# Python Introduction

S1 MCA

# INTRODUCTION

- **Python** is a high-level, interpreted programming language that is:
- ❓ **Easy to learn and use**
- 📝 **Readable and clear** syntax
- 🔁 **Cross-platform** (runs on Windows, macOS, Linux)
- 🚀 Used for **web development, automation, data science, AI, game development, scripting**, and more

# Features in Python

- **Free and Open Source:**

Python language is freely available at the official website. Since it is open-source, this means that source code is also available to the public. So you can download it, use it as well as share it.

- **Easy to code:**

Python is a **high-level programming language**. Python is very easy to learn the language as compared to other languages like C, C#, Javascript, Java, etc. It is very easy to code in the Python language and anybody can learn Python basics in a few hours or days. It is also a developer-friendly language.

- **Easy to Read:**

As you will see, learning Python is quite simple. As was already established, Python's syntax is really straightforward. The code block is defined by the indentations rather than by semicolons or brackets.

- **Object-Oriented Language**

One of the key features of python is **Object-Oriented Programming**. Python supports object-oriented language and concepts of classes, object encapsulation, etc.

- **GUI Programming Support**

Graphical User interfaces can be made using a module such as pyQt5, PyQt4, wxPython, or Tk in Python. PyQt5 is the most popular option for creating graphical apps with Python.

- **High-Level Language**

Python is a high-level language. When we write programs in Python, we do not need to remember the system architecture, nor do we need to manage the memory.

- **Large Community Support**

Python has gained popularity over the years. questions are constantly answered by the enormous StackOverflow community. These websites have already provided answers to many questions about Python, so Python users can consult them as needed.

- **Easy to Debug**

Excellent information for mistake tracing. You will be able to quickly identify and correct the majority of your program's issues once you understand how to interpret Python's error traces. Simply by glancing at the code, you can determine what it is designed to perform.

- **Python is a Portable language**

Python language is also a portable language. For example, if we have Python code for Windows and if we want to run this code on other platforms such as Linux, Unix, and Mac then we do not need to change it, we can run this code on any platform.

- **Python is an Integrated language**

Python is also an Integrated language because we can easily integrate Python with other languages like C, C++, etc.

- **Interpreted Language:**

Python is an Interpreted Language because Python code is executed line by line at a time. like other languages C, C++, Java etc. there is no need to compile Python code this makes it easier to debug our code. The source code of Python is converted into an immediate form called **bytecode**.

- **Large Standard Library**

Python has a large standard Library that provides a rich set of modules and functions so you do not have to write your own code for every single thing.

- **Dynamically Typed Language**

Python is a dynamically-typed language. That means the type (for example- int, double, long, etc.) for a variable is decided at run time not in advance because of this feature we don't need to specify the type of variable.

- **Frontend and backend development**

With a new project py script, you can run and write Python codes in HTML with the help of some simple tags <py-script>, <py-env>, etc. This will help you do frontend development work in Python like javascript. Backend is the strong forte of Python it's extensively used for this work cause of its frameworks like Django and Flask.

- **Allocating Memory Dynamically**

In Python, the variable data type does not need to be specified. The memory is automatically allocated to a variable at runtime when it is given a value. Developers do not need to write int y = 18 if the integer value 15 is set to y. You may just type y=18.

# Python Install

- Many PCs and Macs will have python already installed.
- To check if you have python installed on a Windows PC, search in the start bar for Python or run the following on the Command Line (cmd.exe):
- C:\Users*Your Name*>python --version

To check if you have python installed on a Linux or Mac, then on linux open the command line or on Mac open the Terminal and type:

- python --version

# FOUNDER

# Python Keywords and Identifiers

- **Keywords in Python**

**Python Keywords** are some predefined and reserved words in Python that have special meanings. Keywords are used to define the syntax of the coding. The keyword cannot be used as an identifier, function, or variable name. All the keywords in Python are written in lowercase except True and False.

```
# code for list all keywords

import keyword

print(keyword.kwlist)
```

```
['False', 'None', 'True', 'and', 'as', 'assert', 'async', 'await', 'break', 'class', 'continue', 'def', 'del', 'elif', 'else', 'except', 'finally', 'for', 'from', 'global', 'if', 'import', 'in', 'is',...]
```

- **Identifiers in Python**

**Identifier** is a user-defined name given to a variable, function, class, module, etc. The identifier is a combination of character digits and an underscore. They are case-sensitive i.e., 'num' and 'Num' and 'NUM' are three different identifiers in python. It is a good programming practice to give meaningful names to identifiers to make the code understandable.

