REVISION OF PYTHON PROGRAMMING - I

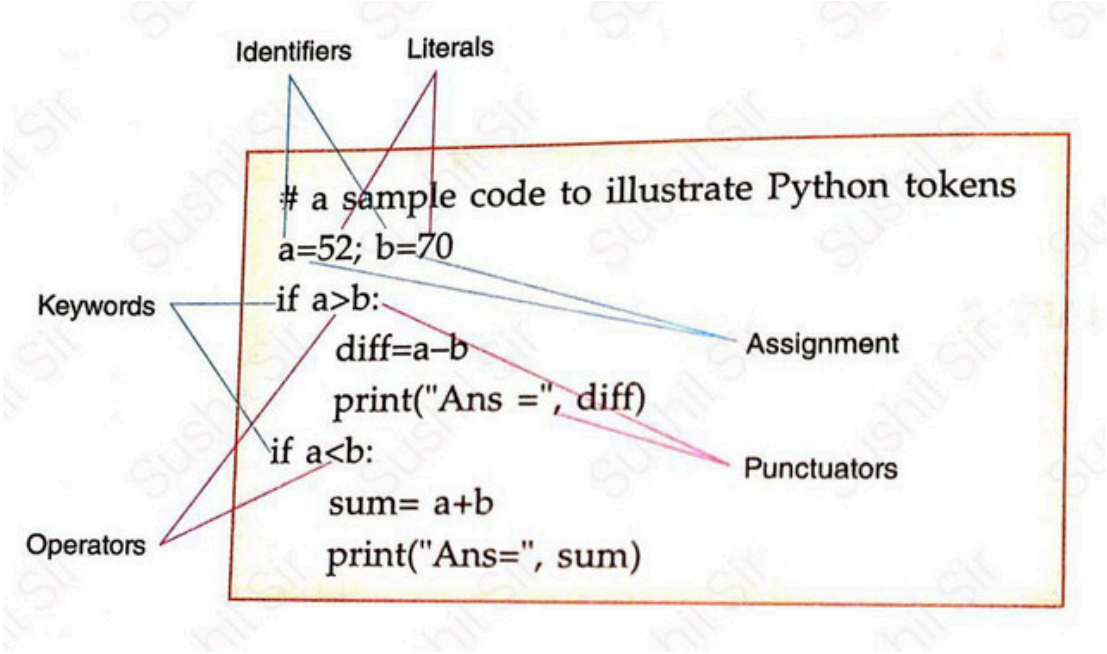
Introduction

- Python language developed by **Guido van Rossum**.
- It is a general purpose interpreter based high-level programming language that also supoorts Object-Oriented Programming concept.
- Python is a platform independent language (i.e., the same code can run on any operating system).
- It is one of the easiest programming language for the beginners.

> **Before introducing new concepts in Python Programming, let us have a quick revision of the topics you have already gone through in the previous class**

TOKENS

- A python statement is composed of various components, Each component of a programming statement is referred to as a Token.
- A token is the smallest individual unit in a python program.
- All statements and instructions in a program are built with tokens.
- The various types of tokens available in Python are:
  (a) Literals
  (b) Identifiers
  (c) Assignment
  (d) Punctuators
  (e) Operators
  (f) Keywords



Literals (Constants)

- The fixed values that remain unchanged throughtout the execution of a program are known as a literals or constants.
- Literals can further be categorised in following ways:

  (i) **Integer Literals**: Integer literals are the whole numbers written without using a decimal point.
  (a) **Decimal Numbers** -- (0-9) (e.g., 16, -29, 28, 245)
  (b) **Octal Numbers** -- (0-7) (e.g., 0o12, 0o37)
  (c) **Hexadecimal Numbers** -- (0-9 A-F) (e.g., 0x12, 0x71, 0x641)

  (ii) **Real Literals**: They are also called floating-point constants. In this type of numbers, the position of the decimal point is not fixed. It means, they may appear after any digit of the number. For example: 87.9345, 0.056454, -34.56494, 5.0E-02

  (iii) **Complex Literals**: These numbers are formed with the combination of a real and an imaginary number. For example: 4 +0j, 5.3 + 3j, 7+2j, 0 + 4j

  (iv) **String Literals**: They cantain the sequence of letters (uppercase, lowercase), digits, special symbols etc. enclosed within single quotes, double quotes or triple quotes. For example: 'K', '@', 'Python', '"""Computer"""'

  (v) **Boolean Literals**: They are the special constants that represent only two values i.e., **True** and **False** and can be used in Python code to check whether a given condition is satisfied or not.

