**Python Introduction**

**Python** is a general purpose, dynamic, high level, and interpreted programming language. It supports Object Oriented programming approach to develop applications. It is simple and easy to learn and provides lots of high-level data structures.

Python is *easy to learn* yet powerful and versatile scripting language, which makes it attractive for Application Development.

Python's syntax and *dynamic typing* with its interpreted nature make it an ideal language for scripting and rapid application development.

Python supports *multiple programming pattern*, including object-oriented, imperative, and functional or procedural programming styles.

Python is not intended to work in a particular area, such as web programming. That is why it is known as *multipurpose* programming language because it can be used with web, enterprise, 3D CAD, etc.

We don't need to use data types to declare variable because it is *dynamically typed* so we can write a=10 to assign an integer value in an integer variable.

Python makes the development and debugging *fast* because there is no compilation step included in Python development, and edit-test-debug cycle is very fast.

Python Features

1) Easy to Learn and Use

Python is easy to learn and use. It is developer-friendly and high level programming language.

2) Expressive Language

Python language is more expressive means that it is more understandable and readable.

3) Interpreted Language

Python is an interpreted language i.e. interpreter executes the code line by line at a time. This makes debugging easy and thus suitable for beginners.

4) Cross-platform Language

Python can run equally on different platforms such as Windows, Linux, Unix and Macintosh etc. So, we can say that Python is a portable language.

5) Free and Open Source

Python language is freely available at offical web address. The source-code is also available. Therefore it is open source.

6) Object-Oriented Language

Python supports object-oriented language and concepts of classes and objects come into existence.

7) Extensible

It implies that other languages such as C/C++ can be used to compile the code and thus it can be used further in our python code.

8) Large Standard Library

Python has a large and broad library and provides rich set of module and functions for rapid application development.

9) GUI Programming Support

Graphical user interfaces can be developed using Python.

10) Integrated

It can be easily integrated with languages like C, C++, JAVA etc.

**Python Variables**

Variable is a name which is used to refer memory location. Variable also known as identifier and used to hold value.

In Python, we don't need to specify the type of variable because Python is a type infer language and smart enough to get variable type.

Variable names can be a group of both letters and digits, but they have to begin with a letter or an underscore.

It is recommended to use **lowercase letters for variable name**. Raghu and raghu both are two different variables.

**Declaring Variable and Assigning Values**

Python does not bound us to declare variable before using in the application. It allows us to create variable at required time.

We don't need to declare explicitly variable in Python. When we assign any value to the variable that variable is declared automatically.

The equal (=) operator is used to assign value to a variable.

**1. >>> print("hello world")**

**hello world**

2. Example: >>> x=y=z=143

>>> print x

File "<stdin>", line 1

print x

^

SyntaxError: Missing parentheses in call to 'print'. Did you mean print(x)?

>>> print(x)

143

>>> print(y)

143

>>> print(z)

143

## Python Variables

A variable is a named location used to store data in the memory. It is helpful to think of variables as a container that holds data which can be changed later throughout programming.

### Numeric Literals

Numeric Literals are immutable (unchangeable). Numeric literals can belong to 3 different numerical types Integer, Float and Complex.

a = 0b1010 #Binary Literals

b = 100 #Decimal Literal

c = 0o310 #Octal Literal

d = 0x12c #Hexadecimal Literal

#Float Literal

float\_1 = 10.5

float\_2 = 1.5e2

#Complex Literal

x = 3.14j

print(a, b, c, d)

print(float\_1, float\_2)

print(x, x.imag, x.real)

### String literals

A string literal is a sequence of characters surrounded by quotes. We can use both single, double or triple quotes for a string. And, a character literal is a single character surrounded by single or double quotes.

strings = "This is Python"

char = "C"

multiline\_str = """This is a multiline string with more than one line code."""

unicode = u"\u00dcnic\u00f6de"

raw\_str = r"raw \n string"

print(strings)

print(char)

print(multiline\_str)

print(unicode)

print(raw\_str)

**Type Conversion:**

The process of converting the value of one data type (integer, string, float, etc.) to another data type is called type conversion. Python has two types of type conversion.

1. Implicit Type Conversion
2. Explicit Type Conversion

**Basic Fundamentals:**

This section contains the basic fundamentals of Python like :

**i)Tokens and their types.**

**ii) Comments**

**a)Tokens:**

* Tokens can be defined as a punctuator mark, reserved words and each individual word in a statement.
* Token is the smallest unit inside the given program.

There are following tokens in Python:

* Keywords.
* Identifiers.
* Literals.
* Operators.

**Python Data Types**

Variables can hold values of different data types. Python is a dynamically typed language hence we need not define the type of the variable while declaring it. The interpreter implicitly binds the value with its type.

Python enables us to check the type of the variable used in the program. Python provides us the **type()** function which returns the type of the variable passed.

