**PYTHON BASICS**

Python is most widely used **general purpose high level programming language** like Java, C, C++ etc. Python was developed by **Guido van Rossum** in the late eighties and early nineties at the National Research Institute for Mathematics and Computer Science in the Netherlands. Python is derived from many other languages, including **ABC, Modula-3, C, C++, Algol-68, SmallTalk, and Unix shell** and other scripting languages. Rossum was a fan of a comedy series from late seventies. Python is named after a TV Comedy Show called ‘**Monty Python’s Flying Circus’** and not after Python-the snake**. Python 3.11.4** is the latest version of Python.

**FEATURES OF PYTHON**

* **General purpose programming language** - As a general-purpose programming language python is designed to be used for widest variety of application domains (a general-purpose language). ...It can be used for developing both desktop and web applications, complex scientific and numeric applications, data analysis and visualization. Conversely, a domain-specific programming language is one designed to be used within a specific application domain. For eg: COBOL (Business applications), FORTRAN(Complex mathematical computations).
* **High level language** - High-level language is any programming language that enables development of a program in a much more user-friendly programming context and is generally independent of the computer's hardware architecture. Like JAVA, C ,C++ Python is a Highlevel programming language.
* **Interpreted** - Python runs on an interpreter system, meaning that code can be executed as soon as it is written. You do not need to compile your program before executing it.
* **Interactive** - We can actually sit at a Python prompt and interact with the interpreter directly to write your programs and It allows interactive testing and debugging of snippets of code.
* **Portable** - Python can run on a wide variety of hardware platforms like Windows, Linux, MAC OS and has the same interface on all platforms. You can move Python programs from one platform to another, and run it without any changes.
* **Multiple Programming Paradigms**- Python also supports several programming paradigms. It supports object oriented and Functional programming. Python can be treated in a procedural way, an object-oriented way or a functional way.
* **Easy to Learn** - Python has a simple syntax similar to the English language. This makes python easier to learn. Python has syntax that allows developers to write programs with fewer lines than some other programming languages. Most of other high level languages like JAVA, C, C++ has so many syntactical constructs like Punctuation marks, semicolon, braces etc to indicate the ending of a statement or to identify block of code. But in Python, It has fewer syntactical constructions than other languages.

For eg:

In **JAVA**

Suppose we need to assign a name **“BOB”** to a variable **name** .In JAVA we have to first specify the type of the variable as **String**. After that we assign value **BOB** to variable **name** and put a semicolon at the end to indicate ending of a line.

**String name=”BOB”;**

In order to print it out we use **System.Out.Println(“name”);** and put a semi colon at the end.

But

In **Python**

We write **name=”BOB”**

In order to print it out we use simply **print(name**)

We do not need to specify the type and put semicolon to indicate the ending of a line. This syntax makes Python code more easier to read and Learn.

* **Dynamic Typed Language** – As specified above, in python we don’t have to specify the type when declaring a variable. It skip the headache of type casting and declaring types when declaring a variable.

**In JAVA,**

**int x=1;**

**x=(int)x/2**;

In order to store integer values to a variable **x** first we have to declare its type as **int**. it means that x always store integer values .In the first line we assign value 1 to integer variable x.

If we take **x=x/2**; the value of x becomes ½. x can never equal 0.5. So first we have to cast the result into type integer using (int) function. Now the result becomes 0.

In python,

x=1

x=x/2

In this case, Python itself take care of type management we don’t need to worry about it.

If we write x=1 at this point type of x is int because we assign integer value to x. if we write x=x/2 now x equals 0.5 and the type of x at this point is float. We don’t need to explicitly type cast the result into float. According to the type of values we store to a variable its type is decided by the interpreter at runtime. We don’t need to worry about it.

* **Databases** - Python provides interfaces to all major commercial databases like POSTGRES, MY SQL, SQLITE.
* **GUI Programming**- Python supports GUI applications. Python provides Tk GUI library to develop user interface in python based application.
* **Free and Open-Source**-Python is developed under an OSI-approved open source license. Hence, it is completely free to use, even for commercial purposes. It doesn't cost anything to download Python or to include it in your application. It can also be freely modified and re-distributed. Python can be downloaded from the official Python website.
* **Robust Standard Library**-Python has an extensive standard library available for anyone to use. This means that programmers don’t have to write their code for every single thing unlike other programming languages. There are libraries for image manipulation, databases, unit-testing, expressions and a lot of other functionalities.
* **Large Community Support-**Python was founded around 30 years ago and so it has a vast community of efficient developers. These developers are constantly helping out the beginners through their constant support and in-depth journals. There is plenty of documentation and guides that the newbie programmers could learn and enhance their Python.

**Setting up of Environment**

In order to use python first it must be installed on our computer, Follow these steps

1).Go to the Python website [www.python.org](http://www.python.org/) and click the Download menu choice.

