# Familiarization with the basics of Python programming

## Computer Program:

A computer program is a set of instructions that can be executed by the computer to perform and solve a certain task. These programs are written in a special language known as Programming Language.

## Computer Programming:

It is the process to create a computer program.

## Python Programming Language:

Python is an interpreted, high-level programming language. It was developed by Guido van Rossum. It is user-friendly and is most popular for its easy-to-use syntax and readable code.

## Features of Python

- High-level Programming language.
- Interpreted language (as Python programs are executed by an interpreter)

- Easy to use
- Simple Syntax
- Python programs are generally written with fewer Lines of Code as compared to other programming languages.
- Case-sensitive. For example, 'NUMBER' and 'number' are treated differently in Python.
- Portable programming language - has ability run programs on many computer architectures and operating systems.

*Syntax: Set of rules for framing a valid statement in a programming language.*

## Working in Python

- Install Python on the computer (https://www.python.org/downloads/). Refer Appendix for installation instructions.
- Use Python IDLE (Integrated Development and Learning Environment) for developing python Programs.

## How to Display Data

print( ) function is used to print a message on the screen.

## A Simple Hello World Program

print("Hello World")

```
>>> print("Hello World")
Hello World
```

## Modes of working in Python

- Interactive Mode
- Script Mode

## Interactive Mode

In Interactive Mode, a python statement is executed in a command-line shell. It provides instant feedback for each statement while keeping prior statements in memory.

## Script Mode

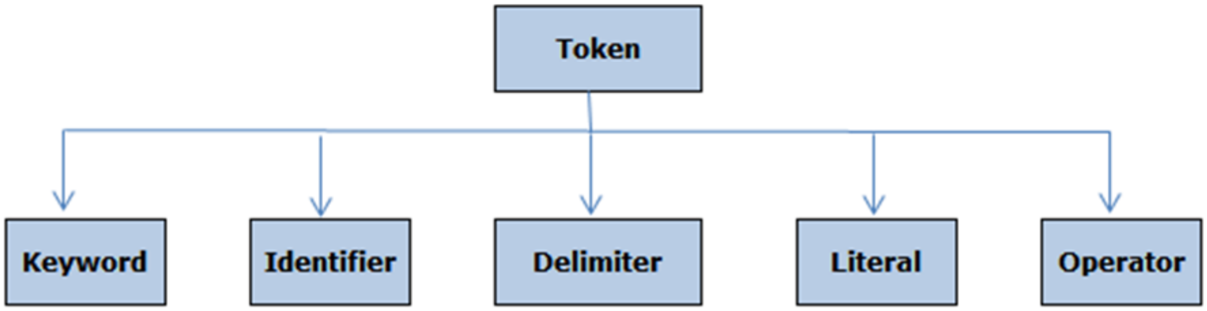
In script mode, the instructions are saved in a file with a '**.py**' extension and executed from the beginning of the file. This mode is suitable for creating and running large programs. In script mode,  all commands are stored in the form of a program or a script.

## Character Set:

A character set is a collection of valid characters that can be recognized by a language. Python Language recognises the following characters as valid:

|  |  |
|---|---|
| Letters | : A-Z, a-z |
| Digits | : 0-9 |
| Special Symbol | : + / @ ! - = <> == etc. |
| Whitespaces | : '\n', '\t' etc. |

## Tokens
Tokens are the smallest individual units of a program.



## Keywords
Keywords are reserved words with special meaning (known to the interpreter). These can not be used as identifiers in a program.
Example: for, while, else, if

## Identifier
A variable, function, class, module, or other objects are identified by a name known as identifier.
## Rules for naming an identifier:
1. First character of a variable can be an alphabet(A-Z or a-z) or an underscore(_).
2. Next characters of a variable can be an alphabet(A-Z or a-z), an underscore(_) or a digit.
3. Keywords cannot be used as variables.
4. First character cannot be a digit.
5. Special characters including white spaces are not allowed.

## Literals:
Literals are data-items that have a fixed value of a certain data type, such as a number, string, boolean, or None. They are also known as Constants.
Example :

|  |  |
|---|---|
| String literals (Text) | : 'Hello World' |
| Numeric literals (Numbers) | : 3.14 |
| Boolean literals (Truth Value) | : True/False |
| None Literal | : The None keyword is used to define a null value or absence of a value. None is different from 0. |

## Operators and Operands

Operators are symbols (or keywords) that perform various operations on the operands. An operand is a variable or a literal on which the operation is performed.
Example: 50 + 20
Here 50 and 20 are operands, and + is the operator.