**Rules for Naming Python Identifiers**

- It cannot be a reserved python keyword.
- It should not contain white space.
- It can be a combination of A-Z, a-z, 0-9, or underscore.
- It should start with an alphabet character or an underscore ( _ ).
- It should not contain any special character other than an underscore ( _ ).

***Valid identifiers:***

- *var1*
- *_var1*
- *_1_var*
- *var_1*

***Invalid Identifiers***

- *!var1*
- *1var*
- *1_var*
- *var#1*
- *var 1*

# Python Indentation

- Indentation refers to the spaces at the beginning of a code line.

Where in other programming languages the indentation in code is for readability only, the indentation in Python is very important.

Python uses indentation to indicate a block of code.

- ✓

```
if 5 > 2:
        print("Five is greater than two!")
```

- ✗

```
if 5 > 2:
print("Five is greater than two!")
```

- ✓The number of spaces is up to you as a programmer, the most common use is four, but it has to be at least one.

```
if 5 > 2:
        print("Five is greater than two!")
if 5 > 2:
                print("Five is greater than two!")
```

- ✗You have to use the same number of spaces in the same block of code, otherwise Python will give you an error☺

```
if 5 > 2:
        print("Five is greater than two!")
                print("Five is greater than two!")
```

# Comments

- Python has commenting capability for the purpose of in-code documentation.
- Comments start with a #, and Python will render the rest of the line as a comment
- Comments can be used to explain Python code.
- Comments can be used to make the code more readable.
- Comments can be used to prevent execution when testing code.

EX:

- print("Hello, World!") #This is a comment
- #print("Hello, World!")
  print("Cheers, Mate!")

# Multiline Comments

- Since Python will ignore string literals that are not assigned to a variable, you can add a multiline string (triple quotes) in your code, and place your comment inside it.
- """
  This is a comment
  written in
  more than just one line
  """
  print("Hello, World!")

# Variables in Python

- Variables are reserved memory locations to store values.
- Python has no command for declaring a variable.
- A variable is created the moment you first assign a value to it. This means that when you create a variable thr interpreter allocate memory and memory and what can be stored in the reserved memory.
- Based on the data type of a variable, memory space is allocated to it. Therefore, by assigning different data types to Python variables, you can store integers, decimals or characters in these variables.
- the equal(=) sign is used to assign value to varables.

Ex:

- printing variables

```
counter = 100          # Creates an integer variable
miles   = 1000.0       # Creates a floating point variable
name    = "Zara Ali"   # Creates a string variable

print (counter)
print (miles)
print (name)
```

Output:

```
100
1000.0
Zara Ali
```

# Deleting Python Variables

- You can delete the reference to a number object by using the del statement. The syntax of the del statement is

  syntax: del var1,var2,var3....,varN

EX:

```
counter = 100
print (counter)
del counter
print (counter)
```

```
Traceback (most recent call last):
  File "main.py", line 7, in <module>
    print (counter)
NameError: name 'counter' is not defined
```

# Getting Type of a Variable

- You can get the data type of a Python variable using the python built-in function type() as follows.

```
x = "Zara"
y =  10
z =  10.10
print(type(x))
print(type(y))
print(type(z))
```

This will produce the following result:

- <class 'str'>
- <class 'int'>
- <class 'float'>

# Multi Words Variable Names

- Camel Case

Each word, except the first, starts with a capital letter:

myVariableName = "John"

- Pascal Case

Each word starts with a capital letter:

MyVariableName = "John"

- Snake Case

Each word is separated by an underscore character:

my_variable_name = "John"

# Casting Python Variables

- **Type casting** is the process of **converting one data type into another**.
- You can specify the data type of a variable with the help of casting as follows:

```
</>

x = str(10)    # x will be '10'
y = int(10)    # y will be 10
z = float(10)  # z will be 10.0

print( "x =", x )
print( "y =", y )
print( "z =", z )
```

```
x = 10
y = 10
z = 10.0
```