(vi) **None Literal**: It is also a special literal and is used to indicate the absence of value,

In [49]:
```
## Literals
### Integer Literals
x = 12  # Decimal
y = 0o7 # Octal
z = 0xA # hexadecimal

print(x,y,z)
```

```
12 7 10
```

In [50]:
```
### Real Literal
a  = 87.95
b = 10.5E5
c = -34.56495

print(a,b,c)
```

```
87.95 1050000.0 -34.56495
```

In [51]:
```
### Complex Literal
p = 7 + 6j
q = -7j

print(p,q)
```

```
(7+6j) (-0-7j)
```

In [52]:
```
### String Literal
a = 'K'
b = "Computer"

print(a,b)
```

```
K Computer
```

In [53]:
```
### Boolean Literal
a = True
b = False

print(a,b)
```

```
True False
```

In [54]:
```
### None Literal
z = None
print(z)
```

```
None
```

Identifiers

- An identifier is a token to identify a block which may be a **function name**, **class name** or **variable name**.
- **Variable**: It is a named memory location that contains a value.
- When values are assigned to variables, the interpreter understand the data type (by default) and perform the tasks.

- Rules for naming a variable:
  --> Variable name should contain only letters, digits and underscore.
  --> Variable name must not begin with a digit.
  --> It should not contain any space in between the characters.
  --> The variable name should not be any reserved word or keyword.

Valid Variable Names: abc_12, abc12, _abc12, a_b_c_12
Invalid Variable Names: 12abc, 12_abc, if, print, ab 12,

Assignment

- An assignment statement is a fundamental construct that is used to set or reset the value in the memory location referred with a variable name.
- The value is set or reset in the variable by using an equal to (**=**) sign, also called an **assignment operator**.
- **Syntax**

  > Variable = Constant or Expression

- **For Example**

```
p = 91; q = 74.58; ch = 'A'
a = b = c = 99
x = b*b - 4*a*c
```

In [55]:
```
a=10; b = 12; c = 20
print(a,b,c)
```

```
10 12 20
```

In [56]:
```
a,b,c = 1,2,3
print(a)
print(b)
```

```
1
2
```

Punctuators

- The punctuators are the symbols that implement grammatical structure of a syntax.
- Some of the punctuators: Comma(**,**) , Semicolon (**;**), Colon(**:**), Parentheses **()**, Curly Brackets **{}**, Square Brackets **[]**

Operators

- Operators are the symbols or signs used to perform various operations in Python Programming.
- **Arithmetical Operators:** +, -, /, , %, //, *
- **Logical Operators:** and, or , not etc.,

- **Relational Operators:** <,>,==,!=,<= etc.,

Note: Other operators are discussed later in the notes.

Keywords

- Keywords are the reserved words which carry special meaning for the language translator (interpreter).
- These words can't be used as variable names in the program.
- Some of the common example of keywords are:

```
False  class  else  return  None  continue  for  not
True  def  if  or  and  del  import  pass  break
elif  is  while
```

DATA TYPES

- **Numeric Types**: Integer, Float, Complex
- **Non-Numeric**: Boolean
- **Sequence**: String, List, Tuple
- **Mapping**: Dictionary

type() is used to check the data type of a input value.