## Arithmetic Operators

**:** Arithmetic operators perform mathematical operations like addition, subtraction, multiplication, division, floor division, exponent and modulus. (+,-,*,/,//,**,%)

## Relational Operators:

Relational operators perform comparison between values. An expression having relational operators evaluates to either True or False. Example >, <, >=, <=, ==, !=

## Python Comments

Comments are descriptions about the code. They help other programmers understand the functionality of the code. Comments are ignored by the Python interpreter, and are only relevant to the programmer.

## Variable

Variables refer to an object (data items like int, float, list, dictionary etc) stored in the memory. Value stored in a variable can be changed during the program execution.
Rules for naming a variable are the same as the rules for naming an Identifier.
Example: pri = 5
Here pri is a variable with value 5.

**Valid variable name examples** : name, Age, _total
**Invalid variable name examples** : print value (whitespace not allowed), 1_gender (cannot start with a digit)

## Concept of L Value and R Value

In Python, the l-value refers to the left-hand side of an assignment operator, while the r-value refers to the right-hand side of an assignment operator.
The l-value is associated with a valid memory location in the computer. The memory location can be checked by using the id( ) function.
The r-value may be any valid expression which is executed by the interpreter.

Consider the following assignment statement:

x = 5+2

In the statement above, the l-value is 'x' as it is on the left-hand side of the = operator.

The r-value is 7 as it is on the right-hand side of the = operator, and is generated by performing the addition operation on 5 and 2.

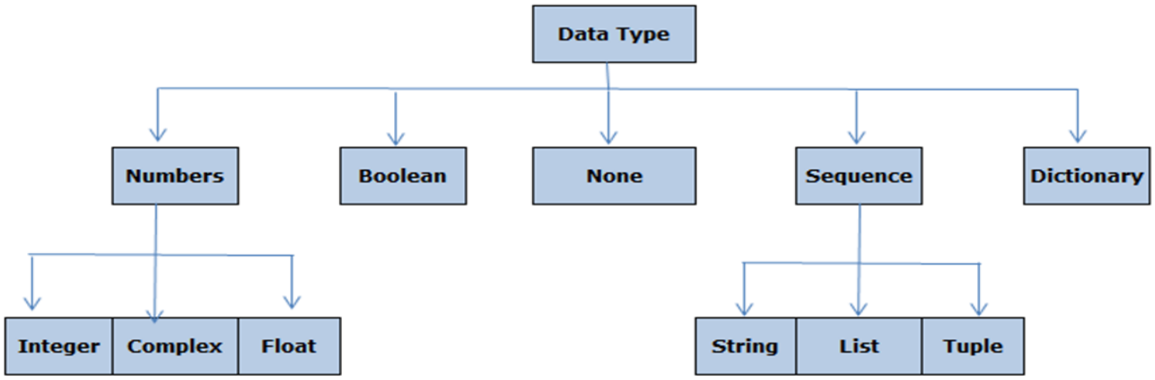
The memory location of x may be checked by executing the command id(x).

---

## Knowledge of Data Types

### Data Type

Data type represents the type of data a Variable or a Literal is referring to. Each data type has specific characteristics and operations associated with it. In Python, there are various data types, including number, string, boolean, list, tuple, and dictionary.