# Case-Sensitivity of Python Variables

Python variables are case sensitive which means Age and age are two different variables:

```
</>

age = 20
Age = 30

print( "age =", age )
print( "Age =", Age )
```

```
age = 20
Age = 30
```

# Python Variables - Multiple Assignment

- Python allows to initialize more than one variables in a single statement. In the following case, three variables have same value.

```
>>> a=10
>>> b=10
>>> c=10
```

- Instead of separate assignments, you can do it in a single assignment statement as follows –

```
>>> a=b=c=10
>>> print (a,b,c)
10 10 10
```

```
>>> a=10
>>> b=20
>>> c=30
```

```
>>> a,b,c = 10,20,30
>>> print (a,b,c)
10 20 30
```

```
a = b = c = 100

print (a)
print (b)
print (c)
```

```
100
100
100
```

```
a,b,c = 1,2,"Zara Ali"

print (a)
print (b)
print (c)
```

```
1
2
Zara Ali
```

- String variables can be declared either by using single or double quotes:

```
x = "John"
# is the same as
x = 'John’
```

**Unpack a Collection**

- If you have a collection of values in a list, tuple etc. Python allows you to extract the values into variables. This is called *unpacking*.

```
fruits = ["apple", "banana", "cherry"]
x, y, z = fruits
print(x)
print(y)
print(z)
```

# OUTPUT VARIABLE

The Python `print()` function is often used to output variables.

Example:

```
x = "Python is awesome"
print(x)
```

- In the print() function, you output multiple variables, separated by a comma:

Example

```
x = "Python"
y = "is"
z = "awesome"
print(x, y, z)
```

- You can also use the + operator to output multiple variables:

```
x = "Python "
y = "is "
z = "awesome"
print(x + y + z)
```

**Notice the space character after "Python " and "is ", without them the result would be "Pythonisawesome".**

- For numbers, the + character works as a mathematical operator:

```
x = 5
y = 10
print(x + y)
```

- In the print() function, when you try to combine a string and a number with the + operator, Python will give you an error:

```
x = 5
y = "John"
print(x + y)
```

- The best way to output multiple variables in the print() function is to separate them with commas, which even support different data types:

```
x = 5
y = "John"
print(x, y)
```

# Python - Global Variables

- Variables that are created outside of a function (as in all of the examples in the previous pages) are known as global variables.
- Global variables can be used by everyone, both inside of functions and outside.

```
x = "awesome"

def myfunc():
  print("Python is " + x)

myfunc()
```

If you create a variable with the same name inside a function, this variable will be local, and can only be used inside the function. The global variable with the same name will remain as it was, global and with the original value.

```
x = "awesome"

def myfunc():
  x = "fantastic"
  print("Python is " + x)

myfunc()

print("Python is " + x)
```

# The global Keyword

- Normally, when you create a variable inside a function, that variable is local, and can only be used inside that function.
- To create a global variable inside a function, you can use the global keyword.
- If you use the global keyword, the variable belongs to the global scope:

```
def myfunc():
  global x
  x = "fantastic"

myfunc()

print("Python is " + x)
```

Also, use the global keyword if you want to change a global variable inside a function.

Example

To change the value of a global variable inside a function, refer to the variable by using the global keyword:

```
x = "awesome"

def myfunc():
  global x
  x = "fantastic"

myfunc()

print("Python is " + x)
```

**What is a correct way to declare a Python variable?**

- `var x = 5`
- `#x = 5`
- `$x = 5`
- `x = 5`

**True or False:**

**You can declare string variables with single or double quotes.**

```
x = "John"
# is the same as
x = 'John'
```

- True
- False

**True or False:**

**Variable names are not case-sensitive.**

```
a = 5
# is the same as
A = 5
```

- True
- False

Submit Answer »

**Select the correct functions to print the data type of a variable:**

( (myvar))

typ type var print echo

Submit Answer »

# INPUT Function

**Reading the Input**

python provides built in functions to read a line of text from standard input those are

- raw_input
- input

**raw_input Function:**

syntax -> raw_input([your input])

This prompt allow you to enter the string and display the same string while print

str =raw_input(“Enter Name:”)

print(“Name:”,str)

output:

Enter Name: Albin

Name: Albin

**Input Function:**

syntax: input([Enter Prompt])