In [57]:
```
a = 12
print(a)
print(type(a))
```

```
12
<class 'int'>
```

In [58]:
```
b = 12.56
print(b)
print(type(b))
```

```
12.56
<class 'float'>
```

In [59]:
```
c = 12+5j
print(c)
print(type(c))
```

```
(12+5j)
<class 'complex'>
```

In [60]:
```
e = True
print(e)
print(type(e))
```

```
True
<class 'bool'>
```

In [61]:
```
d = "DPS"
print(d)
```

```
print(type(d))
print(d[0], d[-1], d[-3], d[2])
```

```
DPS
<class 'str'>
D S D S
```

In [62]:
```
f = [1,2,3,4,5]
print(f)
print(type(f))
print(f[0], f[-1], f[-3], f[4])
```

```
[1, 2, 3, 4, 5]
<class 'list'>
1 5 3 5
```

In [63]:
```
g = (1,2,3,4,5)
print(g)
print(type(g))
print(g[0], g[-1], g[2])
```

```
(1, 2, 3, 4, 5)
<class 'tuple'>
1 5 3
```

In [64]:
```
h = {'a':1, 'b':2, 'c':3}
print(h)
print(type(h))
print(h['a'])
print(h.keys())
print(h.values())
```

```
{'a': 1, 'b': 2, 'c': 3}
<class 'dict'>
1
dict_keys(['a', 'b', 'c'])
dict_values([1, 2, 3])
```

TYPE CONVERSION

- The process of converting the value from one data type to another is known as **Type Conversion**
- It is of two types:
  - **Implicit Type Conversion**: Results get automatically converted into the higher most data types available in the expression.
    Preference is **complex > float > int**
  - **Explicit Type Conversion**: When the data type of the result gets converted into a specific type based on user's request.
    int(), float(), str(), complex()

In [65]:
```
## IMPLICIT TYPE CONVERSION
a = 12; b  = 18.56; c = 14+5j
p1 = a + b  # int + float converts into float  (float > int)
print(p1)
```

```
30.56
```

In [66]:
```
## implicit type conversion
a = 12; b  = 18.56; c = 14+5j
p1 = b + c  # float + complex converts into complex  (complex > float > int)
print(p1)
```

```
(32.56+5j)
```

In [67]:
```
## EXPLICIT TYPE CONVERSION
a = 30; b = 12.30; c= 4
p = int(a + b)  ## Forcefully converted into int type using int() function
print(p)
```

```
42
```

Operators

- Arithmetic Operators **(+, -, *, /, //, %,** , +=, -=, %=, **=)**
- Relational Operators **(<, >, <=, >=, ==, !=)**
- Logical Operators **(and, or, not)**
- Identity Operators **(is, is not)**
- Membership Operators **(in, not in)**
- Bitwise Operators **(&, |, ~, ^)**

Precedence of Operators

- () > ** > *,/,//,% > +,-
- not > and > or

In [68]:
```
5*2**3/2   # first ** will be solved then multiplication and then division
```

Out[68]:
```
20.0
```

In [69]:
```
3/2  # TRUE DIVISION
```

Out[69]:
```
1.5
```

In [70]:
```
3//2 ## FLOOR DIVISION  --> QOUTIENT ONLY
```

Out[70]:
```
1
```

In [71]:
```
3%2 ## MODULUS OPERATOR --> REMAINDER ONLY
```

Out[71]:
```
1
```

In [72]:
```
a,b,c,d = 18,12,7,2
print(a*b%c//d)
```

```
3
```

In [73]:
```
12>13 and 14 < 12 or not (2 > 3)
```

Out[73]:
```
True
```

MATHEMATICAL FUNCTIONS

- **max(a,b,c,d,e)** --> Used to find maximum of given numbers (returns both int or float)
- **min(a,b,c,d,e)** --> Used to find minimum of given numbers (returns both int or float)
- **pow(a,b)** --> a to the power b (returns both int and float depends on values)
- **round(n)** --> rounded the value of the given argument with the specified number of digits after decimal point, (returns float only)

In [74]:
```
max(2,3,4,5,6)
```

Out[74]: 6

In [75]:
```
min(1,2,3,4,5)
```

Out[75]: 1

In [76]:
```
pow(2,3) # 2**3
```

Out[76]: 8

In [77]:
```
pow(2,3,8) # (2**3)%8
```

Out[77]: 0

In [78]:
```
round(2.678, 2)  # till two decimal
```

Out[78]: 2.68

In [79]:
```
round(2.897) # no decimal point required (in this case it will return int as we
```

Out[79]: 3

Difference Between pow() and math.pow() function

- **pow()** can return values in **int or float** & **math.pow()** will return only in **float data type**
- pow() function can take **three arguments** & math.pow() can only take **two arguments**

```
pow(int, int) --> return type int
pow(int, float) --> return type float
pow(float, int) --> return type float
pow(float, float) --> return type float
```