### Mutable and Immutable Data Objects

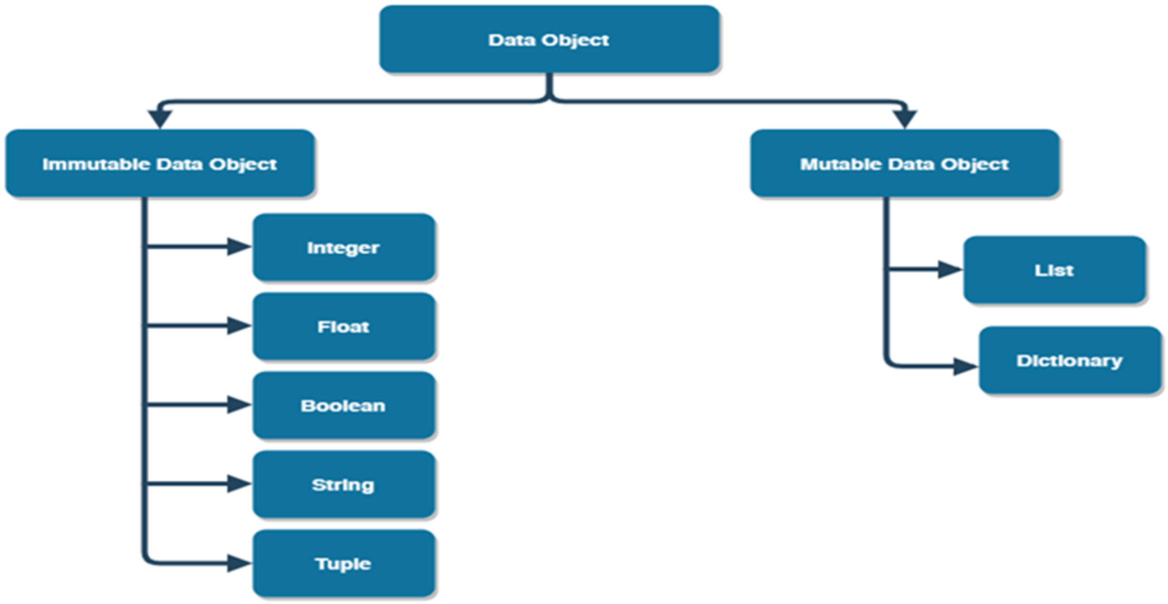
Python variables are memory references. It may be required to change or update the value of an object referenced by a variable. However, for certain data objects, Python does not allow us to change the value of an object.

### Mutable Objects:

Mutable data objects are objects that can be changed after they are created. It is possible to add, remove, or modify elements within these data types. Example of mutable data types: List, Set and Dictionary.

### Immutable Objects :

Objects whose values cannot be changed after they are created are called immutable objects. To change the value, a new object is created. Example of immutable data types: Number (Integer, Float), String, and Tuple.



## Numeric Data Types (Number)

Python has three numeric data types:

1. Integer
2. Float
3. Complex

### Integer

- Numbers with No Fractional Part
- Can be Positive or Negative
- In Python 3, the int type has no max limit. Values can be as large as the available memory allows.
- Example 100, 0o55 (octal number), 0x68(hexadecimal number)

### Float

- Numbers with Fractional Part
- Example 3.14, .314E01

### Complex

- Numbers with both real and imaginary components
- A complex number is represented by "x + yj". Example 10 + 9j

### Boolean

A boolean data type can assume one of the two possible values : True or False.

**Sequence:** Sequence is an ordered collection of items or elements which includes several built-in types as String, List, and Tuple. Values in the sequence are called elements/items. Each element in a sequence has a unique index.

## String:

- A string is an ordered sequence of characters enclosed in single/double/triple quotes.
- Single Line String : Terminates in a single line. Example – **'This is an example of single line'**
- Multi Line String : Does not terminate in a single line. A multiline string may be created using three quotes

  Example -
  ```
  '''This is a
     Multiline
     string'''
  ```

## List:

- List is an ordered sequence data type which can store values of any data type.
- List is mutable.
- List is enclosed in square brackets [ ]
- Example : [ ] is an empty List,
  `[5, 6.5, True, 'Hello']` is a List having 5, 6.5, True and 'Hello' as four elements.

## Tuple:

- Tuple is an ordered sequence which can store values of any data type.
- They are immutable, i.e the items of a Tuple cannot be updated after creation.
- Tuple is enclosed in parenthesis ( )
- Example : t=( ) is an empty Tuple
  t=(9,) is a Tuple with 9 as an item
  t=`(8, 5, 9.5, False)` is a Tuple with 8, 5, 9.5, False as four items

## Dictionary:

- Dictionary in Python stores items in the form of key-value pairs
- Syntax is dict_variable = {key1:value1, key2:value2, …, key-n:value-n}
- Items in a dictionary are enclosed in curly brackets { } and are separated by commas
- In a key-value pair, the key is separated from the value using a colon (:)
- To access any value in the dictionary, specify the key as the index using square brackets [ ].
- Example : { } is an empty dictionary,
  ```
  D = {'name':'Python','version':'3.7.2', 'OS': 'Windows'}
  print(D['version']) # shows 3.7.2 as output
  ```

## Special Data-type: None

The None data type/keyword is used to indicate absence of a value (No value or Missing Value).