2). Next , Download the Python 3.10 or Python 3.9 installer.

3).When the download is completed, double-click the file and follow the instructions to install it.

**First Python Program**

Let us execute the programs in different modes of programming.

* **Interactive Mode Programming**.

Python provides Interactive Shell to execute code immediately and produce output instantly. To test a short amount of code in python sometimes it is quickest and easiest not to write the code in a file. This is made possible because Python can be run as a command line itself.

Type the following text at the Python prompt and press Enter –

>>> **print ("Hello, Python!")**

This produces the following result –

>>>**Hello, Python!**

* **Script Mode Programming**

Using **Script Mode**, we can write our Python code in a separate file of any editor in our Operating System. Let us write a simple Python program in a script. Python files have the extension .**py**. Type the following source code in **a test.py** file –

**print ("Hello, Python!")**

Now open Command prompt and execute it by :

>>> **python test.py**

This produces the following result –

>>>**Hello, Python!**

* **USING IDLE**

When Python is installed, a program called **IDLE** is also installed along with it. It provides graphical user interface to work with Python.

Open IDLE, copy the following code below and press enter.

>>>**print("Hello, World!")**

Or

To create a file in IDLE, go to **File > New Window (Shortcut: Ctrl+N).**

Write Python code (you can copy the code below for now) and save (Shortcut: Ctrl+S) with .py file extension like: hello.py or your-first-program.py

**print("Hello, World!")**

Go to Run > Run module (Shortcut: F5) and we can see the output.

**IDENTIFIERS**

* Identifier is the name given to entities like class, functions, variables etc. in Python. It helps differentiating one entity from another.

**Rules for writing identifiers** Identifiers can be a combination of letters in lowercase (a to z) or uppercase (A to Z) or digits (0 to 9) or an underscore (\_). Names like myClass, var\_1 and print\_this\_to\_screen, all are valid example.

* An identifier cannot start with a digit. 1variable is invalid, but variable1 is perfectly fine.
* Keywords cannot be used as identifiers.

>>> global = 1

File "<interactive input>", line 1

global = 1

^

SyntaxError: invalid syntax

* We cannot use special symbols like !, @, #, $, % etc. in our identifier.

**>>> a@ = 0**

**File "<interactive input>", line 1**

**a@ = 0**

**^**

**SyntaxError: invalid syntax**

* Identifier can be of any length.
* Python is a case-sensitive language. This means, Variable and variable are not the same. Always name identifiers that make sense.

**Python Keywords**

Keywords are the reserved words in Python. We cannot use a keyword as variable name, function name or any other identifier. They are used to define the syntax and structure of the Python language. In Python, keywords are case sensitive. All the keywords except True, False and None are in lowercase and they must be written as it is. The list of all the keywords are given below.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| True | False | None | and | as |
| Asset | Def | Class | continue | break |
| Else | Finally | Elif | del | except |
| Global | For | If | from | import |
| Raise | Try | or | return | pass |
| nonlocal | In | not | is | lambda |

**Lines and Indentation**

Python does not use braces ({}) to indicate blocks of code for class and function definitions or flow control. Blocks of code are denoted by line indentation, which is rigidly enforced. The number of spaces in the indentation is variable, but all statements within the block must be indented the same amount. For example −

**if (True):**

**print ("True")**

**else:**

**print ("False")**

**Multi-Line Statements**

Statements in Python typically end with a new line. Python, however, allows the use of the **line continuation character (\)** to denote that the line should continue. For example −

total = item\_one + \

item\_two + \

item\_three

The statements contained within the [], {}, or () brackets do not need to use the line continuation character. For example −

days = ['Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday']

**Quotation in Python**

Python accepts single ('), double (") and triple (''' or """) quotes to denote string literals, as long as the same type of quote starts and ends the string. The triple quotes are used to span the string across multiple lines. For example, all the following are legal −

**word = 'word'**

**sentence = "This is a sentence."**

**paragraph = """This is a paragraph. It is**

**made up of multiple lines and sentences."""**

**Comments in Python**

Python supports two types of comments:

1) **Single lined comment**:

In case user wants to specify a single line comment, then comment must start with **#**

eg: **# This is single line comment**.

2) **Multi lined Comment**:

Multi lined comment can be given inside triple quotes.

eg: **''''' This**

**Is**

**Multipline comment'''**

**Multiple Statements on a Single Line**

The semicolon ( ; ) allows multiple statements on a single line given that no statement starts a new code block

For eg:

**If(a>b);print(“a is greater than b”)**

**Variables**

Variables are nothing but reserved memory locations to store values. It means that when you create a variable, you reserve some space in the memory. Based on the data type of a variable, the interpreter allocates memory and decides what can be stored in the reserved memory. Therefore, by assigning different data types to the variables, you can store integers, decimals or characters in these variables.