> pow(a,b,c) means (a**b)%c

In [80]:
```
pow(2,4)
```

Out[80]: 16

In [81]:
```
pow(2,4,4) # 2**4 = 16, 16%4 = 0 (as remainder when 16 divided by 4)
```

Out[81]: 0

In [82]:
```
import math
math.pow(2,4)
```

Out[82]: 16.0

In [83]:
```
math.pow(2,4,5) # cannot take three arguments so error
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[83], line 1
----> 1 math.pow(2,4,5) # cannot take three arguments so error

TypeError: pow expected 2 arguments, got 3
```

MATHS LIBRARY

In [84]:
```
import math

## from math import sqrt  <-- also correct

## square root

## it always return float type

s = math.sqrt(9)
print(s)

r = math.sqrt(240.25)
print(r)
```

```
3.0
15.5
```

In [85]:
```
print(math.sqrt(25))
```

```
5.0
```

In [86]:
```
## ceil function
### it returns higher integer between two integer values
### always return int type

c = math.ceil(6.5)
print(c)

d = math.ceil(-7.5)
print(d)
```

```
7
-7
```

In [87]:
```
## floor function
### it returns lower integer between two integer values
### always return int type
```

```
e = math.floor(6.5)
print(e)

f = math.floor(-7.5)
print(f)
```

```
6
-8
```

In [88]:
```
## fabs
### it returns absolute value
### always return float type

math.fabs(-7)
```

Out[88]:
```
7.0
```

In [89]:
```
math.fabs(-99.99)
```

Out[89]:
```
99.99
```

Trigonometric Functions

In [90]:
```
import math

a = 90
x = (22/(7*180))*a
v = math.sin(x)  # sin() function returns the sine of an angle (entered in radia
print(round(v,2))
```

```
1.0
```

In [91]:
```
print(round(math.cos(x), 2))
print(round(math.tan(x), 2))
```

```
-0.0
-1581.67
```

In [92]:
```
### INVERSE FUNCTION
import math
b = 0.5  # values must be in between -1 and 1
k = math.asin(b)  # returns the angle in radian
d = (k*7*180)/22  # convert in angle from radian to degree
print(round(d))
```

```
30
```

Random Function

- random() function is used to return a number randomly between the range of two numbers.

(i) random()

This function will display a float number randomly between **0.0 and 1.0 (0.0 <= n <= 1.0)**

In [94]:
```
import random
# from random import random

a = random.random()
print(a)
```

```
0.432808234652426
```

(ii) random.randint()

- This function will return a random number between the specified range of integer numbers.
- It will generate a random integer between **1 and 100 (both inclusive)**

In [95]:
```
import random

r = random.randint(1, 100)
print(r)
```

```
96
```

(iii) random.randrange()

- This function will return a randomly selected number from the range specified by the start,stop and step arguments.
- By default the start value is 0, and the step value is 1

In [96]:
```
import random

t = random.randrange(2,100,3)
print(t)
```

```
53
```

Statistical Function

In [97]:
```
import statistics

# from statistics import mean, median, mode
data = [1,2,3,4,5]
avg_l = statistics.mean(data)
avg_t = statistics.mean((1,2,3,4,5))

print(avg_l, avg_t)
```

```
3 3
```

In [98]:
```
import statistics

median_l = statistics.median([1,2,3,4,5,6])
median_t = statistics.median((1,2,3,4,5))
```

```
print(median_l, median_t)
```

```
3.5 3
```

In [99]:
```
import statistics

mode_l = statistics.mode([1,2,3,4,5,5,5,5])
mode_t = statistics.mode((1,2.0,3.5,4.0,5,6.0,6.0,6.0,6.0))

print(mode_l, mode_t)
```

```
5 6.0
```

INPUT STATEMENT

In [100…
```
n = input("Enter a string: ")  # Takes String Input
n1 = int(input("Enter a number: "))  # Takes Int Input
n2 = float(input("Enter a floating number: ")) # Takes a decimal number

print(n,n1,n2)
```

```
Enter a string: DPS
Enter a number: 12
Enter a floating number: 12.45
DPS 12 12.45
```