Chapter 1 : Introduction to Python

Aviral Janveja

1 Introduction

Programming is the art of instructing computers to perform specific tasks. At its core, an **algorithm** serves as a systematic set of steps or instructions designed to solve a computational problem or accomplish a specific calculation. So, while an algorithm provides the strategy, a program brings that strategy to life in a language that computers can comprehend.

In Python, a program is typically composed of a sequence of statements, which can include definitions, such as creating variables or functions and commands, like print statements and other executable instructions.

2 Print

The print statement is a fundamental tool for displaying information and results in programs during development and debugging. It is particularly useful for quickly checking the values of variables or printing messages to understand how exactly your code is executing. The basic print statement looks as follows :

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

You can check the output generated by the above command and some additional features in the following program : print.py uploaded to GitHub.

3 Variables

Variable is a name associated to a value. You can think of it as a label assigned to a piece of data. Variable names are associated to data values through the **assignment** operator "=". For example :

```
name = "Avi"
print("My name is " + name + ".")
```

It is important to chose variable names appropriately to enhance the readability of the program. Variable names can contain uppercase letters, lowercase letters and digits. However, they cannot start with a digit. The special character underscore is allowed as well. Python variable names are case sensitive, so Avi and avi are separate names.

3.1 Comments

Another good way to enhance readability of the code is to add comments explaining what your code is doing. Text following the # symbol is considered as comment, and therefore not **interpreted** by python.

Finally, there are a small number of reserved keywords with built-in meanings that cannot be used as variable names. For example $class$, $print$, if, $else$ and so on.

4 Data Types

Variables in Python are **dynamically typed**, meaning you don't need to declare their type explicitly; the interpreter infers the data type based on the value assigned. The following are some of the basic data types in Python :

- **Integer (int)** : Represents whole numbers, positive and negative.
- **Float (float)** : Represents numbers with decimal points.
- **String (str)** : Represents text, enclosed in single or double quotes.
- **Boolean (bool)** : Represents boolean values True and False.
- **None (None)** : Represents the absence of a value or a null value.

The built in function **type()** can be used to find out the type of any data value. For example *type(3)* and *type(3.0)* will output the result *int* and *float* respectively.

Concepts in this section are illustrated through the variable.py program on GitHub.

5 Operators

Data, variables and operators can be combined to form expressions. Each expression evaluates to a value of some type. For example, $3+2$ denotes the value of type *int*, whereas $3.0 + 2.0$ denotes the value of type *float*.

We have already come across the assignment operator used to define variables, above. The following is a summary of operators in Python.

5.1 Arithmetic Operators

Arithmetic operators are used to perform mathematical operations. These include:

- **Addition** $(+)$ and **Subtraction** $(-)$
- **Multiplication** $(*)$ and **Division** $(/)$
- **Modulus** $(\%)$ and **Exponentiation** $(**)$

The arithmetic operators have the usual mathematical precedence. The order of evaluation can be changed by using parentheses to group sub-expressions. For instance, $(x + y) * 2$ first adds x and y and then multiplies the result by 2.

5.2 Logical Operators

Logical operators are used to combine or negate Boolean values. These include:

- **And** (and)
- **Or** (or)
- **Not** (not)

5.3 Comparison Operators

Comparison operators are used to compare values and return a Boolean result. These include:

- **Equal to** ($==$)
- **Not equal to** ($!=$)
- **Greater than** ($>$)
- **Less than** ($<$)

Concepts in this section are illustrated through the operators_expressions.py program on GitHub.

6 Strings

Let us look at Strings in further detail. It is a data type which represents a sequence of characters, enclosed in single or double quotes. Unlike many programming languages, Python has no type corresponding to a character. Instead, it uses strings of length 1.

For example, the literal '123' denotes a string of characters, not the number one hundred twenty-three. Here are a few ways operators can work with Strings :

- **Concatenation** : Two or more strings can be linked together in a chain using the $+$ operator. The $*$ operator can be used to duplicate a string multiple times. For example, $2 *$ 'Avi' will have the value 'AviAvi'.
- **Indexing** : You can access individual characters in a string using indexing. For example, 'abc'[0] will display the string 'a'.
- **Length** : The len() function can be used to get the length (number of characters) in a string.

These are just a few basic ways of working with strings in Python. Strings are versatile and play a crucial role in text processing, data manipulation, and various other aspects of programming. String Concepts are illustrated through the string.py program on GitHub.

7 User Input

In Python, you can use the input() function to take user input from the keyboard. The input() function reads a line from the user and returns it as a string. For example :

user_name = input("Enter your name: ")

By default, input() returns the user input as a string. If you need to work with numerical values, you should convert the input to the appropriate data type. For example :

num = int(input("Type any integer : "))

The above is an example of **type casting**, also called type conversion and is used often in Python code. We use the name of a type to convert values to that type. So, for example, the value of int('3') * 4 is 12. And when a float is converted to an int, the number is truncated (not rounded). For example, the value of int(3.9) will be 3.

You can check the following program : user_input.py uploaded to GitHub for an example of the above concepts.