PYTHON PROGRAMMING

MODULE 1

Introduction to Python - The IDLE Python Development Environment - The Python Standard Library - Literals - Numeric Literals - String Literals - Control Characters - String Formatting - Implicit and Explicit Line Joining Variables and Identifiers - Variable Assignment and Keyboard Input- Identifier-Keywords and Other Predefined Identifiers in Python – Operators - Various Operators - Relational Operators-Membership Operators – Boolean Operators - Expression and Data Types - Operator Precedence and Boolean Expressions - Operator Associativity - Mixed-Type Expression

Who Created Python?

Python was created by Guido van Rossum, and first released on February 20, 1991. The name of the Python programming language comes from an old BBC television comedy sketch series called Monty Python's Flying Circus.

- Web development (server-side),
- Software development,
- Mathematics,
- System scripting.

What can Python do?

- Python can be used on a server to create web applications.
- Python can be used alongside software to create workflows.
- Python can connect to database systems. It can also read and modify files.
- Python can be used to handle big data and perform complex mathematics.
- Python can be used for rapid prototyping, or for production-ready software development.

Why Python?

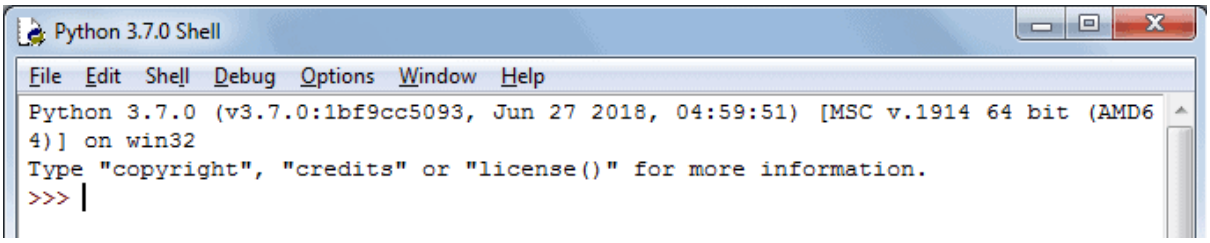
- Python works on different platforms (Windows, Mac, Linux, Raspberry Pi, etc).
- Python has a simple syntax similar to the English language.
- Python has syntax that allows developers to write programs with fewer lines than some other programming languages.
- Python runs on an interpreter system, meaning that code can be executed as soon as it is written. This means that prototyping can be very quick.
- Python can be treated in a procedural way, an object-oriented way or a functional way.

Python - IDLE

IDLE (Integrated Development and Learning Environment) is an integrated development environment (IDE) for Python. The Python installer for Windows contains the IDLE module by default.

IDLE can be used to execute a single statement just like Python Shell and also to create, modify, and execute Python scripts. IDLE provides a fully-featured text editor to create Python script that includes features like syntax highlighting, autocompletion, and smart indent. It also has a debugger with stepping and breakpoints features.

This will open IDLE, where you can write and execute the Python scripts, as shown below.



The editor allows us to save our programs and conveniently make changes to them later. The editor understands the syntax of the Python language and uses different colors to highlight the various components that comprise a program. Much of the work of program development occurs in the editor.

The Python Standard Library

While The Python Language Reference describes the exact syntax and semantics of the Python language, this library reference manual describes the standard library that is distributed with Python. It also describes some of the optional components that are commonly included in Python distributions.

Here are some of the modules available in the Python Standard Library:

os - provides a way to interact with the underlying operating system, such as file and directory operations, process management, and environment variables.

sys - provides access to some variables used or maintained by the interpreter and to functions that interact with the interpreter, like exiting the program.

math - contains mathematical functions for use in scientific and engineering applications.

random - provides functions for generating pseudo-random numbers, shuffling sequences randomly, and selecting random items.

datetime - provides classes for working with dates and times.

json - provides functions for encoding and decoding JSON data.

re - provides regular expression matching operations.

socket - provides low-level networking interfaces for creating client and server sockets.

http - provides classes for working with HTTP servers and clients.

Literals

In Python, literals are the representation of values in source code. They are the most basic and fundamental elements of any program, and they include values like strings, numbers, booleans, and more.

Here are the different types of literals in Python:

Numeric literals: These include integers, floating-point numbers, and complex numbers. For example, 42, 3.14, and 1+2j.

String literals: These are sequences of characters enclosed in either single or double quotes. For example, "hello", 'world', and "Python's string literals".

Boolean literals: These are either True or False.

None literal: This is a special value representing "nothing". It is denoted by the keyword None.

Sequence literals: These include lists, tuples, and ranges. For example, [1, 2, 3], (1, 2, 3), and range(0, 10).

Set literals: These are enclosed in curly braces {} and contain unique elements separated by commas. For example, {1, 2, 3}.

Dictionary literals: These are enclosed in curly braces {} and contain key-value pairs separated by colons. For example, {'name': 'John', 'age': 30}.

Byte and byte array literals: These are used to represent bytes and byte arrays. For example, b'hello' and bytearray([0, 1, 2, 3]).