**Operand**: Value(s) required for operation of operator.

**Operators:** Operators can manipulate the value of operands (variables or values). Various symbols are used as operators.

```
a = 5
b = 6
sum = a + b
```

In above example, + and = are the operators, a, b are operands and sum is a variable.

## Expression

An expression is combination of operators, operands, literals and parenthesis. An expression produces a value when evaluated.

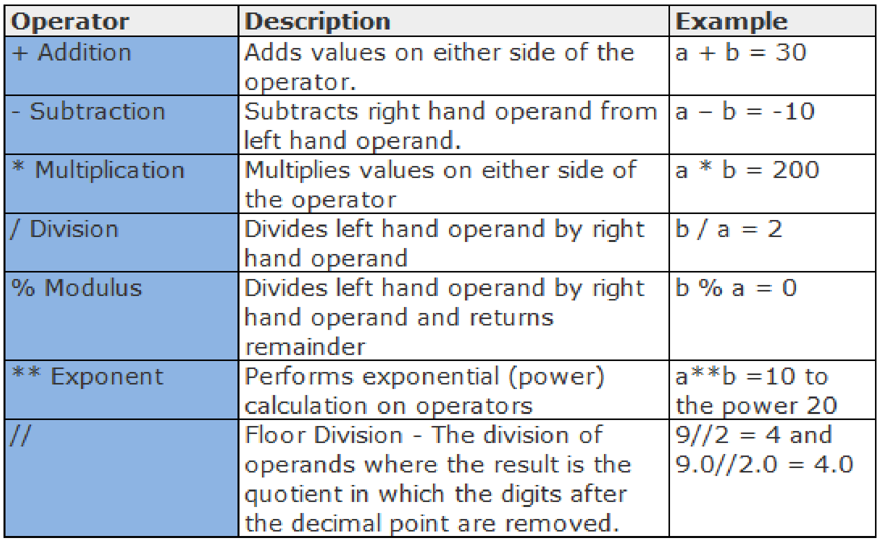
## Python Statement

In Python, a statement is an instruction that the interpreter can execute.

## Types of Operators

1. Arithmetic Operators
2. Relational Operators
3. Logical Operators
4. Assignment Operators
5. Identity Operators
6. Membership Operators

**Arithmetic operators:** Arithmetic operators perform mathematical operations such as addition, subtraction, multiplication, division, and modulus.



| Operator | Description | Example |
|---|---|---|
| + Addition | Adds values on either side of the operator. | a + b = 30 |
| - Subtraction | Subtracts right hand operand from left hand operand. | a – b = -10 |
| * Multiplication | Multiplies values on either side of the operator | a * b = 200 |
| / Division | Divides left hand operand by right hand operand | b / a = 2 |
| % Modulus | Divides left hand operand by right hand operand and returns remainder | b % a = 0 |
| ** Exponent | Performs exponential (power) calculation on operators | a**b =10 to the power 20 |
| // | Floor Division - The division of operands where the result is the quotient in which the digits after the decimal point are removed. | 9//2 = 4 and 9.0//2.0 = 4.0 |

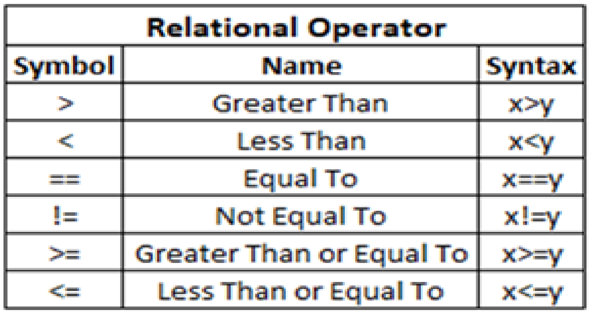
## Modulo Operator

The modulo operator (%) evaluates the remainder of the operation when the operand on the left is divided by the operand on the right of the modulo operator.