**Assigning Values to Variables**

Python variables do not need explicit declaration to reserve memory space. The declaration happens automatically when you assign a value to a variable. The equal sign (=) is used to assign values to variables.

**c = 100 # An integer assignment**

**m = 1000.0 # A floating point**

**name = "John" # A string**

**print (c)**

**print (m)**

**print (name) Output: 100**

**1000.0**

**John**

**Multiple Assignment**

Python allows us to assign a single value to several variables simultaneously.

For example −

**a = b = c = 1**

Here, an integer object is created with the value 1, and all the three variables are assigned to the same memory location. We can also assign multiple objects to multiple variables.

For example −

**a, b, c = 1, 2, "john"**

Here, two integer objects with values 1 and 2 are assigned to the variables a and b respectively, and one string object with the value "john" is assigned to the variable c.

**Standard Data Types**

The data stored in memory can be of many types.

Python provides following data types −

**Numeric Types: int, float, complex**

**Text Type: str**

**Sequence Types: list, tuple**

**Mapping Type: dict**

**Set Type: set**

**Boolean Type: bool**

**None Type: NoneType**

**Python Numbers**

**Number data types store numeric values**. Number objects are created when you assign a value to them. For example −

var1 = 1

We can also delete the reference to a number object by using the **del** statement. The syntax of the **del** statement is −

del var1[,var2[,var3[....,varN]]]]

You can delete a single object or multiple objects by using the **del** statement.

For example −

del var

Python supports three different numerical types −

* **int (signed integers)**
* **float (floating point real values)**
* **complex (complex numbers**)

A complex number consists of an ordered pair of real floating-point numbers denoted by **x + yj,** where x and y are real numbers and j is the imaginary unit.

**Python Strings**

**Strings** in Python are identified as **a contiguous set of characters** represented in the quotation marks. Python allows either pair of single or double quotes.

Subsets of strings can be taken using the **slice operator** ([ ] and [:] ) with indexes starting at 0 in the beginning of the string and working their way from -1 to the end.

The plus (+) sign is the string **concatenation operator** and the asterisk (\*) is the **repetition operator.**

str = 'Hello World!'

print (str) # Prints complete string

print (str[0]) # Prints first character of the string

print (str[2:5]) # Prints characters starting from 3rd to 5th

print (str[2:]) # Prints string starting from 3rd character

print (str \* 2) # Prints string two times

print (str + "TEST") # Prints concatenated string

print (str [ -1]) # Prints Last character of the string

**OUTPUT:**

Hello World!

H

llo

llo World!

Hello World!Hello World!

Hello World!TEST

!

**Python Lists**

**Lists** are the most versatile of Python's compound data types. **A list contains items separated by commas and enclosed within square brackets ([]).** To some extent, lists are similar to **arrays** in C. **One of the differences between them is that all the items belonging to a list can be of different data type**.

The values stored in a list can be accessed using the **slice operator ([ ] and [:])** with indexes starting at 0 in the beginning of the list and working their way to end -1. The plus (+) sign is the list **concatenation operator**, and the asterisk (\*) is the **repetition operator**.

For example −

list = [ 'abcd', 786 , 2.23, 'john', 70.2 ]

tinylist = [123, 'john']

print (list) # Prints complete list

print (list[0]) # Prints first element of the list

print (list[1:3]) # Prints elements starting from 2nd till 3rd

print (list[2:]) # Prints elements starting from 3rd element

print (tinylist \* 2) # Prints list two times

print (list + tinylist) # Prints concatenated lists

**OUTPUT**:

['abcd', 786, 2.23, 'john', 70.200000000000003]

abcd

[786, 2.23]

[2.23, 'john', 70.200000000000003]

[123, 'john', 123, 'john']

['abcd', 786, 2.23, 'john', 70.200000000000003, 123, 'john']

**Python Tuples**

**A tuple** is another sequence data type that is similar to the list. A tuple consists of a number of values separated by commas. Unlike lists, however, **tuples are enclosed within parenthesis.**

**The main difference between lists and tuples are − Lists are enclosed in brackets ( [ ] ) and their elements and size can be changed, while tuples are enclosed in parentheses ( ( ) ) and cannot be updated. Tuples can be thought of as read-only lists**.