>>> A=143

>>> b="Hi Raghavendra"

>>> c = 143.012

>>> print(type(A));

<class 'int'>

>>> print(type(b));

<class 'str'>

>>> print(type(c));

<class 'float'>

## Standard data types

A variable can hold different types of values. For example, a person's name must be stored as a string whereas its id must be stored as an integer.

Python provides various standard data types that define the storage method on each of them. The data types defined in Python are given below.

1. Numbers
2. String
3. List
4. Tuple
5. Dictionary

### 1. Numbers

Number stores numeric values. Python creates Number objects when a number is assigned to a variable. For example;

a = 3 , b = 5  #a and b are number objects

**2.String**

The string can be defined as the sequence of characters represented in the quotation marks. In python, we can use single, double, or triple quotes to define a string.

String handling in python is a straightforward task since there are various inbuilt functions and operators provided.

In the case of string handling, the operator + is used to concatenate two strings as the operation *"hello"+" python"* returns *"hello python"*.

>>> str1 = 'hello raghavendra' #string str1

>>> str2 = ' how are you' #string str2

>>> print (str1[0:2])

he

>>> print (str1[4])

o

>>> print (str1\*2)

hello raghavendrahello raghavendra

>>> print (str1 + str2)

hello raghavendra how are you

### **3.List**

Lists are similar to arrays in C. However; the list can contain data of different types. The items stored in the list are separated with a comma (,) and enclosed within square brackets [].

We can use slice [:] operators to access the data of the list. The concatenation operator (+) and repetition operator (\*) works with the list in the same way as they were working with the strings.

>>> l = [1, "hi", "python", 2]

>>> print (l[3:]);

[2]

>>> print (l[0:2]);

[1, 'hi']

>>> print (l);

[1, 'hi', 'python', 2]

>>> print (l + l);

[1, 'hi', 'python', 2, 1, 'hi', 'python', 2]

>>> print (l \* 3);

[1, 'hi', 'python', 2, 1, 'hi', 'python', 2, 1, 'hi', 'python', 2]

**4.Tuples:**

* Tuple is another form of collection where different type of data can be stored.
* It is similar to list where data is separated by commas. Only the difference is that list uses square bracket and tuple uses parenthesis.
* Tuples are enclosed in parenthesis and cannot be changed.

>>> tuple=('raghu',100,60.4,'raghavendra')

>>> tuple1=('chaithu',10)

>>> tuple

('raghu', 100, 60.4, 'raghavendra')

>>> tuple[2:]

(60.4, 'raghavendra')

>>> tuple1[0]

'chaithu'

>>> tuple+tuple1

('raghu', 100, 60.4, 'raghavendra', 'chaithu', 10)

### **5. Dictionary:**

* Dictionary is a collection which works on a key-value pair.
* It works like an associated array where no two keys can be same.
* Dictionaries are enclosed by curly braces ({}) and values can be retrieved by square bracket([]).

>>> dictionary={'name':'Raghavendra','id':143,'dept':'it'}

>>> dictionary

{'name': 'Raghavendra', 'id': 143, 'dept': 'it'}

>>> dictionary.keys()

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

>>> dictionary.values()

dict\_values(['Raghavendra', 143, 'it'])

# Python If-else statements

Decision making is the most important aspect of almost all the programming languages. As the name implies, decision making allows us to run a particular block of code for a particular decision. Here, the decisions are made on the validity of the particular conditions. Condition checking is the backbone of decision making.

If Statement The if statement is used to test a specific condition. If the condition is true, a block of code (if-block) will be executed.

|  |  |
| --- | --- |
| If - else Statement | The if-else statement is similar to if statement except the fact that, it also provides the block of the code for the false case of the condition to be checked. If the condition provided in the if statement is false, then the else statement will be executed. |

|  |  |
| --- | --- |
| Nested if Statement | Nested if statements enable us to use if ? else statement inside an outer if statement. |

## What is a file?

File is a named location on disk to store related information. It is used to permanently store data in a non-volatile memory (e.g. hard disk).

Since, random access memory (RAM) is volatile which loses its data when computer is turned off, we use files for future use of the data.

When we want to read from or write to a file we need to open it first. When we are done, it needs to be closed, so that resources that are tied with the file are freed.

Hence, in Python, a file operation takes place in the following order.

1. Open a file
2. Read or write (perform operation)
3. Close the file

## How to open a file?

Python has a built-in function open() to open a file. This function returns a file object, also called a handle, as it is used to read or modify the file accordingly.

1. >>> f = open("test.txt") # open file in current directory
2. >>> f = open("C:/Python33/README.txt") # specifying full path

We can specify the mode while opening a file. In mode, we specify whether we want to read 'r', write 'w' or append 'a' to the file. We also specify if we want to open the file in text mode or binary mode.

The default is reading in text mode. In this mode, we get strings when reading from the file.