This fuction will assume the input as a valid python expresion and returns the evaluated result

```
val1 = raw_input("Enter the name: ")
print(type(val1))
print(val1)

val2 = raw_input("Enter the number: ")
print(type(val2))
```

# Python Numbers

There are three numeric types in Python:

- int
- float
- Complex

To verify the type of any object in Python, use the type() function:

## Int

Int, or integer, is a whole number, positive or negative, without decimals, of unlimited length.

```
x = 1
y = 35656222554887711
z = -3255522

print(type(x))
print(type(y))
print(type(z))
```

## Float

Float, or "floating point number" is a number, positive or negative, containing one or more decimals.

```
x = 1.10
y = 1.0
z = -35.59

print(type(x))
print(type(y))
print(type(z))
```

Float can also be scientific numbers with an "e" to indicate the power of 10.

```
x = 35e3
y = 12E4
z = -87.7e100

print(type(x))
print(type(y))
print(type(z))
```

**Complex**

Complex numbers are written with a "j" as the imaginary part:

```
x = 3+5j
y = 5j
z = -5j

print(type(x))
print(type(y))
print(type(z))
```

# Random Number

- Python does not have a **random()** function to make a **random** number, but Python has a built-in module called random that can be used to make random numbers:

```
import random

print(random.randrange(1, 10))
```

# Python Strings

- Strings in python are surrounded by either single quotation marks, or double quotation marks.

  'hello' is the same as "hello“

- Quotes Inside Quotes

```
print("It's alright")
print("He is called 'Johnny'")
print('He is called "Johnny"')
```

- Assign String to a Variable

```
a = "Hello"
print(a)
```

- **Multiline Strings**

You can assign a multiline string to a variable by using three quotes:

```
a = """Lorem ipsum dolor sit amet,
consectetur adipiscing elit,
sed do eiusmod tempor incididunt
ut labore et dolore magna aliqua."""
print(a)
```

```
a = '''Lorem ipsum dolor sit amet,
consectetur adipiscing elit,
sed do eiusmod tempor incididunt
ut labore et dolore magna aliqua.'''
print(a)
```

# Strings are Arrays

- Like many other popular programming languages, strings in Python are arrays of unicode characters.

- However, Python does not have a character data type, a single character is simply a string with a length of 1.

- Square brackets can be used to access elements of the string.

- a = "Hello, World!"
  print(a[1])

# Looping Through a String

- Since strings are arrays, we can loop through the characters in a string, with a for loop.

```
>>>for x in "banana":
      print(x)
```

- **String Length**

To get the length of a string, use the len() function.

- **Check String**

To check if a certain phrase or character is present in a string, we can use the keyword in.

```
txt = "The best things in life are free!"
print("free" in txt)
```

- **Use it in an if statement:**

```
txt = "The best things in life are free!"
if "free" in txt:
  print("Yes, 'free' is present.")
```

- **Check if NOT**

To check if a certain phrase or character is NOT present in a string, we can use the keyword not in.

```
txt = "The best things in life are free!"
print("expensive" not in txt)
```

# Python - Slicing Strings

- **Slicing**

You can return a range of characters by using the slice syntax.

Specify the start index and the end index, separated by a colon, to return a part of the string.

### Example

Get the characters from position 2 to position 5 (not included):

```
b = "Hello, World!"
print(b[2:5])
```

The first character has index 0.

- Slice From the Start

By leaving out the start index, the range will start at the first character:

### Example

Get the characters from the start to position 5 (not included):

```
b = "Hello, World!"
print(b[:5])
```

- Slice To the End

By leaving out the *end* index, the range will go to the end:

### Example

Get the characters from position 2, and all the way to the end:

```
b = "Hello, World!"
print(b[2:])
```

# Negative Indexing

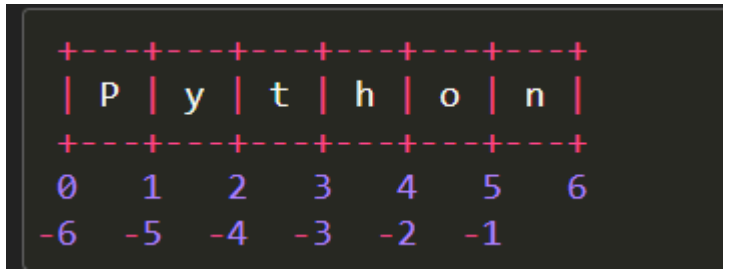
- Use negative indexes to start the slice from the end of the string:

Get the characters:

From: "o" in "World!" (position -5)

To, but not included: "d" in "World!" (position -2):

```
b = "Hello, World!"
print(b[-5:-2])
```



# Python - Modify Strings

- The upper() method returns the string in upper case:

```
a = "Hello, World!"
print(a.upper())
```

- The lower() method returns the string in lower case:

```
a = "Hello, World!"
print(a.lower())
```

- The strip() method removes any whitespace from the beginning or the end:

```
a = " Hello, World! "
print(a.strip()) # returns "Hello, World!"
```

- The replace() method replaces a string with another string:

```
a = "Hello, World!"
print(a.split(",")) # returns ['Hello', ' World!']
```

# Python - String Concatenation

- To concatenate, or combine, two strings you can use the + operator.

Merge variable `a` with variable `b` into variable `c` :

```
a = "Hello"
b = "World"
c = a + b
print(c)
```

To add a space between them, add a `" "` :

```
a = "Hello"
b = "World"
c = a + " " + b
print(c)
```

# Python - Format - Strings

we can combine strings and numbers by using f-strings or the **format()** method!

- **F-Strings:**

F-String was introduced in Python 3.6, and is now the preferred way of formatting strings. To specify a string as an f-string, simply put an f in front of the string literal, and add curly brackets {} as placeholders for variables and other operations.

```
age = 36
txt = f"My name is John, I am {age}"
print(txt)
```

# Placeholders and Modifiers

A placeholder can contain variables, operations, functions, and modifiers to format the value.

```
price = 59
txt = f"The price is {price} dollars"
print(txt)
```

- A placeholder can include a modifier to format the value.

A modifier is included by adding a colon : followed by a legal formatting type, like .2f which means fixed point number with 2 decimals:

Display the price with 2 decimals:

```
price = 59
txt = f"The price is {price:.2f} dollars"
print(txt)
```

- A placeholder can contain Python code, like math operations:

Perform a math operation in the placeholder, and return the result:

```
txt = f"The price is {20 * 59} dollars"
print(txt)
```

# Python - Escape Characters

- To insert characters that are illegal in a string, use an escape character.
- An escape character is a backslash \ followed by the character you want to insert.
- An example of an illegal character is a double quote inside a string that is surrounded by double quotes:

You will get an error if you use double quotes inside a string that is surrounded by double quotes:

```
txt = "We are the so-called "Vikings" from the north."
```

- To fix this problem, use the escape character \":

The escape character allows you to use double quotes when you normally would not be allowed:

```
txt = "We are the so-called \"Vikings\" from the north."
```

# Escape Characters

Other escape characters used in Python:

| Code | Result |
| --- | --- |
| \' | Single Quote |
| \\ | Backslash |
| \n | New Line |
| \r | Carriage Return |
| \t | Tab |
| \b | Backspace |
| \f | Form Feed |
| \ooo | Octal value |
| \xhh | Hex value |

# Python - String Methods

- Python has a set of built-in methods that you can use on strings.

ASSIGNMENT

Method | Description | EX

| Method | Description |
|---|---|
| capitalize() | Converts the first character to upper case |
| casefold() | Converts string into lower case |
| center() | Returns a centered string |
| count() | Returns the number of times a specified value occurs in a string |
| encode() | Returns an encoded version of the string |

# Python Booleans

- Booleans represent one of two values: True or False.
  You can evaluate any expression in Python, and get one of two answers, True or False.

Example

```
print(10 > 9)
print(10 == 9)
print(10 < 9)
```

# Evaluate Values and Variables

- The bool() function allows you to evaluate any value, and give you True or False in return,

Almost any value is evaluated to True if it has some sort of content.

Any string is True, except empty strings.

Any number is True, except 0.

- Any list, tuple, set, and dictionary are True, except empty ones.

In fact, there are not many values that evaluate to False, except empty values, such as (), [], {}, "", the number 0, and the value None. And of course the value False evaluates to False.

The following will return True:

```
bool("abc")
bool(123)
bool(["apple", "cherry", "banana"])
```

The following will return False:

```
bool(False)
bool(None)
bool(0)
bool("")
bool(())
bool([])
bool({})
```