Example

```
15 % 7 = 1
13 % 7 = 6
```

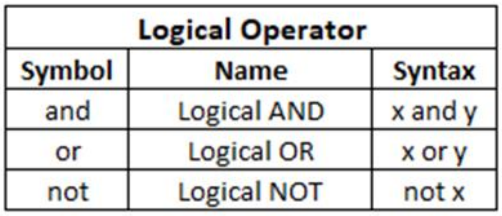
**Relational Operators:** Relational operators compare the operands. They evaluate to either True or False.



| Relational Operator | | |
|---|---|---|
| **Symbol** | **Name** | **Syntax** |
| > | Greater Than | x>y |
| < | Less Than | x<y |
| == | Equal To | x==y |
| != | Not Equal To | x!=y |
| >= | Greater Than or Equal To | x>=y |
| <= | Less Than or Equal To | x<=y |

## Logical operators:

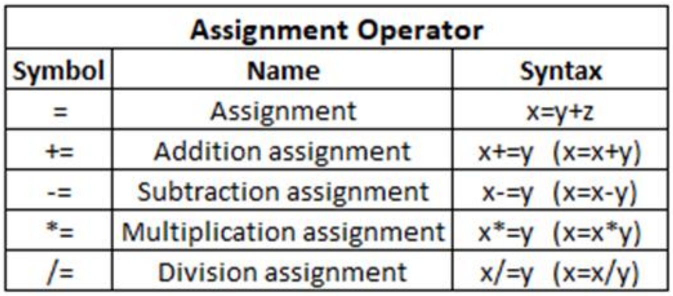
Logical operators combine logical values of true or false into a logical expression. They evaluate to either True or False. There are three basic types of logical operators: 'NOT', 'AND', and 'OR'.



| Logical Operator | | |
|---|---|---|
| **Symbol** | **Name** | **Syntax** |
| and | Logical AND | x and y |
| or | Logical OR | x or y |
| not | Logical NOT | not x |

## Assignment operators:

Variables are assigned values using assignment operators.



| Assignment Operator | | |
|---|---|---|
| **Symbol** | **Name** | **Syntax** |
| = | Assignment | x=y+z |
| += | Addition assignment | x+=y (x=x+y) |
| -= | Subtraction assignment | x-=y (x=x-y) |
| *= | Multiplication assignment | x*=y (x=x*y) |
| /= | Division assignment | x/=y (x=x/y) |

**Example :**

```
sum = 5 + x
result += 5
```

**Here, the sum variable is assigned the sum of 5 and x. And, the result variable is assigned the sum of result and 5.**

## Identity Operators

The identity operators in Python are **"IS"** and **"IS NOT"**. They check whether the two objects are of the same data type and share the same memory address.
**Example :**

```
x = 5
y = 5
if x is y:
     print("x and y are the same object")
else:
     print("x and y are different object")
```

Output will display **x and y are the same object** because both of them reference the same memory location.

## Membership Operators

The membership operators in Python are '**IN**' and '**NOT IN**'. They check whether the operand on the left side of the operator is a member of a sequence (such as a list or a string) on the right side of the operator.

**Example :**

```
x = ['Red', 2, 'Green']
if 'Red' in x:
     print("Found")
else:
     print("Not Found")
```

Output will display **Found** because the item 'Red' is a member of the List x.

## Operator Precedence

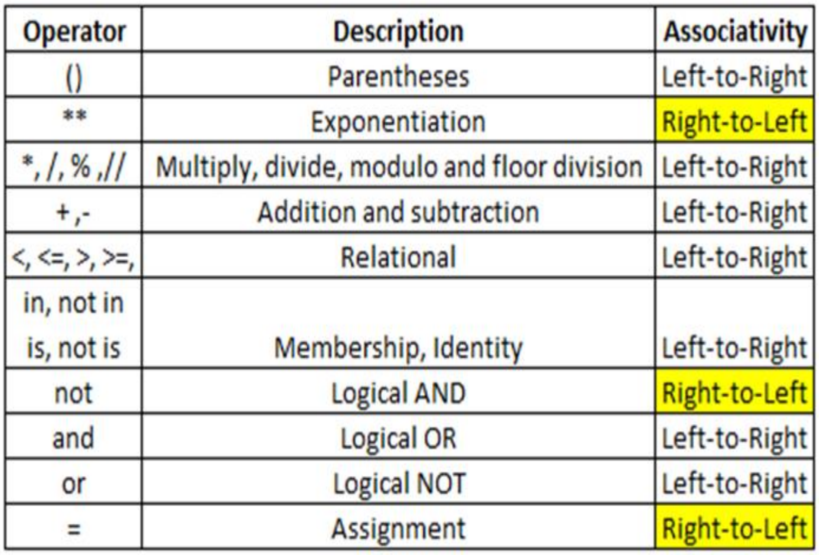
Operator precedence determines the priority of operators in an expression. The precedence of various operators is shown in the Precedence Table.