Numeric Literals

Numeric literals can be of three types: integers, floating-point numbers, and complex numbers.

Integers: Integers are whole numbers, such as 42 or -10. Integers can be written in decimal, binary, octal, or hexadecimal format. Here are some examples:

Decimal integer: 42

Binary integer: 0b101010

Octal integer: 0o52

Hexadecimal integer: 0x2a

Floating-point numbers: Floating-point numbers are numbers with a fractional part, such as 3.14 or -0.1. They are represented using a decimal point. Here are some examples:

Positive floating-point number: 3.14

Negative floating-point number: -0.1

Scientific notation: 2.5e-3

Complex numbers: Complex numbers are numbers with a real and imaginary part, such as 2+3j or -1-4j. They are represented using the j suffix. Here are some examples:

Positive complex number: 2+3j

Negative complex number: -1-4j

String Literals

String literals are enclosed in either single quotes ' or double quotes "

Single-line strings: These are simple strings that can be written on a single line. For example:

Single quotes: 'hello world'

Double quotes: "hello world"

Multi-line strings: These are strings that span multiple lines. In Python, you can create multi-line strings using triple quotes (''' or """). Here are some examples:

Single quotes: '''hello world'''

Double quotes: """hello world"""

Raw strings: These are strings that treat backslashes (\) as literal characters instead of escape characters. Raw strings are created by prefixing a string literal with r or R. For example:

Raw string with single quotes: r'hello\nworld'

Raw string with double quotes: R"hello\tworld"

String literals can be used in various ways in Python, such as concatenation, slicing, formatting, and more. It's important to keep in mind that strings are immutable in Python, meaning that once a string is created, it cannot be modified. Instead, operations on strings create new strings with the desired modifications.

Control Characters

Newline (\n): This control character is used to start a new line in a string.

Example: "Hello\nworld"

Ouput :

Hello

World

Tab (\t): This control character is used to insert a horizontal tab in a string.

Example: "Name:\tMurugan"

Output:

Name: Murugan

Backslash (\\): This control character is used to represent a literal backslash in a string.

Example: "C:\\Windows\\System32"

Output:

C:\Windows\System32

Single quote (\'): This control character is used to represent a literal single quote in a string enclosed in single quotes.

Example: ' He said, \'Hello.\' '

He said, 'Hello.'

Double quote (\"): This control character is used to represent a literal double quote in a string enclosed in double quotes.

Example: "She said, \"Goodbye.\""

Output:

She said, "Goodbye."

Carriage return (\r): This control character is used to move the cursor to the beginning of the current line in a string.

Example: "Hello\rworld"

Output:

World

String Formatting (Joining Variables with string)

Use the format() method to insert numbers into strings:

Example

```
age = 17
txt = "My name is David, and I am {} years old "
print(txt.format(age))
```

Output

My name is David, and I am 17 years old

Example 2

```
quantity = 3
itemno = 567
price = 49.95
myorder = "I want to pay {2} dollars for {0} pieces of item {1}."
print(myorder.format(quantity, itemno, price))
```

Output

```
I want to pay 49.95 dollars for 3 pieces of item 567
```

Implicit line joining

Expressions in parentheses, square brackets or curly braces can be split over more than one physical line without using backslashes.

For example:

```
Weekdays=[' Monday ', 'Tuesday ', 'Wednesday ', 'Thursday ', 'Friday ', 'Saturday ', 'Sunday ']
```

Explicit line joining

Two or more physical lines may be joined into logical lines using backslash characters (\), as follows: when a physical line ends in a backslash that is not part of a string literal or comment, it is joined with the following forming a single logical line, deleting the backslash and the following end-of-line character. For example:

```
if  year < 2001 and month <= 12 \
  and day <= 31
     return 1
```

Variable Assignment

A variable is created the moment you first assign a value to it. Variables do not need to be declared with any particular type, and can even change type after they have been set.

Example

```
x = 5
y = "Senthil"
print(x)
print(y)
```

Output

```
5
Senthil
```

Variable Assignment and Keyboard Input:

- The value that is assigned to a given variable does not have to be specified in the program, as demonstrated in previous examples.
- The value can come from the user by use of the input function.

```
name = input('What is your first name?')
```

What is your first name? John

- In this case, the variable name is assigned the string 'John'. If the user hit return without entering any value, name would be assigned to the empty string ('').
- All input is returned by the input function as a string type.
- For the input of numeric values, the response must be converted to the appropriate type.
- Python provides built-in type conversion functions int () and float() for this purpose, as shown below for a gpa calculation program,

```
line = input('How many credits do you have?')
num_credits = int(line)
line = input('What is your grade point average?')
print(float(line))
```

- All input is returned by the input function as a string type.
- Built-in functions int () and float() can be used to convert a string to a numeric type.

Identifier

Identifiers are the name given to variables, classes, methods, etc. For example,

```
lang = 'Python'
```

Here, language is a variable (an identifier) which holds the value 'Python'.