For example −

tuple = ( 'abcd', 786 , 2.23, 'john', 70.2 )

tinytuple = (123, 'john')

print (tuple) # Prints complete tuple

print (tuple[0]) # Prints first element of the tuple

print (tuple[1:3]) # Prints elements starting from 2nd till 3rd

print (tuple[2:]) # Prints elements starting from 3rd element

print (tinytuple \* 2) # Prints tuple two times

print (tuple + tinytuple) # Prints concatenated tuple

OUTPUT:

('abcd', 786, 2.23, 'john', 70.200000000000003)

abcd

(786, 2.23)

(2.23, 'john', 70.200000000000003)

(123, 'john', 123, 'john')

('abcd', 786, 2.23, 'john', 70.200000000000003, 123, 'john')

The following code is invalid with tuple, because we attempted to update a tuple, which is not allowed. Similar case is possible with lists −

tuple = ( 'abcd', 786 , 2.23, 'john', 70.2 )

list = [ 'abcd', 786 , 2.23, 'john', 70.2 ]

tuple[2] = 1000 # Invalid syntax with tuple

list[2] = 1000 # Valid syntax with l

**Python Dictionary**

They work like associative arrays or hashes found in Perl and **consist of key-value pairs**. A dictionary key can be almost any Python type, but are usually numbers or strings. Values, on the other hand, can be any arbitrary Python object.

**Dictionaries are enclosed by curly braces ({ }) and values can be assigned and accessed using square braces ([]).**

For example −

dict = {}

dict['one'] = "This is one"

dict[2] = "This is two"

tinydict = {'name': 'john','code':6734, 'dept': 'sales'}

print (dict['one']) # Prints value for 'one' key

print (dict[2]) # Prints value for 2 key

print (tinydict) # Prints complete dictionary

print (tinydict.keys()) # Prints all the keys

print (tinydict.values()) # Prints all the values

OUTPUT−

This is one

This is two

{'name': 'john', 'dept': 'sales', 'code': 6734}

dict\_keys(['name', 'dept', 'code'])

dict\_values(['john', 'sales', 6734])

Dictionaries have no concept of order among the elements. It is incorrect to say that the elements are "out of order"; **they are simply unordered**.

**Set**

A set is an **unordered collection of items**. Every element is **unique (no duplicates) and must be immutable (which cannot be changed)**. However, **the set itself is mutable**. We can add or remove items from it. Sets can be used to perform mathematical set operations like **union, intersection, symmetric difference** etc.

A set is created by placing all the items (elements) inside curly braces {}, separated by comma or by using the built-in function set().It can have any number of items and they may be of different types (integer, float, tuple, string etc.). But a **set cannot have a mutable element, like list, set or dictionary, as its element**.

For eg:

**my\_set = {1, 2, 3}**

**print(my\_set) Output:{1,2,3}**

**my\_set = {1.0, "Hello", (1, 2, 3)}**

**print(my\_set) Output:{1.0,”Hello”,(1,2,3)}**

\* **Set do not have duplicates**

**my\_set = {1,2,3,4,3,2}**

**print(my\_set) Output:{1,2,2,4}**

\***Set cannot have mutable items**

**my\_set = {1, 2**

**, [3, 4]} Output: Type Error: unhashable type: 'list'**

\***Using set() Function**

**my\_set = set([1,2,3,2])**

**print(my\_set)** Output: {1, 2, 3}

\* **Creating an empty set**

**Empty curly braces {} will make an empty dictionary in Python.** To make a set without any elements we use the **set() function without any argument**.

For eg:

**a = set()**

\*We cannot access or change an element of set using indexing or slicing. Set does not support it. We can add single element using the **add()** method and multiple elements using the **update()** method. The update() method can take tuples, lists, strings or other sets as its argument. In all cases, duplicates are avoided.

For eg:

**my\_set = {1,3}**

**print(my\_set) Output:{1,3}**

**my\_set.add(2)**

**print(my\_set) Output:{1,2,3}**

**my\_set.update([2,3,4])**

**print(my\_set) Output:{1,2,3,4}**

**my\_set.update(((4,5), 1,6,8}))**

**print(my\_set) Output: {1, 2, 3, 4, (4,5), 6, 8}**

\***A particular item can be removed from set using methods, discard() and remove().**

**The only difference between the two is that, while using discard() if the item does not exist in the set, it remains unchanged. But remove() will raise an error in such condition**.

**The following example will illustrate this.**

**# initialize my\_set**

**my\_set = {1, 3, 4, 5, 6}**

**print(my\_set) Output: {1, 3, 4, 5, 6}**

**#discard an element**

**my\_set.discard(4)**

**print(my\_set) Output: {1, 3, 5, 6}**

**# remove an element**

**my\_set.remove(6)**

**print(my\_set) Output: {1, 3, 5}**

**# discard an element**

**# not present in my\_set**

**my\_set.discard(2)**

**print(my\_set) Output: {1, 3, 5}**

**# remove an element**

**# not present in my\_set**

**# we will get an error.**

**my\_set.remove(2) Output: Key Error: 2**

**Python Booleans**

Booleans represent one of two values: True or False.

**Python None Type**

None is a data type of its own (NoneType) and only None can be None.

The None keyword is used to define a null value, or no value at all. None is not the same as 0, False, or an empty string