For example, x = 7 + 3 * 2;  will result in 13, not 20, because operator * has higher precedence than +. The expression is evaluated as 7 + 6.

## Associativity of Python Operators

Associativity refers to the order in which operators of the same precedence are evaluated. The associativity of an operator may be **left-to-right** or **right-to-left**.

## Precedence Table



| Operator | Description | Associativity |
|---|---|---|
| () | Parentheses | Left-to-Right |
| ** | Exponentiation | Right-to-Left |
| *, /, % ,// | Multiply, divide, modulo and floor division | Left-to-Right |
| + ,- | Addition and subtraction | Left-to-Right |
| <, <=, >, >=, | Relational | Left-to-Right |
| in, not in<br>is, not is | Membership, Identity | Left-to-Right |
| not | Logical AND | Right-to-Left |
| and | Logical OR | Left-to-Right |
| or | Logical NOT | Left-to-Right |
| = | Assignment | Right-to-Left |

## Evaluation of Expression

**1. Evaluate the expression 50 + 20 * 30**

Evaluation:

= 50 + (20 * 30) #precedence of * is more than that of +

= 50 + 600

= 650

**2. Evaluate the expression 100 - 20 + 50**

Evaluation:

The two operators (–) and (+) have equal precedence and the associativity is from left to right so the left operator (i.e. -) will be evaluated first.

= (100 – 20) + 50

= 80 + 50

= 130

**3. Evaluate the expression 9 + 3 ** 2 * 4 // 3**

Evaluation:

= 9 + (3 ** 3) * 4 // 3

= 9 + 27 * 4 // 3 (* and // has left to right associativity)
= 9 + 108 // 3
= 9 + 36
= 45

## Type conversion

It is the process of converting the value from one data type into another. Python supports two ways of Type conversion:

- Implicit conversion
- Explicit conversion

## Implicit conversion

This type of conversion is performed by Python Interpreter automatically without the user's intervention.
**Example:**

```
num1 = 20                     # num1 is integer
num2 = 30.5                   # num2 is  float
sum1  =  num1  +  num2  #  sum1  will  use  float  to  avoid
                             # loss of fractional part during addition
print(sum1)
print(type(sum1))

Output:
50.5
<class 'float'>
```

## Explicit Conversion:

This type of conversion is performed by the user manually. It is also known as type-casting. Explicit type-casting is performed using functions such as int( ), float( ), str( ) etc.
Syntax : new_data_type (expression)
**Example**

```
num1 = input("Enter a number : ")   # takes a string input by default
var1 = int(num1)          #converts string to integer
var1 = var1 * 3
print(var)
```

**Output**

```
Enter a number : 2
6
```

## Input and Output in Python

### Data Input:

input( ) function is used for getting input from the user. It returns the input as a String data type by default. Explicit type casting is required to convert the input string into any other data type if required.

### Data Output:

print( ) function is used for displaying output on the screen.

**Example-1**

```
num1 = int(input("Enter a Number : "))  # type casting the input from
String to Int
print("The Number is : ", num1)
```

**Example-2**

```
num = int(input("Enter a Number : "))
num_cube = num*num*num
print("The cube is : ",num_cube)
```

## Errors

An error is a problem that occurs in a program. Error sometimes halt the program execution, or produce unexpected results, and cause the program to behave abnormally.

- Syntax error
- Logical error
- Runtime error

### Syntax Error

Syntax are the rules for framing statements in a programming language. Any violations in the rules while writing a program are known as Syntax Errors. They prevent the code from executing and are detected by the Python interpreter before the execution of the program begins.
Some of the Common Syntax Errors are

- Parenthesis Mismatch
- Misspelled keyword
- Incorrect Indentation

## Logical Error

Logical errors occur when the code runs without any errors, but the output is not as expected. Logical errors are caused by a problem in the logic of the code.
Example : Average = mark_1 + mark_2 / 2   #  incorrect calculation of average marks
Corrected Code : Average = (mark_1  + mark_2 ) / 2

## Runtime Error

A runtime error causes abnormal termination of the program during the execution. Runtime error occurs when the statement is correct syntactically, but the interpreter cannot execute it.

**Example: 'division by zero'**

```
num1 = 5.0
num2 = int(input("num2 = "))  #if the user inputs zero, a runtime error
will occur
print(num1/num2)
```