integer divisions, resulting in float values.
* Python follows the order of operations (BODMASS) to perform operations with multiple expressions.
* Variables store and manipulate data, allowing you to access and modify values throughout your code.
* The assignment operator "=" assigns a value to a variable.
* ":" denotes the value of the variable within the code.
* Assigning another value to the same variable overrides the previous value of that variable.
* You can perform mathematical operations on variables using the same or different variables.
* While performing operations with various variables, modifying a value in one variable will lead to changes in the other variables.
* Python string operations involve manipulating text data using tasks such as indexing, concatenation, slicing, and formatting.
* A string is usually written within double quotes or single quotes, including letters, white space, digits, or special characters.
* A string attaches to another variable and is an ordered sequence of characters.
* Characters in a string identify their index numbers, which can be positive or negative.
* We use strings as a sequence to perform sequence operations.
* You can input a stride value to perform slicing while operating on a string.
* Operations like finding the length of the string, combining, concatenating, and replicating, result in a new string.
* You cannot modify an existing string; they are immutable.
* You can perform escape sequences using " " to change the layout of the string.
* In Python, you perform tasks such as searching, modifying, and formatting text data with its pre-built string methods functions.
* You apply a method to a string to change its value, resulting in another string.
* You can perform actions such as changing the case of characters in a string, replacing items in a string, finding items in a string, and so on using pre-built string methods.

Format strings are a way to inject variables into a string in Python. They are used to format strings and produce more human-readable outputs. There are several ways to format strings in Python:

**String interpolation (f-strings)**

Introduced in Python 3.6, f-strings are a new way to format strings in Python. They are prefixed with 'f' and use curly braces {} to enclose the variables that will be formatted. For example:

1. 1
2. 2
3. 3
4. name = "John"
5. age = 30
6. print(f"My name is {name} and I am {age} years old.")

Copied!

This will output:

1. 1
2. My name is John and I am 30 years old.

Copied!

**str.format()**

This is another way to format strings in Python. It uses curly braces {} as placeholders for variables which are passed as arguments in the format() method. For example:

1. 1
2. 2
3. 3
4. name = "John"
5. age = 50
6. print("My name is {} and I am {} years old.".format(name, age))

Copied!

This will output:

1. 1
2. My name is John and I am 50 years old.

Copied!

**% Operator**

This is one of the oldest ways to format strings in Python. It uses the % operator to replace variables in the string. For example:

1. 1
2. 2
3. 3
4. name = "Johnathan"
5. age = 30
6. print("My name is %s and I am %d years old." % (name, age))

Copied!

This will output:

1. 1
2. My name is Johnathan and I am 30 years old.

Copied!

Each of these methods has its own advantages and use cases. However, f-strings are generally considered the most modern and preferred way to format strings in Python due to their readability and performance.

**Additional capabilities**

F-strings are also able to evaluate expressions inside the curly braces, which can be very handy. For example:

1. 1
2. 2
3. 3
4. x = 10
5. y = 20
6. print(f"The sum of x and y is {x+y}.")

Copied!

This will output:

1. 1
2. The sum of x and y is 30.

Copied!

**Raw String (r’’)**

In Python, raw strings are a powerful tool for handling textual data, especially when dealing with escape characters. By prefixing a string literal with the letter ‘r’, Python treats the string as raw, meaning it interprets backslashes as literal characters rather than escape sequences.

Consider the following examples of regular string and raw string:

**Regular string:**

1. 1
2. 2
3. regular\_string = "C:\new\_folder\file.txt"
4. print("Regular String:", regular\_string)

Copied!

This will output:

1. 1
2. 2
3. Regular String: C:
4. ew\_folderile.txt

Copied!

In the regular string regular\_string variable, the backslashes (\n) are interpreted as escape sequences. Therefore, \n represents a newline character, which would lead to an incorrect file path representation.

**Raw string:**

1. 1
2. 2
3. raw\_string = r"C:\new\_folder\file.txt"
4. print("Raw String:", raw\_string)

Copied!

This will output:

1. 1
2. Raw String: C:\new\_folder\file.txt

Copied!

However, in the raw string raw\_string, the backslashes are treated as literal characters. This means that \n is not interpreted as a newline character, but rather as two separate characters, ‘’ and ‘n’. Consequently, the file path is represented exactly as it appears.