On the other hand, binary mode returns bytes and this is the mode to be used when dealing with non-text files like image or exe files.

|  |  |
| --- | --- |
| Mode | Description |
| 'r' | Open a file for reading. (default) |
| 'w' | Open a file for writing. Creates a new file if it does not exist or truncates the file if it exists. |
| 'x' | Open a file for exclusive creation. If the file already exists, the operation fails. |
| 'a' | Open for appending at the end of the file without truncating it. Creates a new file if it does not exist. |
| 't' | Open in text mode. (default) |
| 'b' | Open in binary mode. |
| '+' | Open a file for updating (reading and writing) |
| Python File Modes | |

1. f = open("test.txt") # equivalent to 'r' or 'rt'
2. f = open("test.txt",'w') # write in text mode
3. f = open("img.bmp",'r+b') # read and write in binary mode

Unlike other languages, the character 'a' does not imply the number 97 until it is encoded using ASCII (or other equivalent encodings).

Moreover, the default encoding is platform dependent. In windows, it is 'cp1252' but 'utf-8' in Linux.

So, we must not also rely on the default encoding or else our code will behave differently in different platforms.

Hence, when working with files in text mode, it is highly recommended to specify the encoding type.

1. f = open("test.txt",mode = 'r',encoding = 'utf-8')

## How to close a file Using Python?

When we are done with operations to the file, we need to properly close the file.

Closing a file will free up the resources that were tied with the file and is done using Python close() method.

Python has a garbage collector to clean up unreferenced objects but, we must not rely on it to close the file.

1. f = open("test.txt",encoding = 'utf-8')
2. # perform file operations
3. f.close()

This method is not entirely safe. If an exception occurs when we are performing some operation with the file, the code exits without closing the file.

A safer way is to use a [try...finally](https://www.programiz.com/python-programming/exception-handling) block.

1. try:
2. f = open("test.txt",encoding = 'utf-8')
3. # perform file operations
4. finally:
5. f.close()

This way, we are guaranteed that the file is properly closed even if an exception is raised, causing program flow to stop.

The best way to do this is using the with statement. This ensures that the file is closed when the block inside with is exited.

We don't need to explicitly call the close() method. It is done internally.

1. with open("test.txt",encoding = 'utf-8') as f:
2. # perform file operations

## How to write to File Using Python?

In order to write into a file in Python, we need to open it in write 'w', append 'a' or exclusive creation 'x' mode.

We need to be careful with the 'w' mode as it will overwrite into the file if it already exists. All previous data are erased.

Writing a string or sequence of bytes (for binary files) is done using write() method. This method returns the number of characters written to the file.

1. with open("test.txt",'w',encoding = 'utf-8') as f:
2. f.write("my first file\n")
3. f.write("This file\n\n")
4. f.write("contains three lines\n")

This program will create a new file named 'test.txt' if it does not exist. If it does exist, it is overwritten.

We must include the newline characters ourselves to distinguish different lines.

## How to read files in Python?

To read a file in Python, we must open the file in reading mode.

There are various methods available for this purpose. We can use the read(size) method to read in size number of data. If size parameter is not specified, it reads and returns up to the end of the file.

1. >>> f = open("test.txt",'r',encoding = 'utf-8')
2. >>> f.read(4) # read the first 4 data
3. 'This'
4. >>> f.read(4) # read the next 4 data
5. ' is '
6. >>> f.read() # read in the rest till end of file
7. 'my first file\nThis file\ncontains three lines\n'
8. >>> f.read() # further reading returns empty sting
9. ''

We can see that, the read() method returns newline as '\n'. Once the end of file is reached, we get empty string on further reading.

We can change our current file cursor (position) using the seek() method. Similarly, the tell() method returns our current position (in number of bytes).

1. >>> f.tell() # get the current file position
2. 56
3. >>> f.seek(0) # bring file cursor to initial position
4. 0
5. >>> print(f.read()) # read the entire file
6. This is my first file
7. This file
8. contains three lines

We can read a file line-by-line using a [for loop](https://www.programiz.com/python-programming/for-loop). This is both efficient and fast.

1. >>> for line in f:
2. ... print(line, end = '')
3. ...
4. This is my first file
5. This file
6. contains three lines

The lines in file itself has a newline character '\n'.

Moreover, the print() end parameter to avoid two newlines when printing.

Alternately, we can use readline() method to read individual lines of a file. This method reads a file till the newline, including the newline character.

1. >>> f.readline()
2. 'This is my first file\n'
3. >>> f.readline()
4. 'This file\n'
5. >>> f.readline()
6. 'contains three lines\n'
7. >>> f.readline()
8. ''

Lastly, the readlines() method returns a list of remaining lines of the entire file. All these reading method return empty values when end of file (EOF) is reached.

1. >>> f.readlines()
2. ['This is my first file\n', 'This file\n', 'contains three lines\n']