We cannot use keywords as variable names as they are reserved names that are built-in to Python. For example,

```
continue = 'Python'
```

The above code is wrong because we have used continue as a variable name. To learn more about variables, visit Python Variables.

Rules for Naming an Identifier

- Identifiers cannot be a keyword.
- Identifiers are case-sensitive.
- It can have a sequence of letters and digits. However, it must begin with a letter or _. The first letter of an identifier cannot be a digit.
- It's a convention to start an identifier with a letter rather _.
- Whitespaces are not allowed.
- We cannot use special symbols like !, @, #, $, and so on.

Valid and Invalid Identifiers in Python

Valid Identifiers	Invalid Identifiers
score	@core
return_value	return
highest_score	highest score
name1	1name

Python Operators

Python divides the operators in the following groups:

- Arithmetic operators
- Assignment operators
- Comparison operators
- Logical operators
- Identity operators
- Membership operators
- Bitwise operators

Arithmetic Operators

Arithmetic operators are used with numeric values to perform common mathematical operations:

Operator	Name	Example
+	Addition	x + y
-	Subtraction	x - y
*	Multiplication	x * y
/	Division	x / y
%	Modulus	x % y
**	Exponentiation	x ** y
//	Floor division	x // y

Assignment Operators

Assignment operators are used to assign values to variables:

Operator	Example	Same As
=	x = 5	x = 5
+=	x += 3	x = x + 3
-=	x -= 3	x = x - 3
*=	x *= 3	x = x * 3
/=	x /= 3	x = x / 3
%=	x %= 3	x = x % 3
//=	x //= 3	x = x // 3
**=	x **= 3	x = x ** 3
&=	x &= 3	x = x & 3
|=	x |= 3	x = x | 3
^=	x ^= 3	x = x ^ 3
>>=	x >>= 3	x = x >> 3
<<=	x <<= 3	x = x << 3

Python Comparison Operators

Comparison operators are used to compare two values:

Operator	Name	Example	Try it
==	Equal	x == y	
!=	Not equal	x != y	
>	Greater than	x > y	
<	Less than	x < y	
>=	Greater than or equal to	x >= y	
<=	Less than or equal to	x <= y	

Python Logical Operators

Logical operators are used to combine conditional statements:

Operator	Description	Example
and	Returns True if both statements are true	x < 5 and x < 10
or	Returns True if one of the statements is true	x < 5 or x < 4
not	Reverse the result, returns False if the result is true	not(x < 5 and x < 10)

Identity Operators

Identity operators are used to compare the objects, not if they are equal, but if they are actually the same object, with the same memory location:

Operator	Description	Example
is	Returns True if both variables are the same object	x is y
is not	Returns True if both variables are not the same object	x is not y

Python Bitwise Operators

Bitwise operators are used to compare (binary) numbers:

Operator	Name	Description	Example
&	AND	Sets each bit to 1 if both bits are 1	x & y
|	OR	Sets each bit to 1 if one of two bits is 1	x | y
^	XOR	Sets each bit to 1 if only one of two bits is 1	x ^ y
~	NOT	Inverts all the bits	~x
<<	Zero fillleft shift,	Shift left by pushing zeros in from the right and let the leftmost bits fall off	x << 2
>>	Signed right shift	Shift right by pushing copies of the leftmost bit in from the left, and let the rightmost bits fall off	x >> 2

Operator Precedence

Operator precedence describes the order in which operations are performed.

Example

Parentheses has the highest precedence, meaning that expressions inside parentheses must be evaluated first:

```
print((6 + 3) - (6 + 3))
```

Example

Multiplication * has higher precedence than addition +, and therefor multiplications are evaluated before additions:

```
print(100 + 5 * 3)
```

The precedence order is described in the table below, starting with the highest precedence at the top:

Operator	Description
()	Parentheses
**	Exponentiation
+x -x ~x	Unary plus, unary minus, and bitwise NOT
* / // %	Multiplication, division, floor division, and modulus
+ -	Addition and subtraction
<< >>	Bitwise left and right shifts
&	Bitwise AND
^	Bitwise XOR
|	Bitwise OR
== != > >= < <=	is, is not, in, not in, are Comparisons, identity, and membership operators
not	Logical NOT
and	AND
or	OR

If two operators have the same precedence, the expression is evaluated from left to right.

Example

Addition + and subtraction - has the same precedence, and therefor we evaluate the expression from left to right:

```
print(5 + 4 - 7 + 3)
```

Boolean Type

The Python Boolean type has only two possible values:

True

False

No other value will have bool as its type. You can check the type of True and False with the built-in type():

```
>>> type(False)
<class 'bool'>
>>> type(True)
<class 'bool'>
```

Associativity of Python Operators

Associativity is the order in which an expression is evaluated that has multiple operators of the same precedence. Almost all the operators have left-to-right associativity.

For example, multiplication and floor division have the same precedence. Hence, if both of them are present in an expression, the left one is evaluated first.

Left-right associatively

```
        print(5 * 2 / 3)
# Output: 3.3333
```