Imp

⇒ Python :-

Python is a computer programming language used to build website and software, automated tasks and conduct data analysis.

"Python is an interpreted, object-oriented, high-level programming language with dynamic semantices developed by "Guido van Rossum"."

⇒ Python Developed by :-

It was developed by "Guido van Rossum" in 1990.

	Version	Release Date
*	Python 1.0	Jan, 1994
*	Python 1.5	Dec, 1997
*	Python 1.6	Sep, 2000
*	Python 2.0	Oct, 2000
*	Python 2.1	Apr, 2001
*	Python 3.8	Oct, 2019

(Unit-(Ist))

Python Programming

⇒ Parts of python programming Language:

(i) Python Interpreter / Shell :-

The python interactive console (also called the python interpreter or python shell) provides programmers with a quick way to execute commands and try out or ~~take~~ test code without creating file.

(ii) ⇒ Python Shell :-

Python shell gives you a command line interface you can use to specify commands directly to the python interpreter in an interactive manner.

Imp:

(iii) ⇒ Identifiers in ~~Ph~~ python :-

- Identifier is a name used to identify a variable, function, class, module, etc.
- The identifier is a combination of character, digits, and underscore.
- The Identifier should start with a character or underscore then use a digit.

- The character are A-Z or a-z, an underscore (_), and digit (0-9).
- We should not use special characters (#, @, $, %, !) in identifiers.

⇒ Example of valid identifiers :-

* Var1.
* _Var1.
* _1_Var.
* Var_1.

⇒ Example of invalid identifiers :-

* !var1
* 1Var
* 1_Var
* Var#1

M.Imp :-

(iv) Keywords :- Keywords are some predefined and reserved words in python that have special meanings.

- Keywords are used to define the syntax of the coding.
- The keywords cannot be used as an identifier, function, and variable name.
- All the keywords in python are written in lower case except True and false.
- There are 33 keywords in python 3.7.

(i) and ⇒ This is a logical operator it returns true if both the operands are true else return false.

(ii) Or ⇒ This is also a logical operator it returns true if anyone operand is true else return false.

(iii) not ⇒ This is again a logical operator it returns true if the operands is false else return false.

(iv) if ⇒ This is used to make a condition-al statement. ~~Elif is a condition statement the elif statement~~

(v) Elif ⇒ Elif is a condition statement used with an if statement the elif statement is executed if the previous conditions were not true.

(vi) Else ⇒ Else is used with if and elif conditional statement the else block is executed if the given condition is not true.

(vii) for ⇒ This is created for a loop.

(viii) while ⇒ This Keyword is used to create a while loop.

(ix) Break ⇒ This is used to terminate the loop.

(x) as ⇒ This is used to create an alternative.

(xi) def ⇒ It helps us to define functions.

(xii) lambda ⇒ It is used to idefine the anonymous function.

(xiii) Pass ⇒ this is a null statement which means it will do nothing.

(xiv) return ⇒ It will return a value and exit the function.

(xv) True ⇒ This is a boolean value.

(xvi) false ⇒ This is also a boolean value.

(xvii) try ⇒ It makes a try - except statement.

(xviii) with ⇒ The with keyword is used to

simplify exceptions handling.

(xix) assert :> This function is used for debugging purposes. Usually used to check the correctness of code.

(xx) Class :> It helps us to define a class.

(21) Continue :> It continues to the next iteration of a loop.

(22) del :> It is used deletes a reference to an object.

(23) except :> Used with exceptions, what to do when an exception occurs.

(24) finally :> finally is used with exception a block of code will be executed no matter if there is an exception or not.

(25) from :> The from is used to import specifice parts of any module.

(26) global :> This declares a global variable.

(27) in :> It is used to check if a

value is present in a list, tuple, etc or not.

(28) import :> This is used to import a module.

(29) is :> This is used to check if the two variable are equal or not.

(30) None :> This is a special constant used to denote a null value or avoid, o, any empty container (e.g empty list) do not computer to none.

(31) nonlocal :> It's declared a non-local variable.

(32) Raise :> This raises an exception.

(33) yield :> It's ends a function and returns a generator.

Imp

(v) Statement :> A statement is an instruction that the python interpreter can execute.

Type of statements in python :>

* Assignment statement.
* Looping statement.
* Conditional statement.

(vi) Expression :> Expression in python is a combination of operators and operands.

=> for example :> $x = x + 10$.

In this expression, the first 10 is added to the variable x. After the addition is performed, the result is assigned to the variable x.

Ex :>

```
x = 25          (# a statement)
x = x+10        # an expression).
Print (x)
```

Output :>

35.

M. Imp (vii) Variables :> Python variables is Containers which store values. Python is not "Statically typed".

we do not need to declare variable before using them or declare their type. A variable is created the moment we first assign a value to it. A python variable is a name given to a memory location. It is the basics unit of storage in a program.

Example :-

```
var = "Shivam"
Print (var)
```

Output :-

Shivam.

m.Imp

(viii) Operators :- Operators are special symbols that perform operations on variables and values.

⇒ Types of operators in python :- There are 6 types of operators in python programming language:-

(1) → Arithmetic operators.
(2) → Assignment operators.
(3) → Comparison operators.
(4) → Logical operators.
(5) → Bitwise operators.
(6) → Special Operators.

* (1) Arithmetic Operators :→

Arithmetic operators are used to perform mathematical operations like addition, subtraction, multiplication, etc.

Operator	Operation	Example
+	Addition	5+2 = 7
-	Subtraction	4-2 = 2
*	Multiplication	2*3=6
/	Division	4/2 = 2
%	Modulo	5%2 = 1
**	Power	4**2 = 16

Example :→

```
a = 7
b = 2
Print ("Sum:", a+b)
Print ("Subtraction:", a-b)
Print ("Multiplication:", a×b)
Print ("Division:", a/b)
Print ("Modulo:", a%b)
Print ("Power:", a**b)
```

Output :→

```
Sum : 9
Subtraction: 5
Multiplication: 14
Division: 3.5
modulo: 1
Power: 49
```

* (2) Assignment Operators :-

Assignment Operators are used to assign values to variables.

Operators	Name	Example.
=	Assignment Operators	a = 7
+=	Addition Assignment	a+= 1 a = a + 1
-=	Subtraction Assignment	a -= 3 a = a - 3.
x=	multiplication Assignment	a *= 4 a = a * 4.
/=	Division Assignment	a /= 3 a = a /3.
%=	Remainder Assignment	a %= 10 a = a % 10.
**=	Exponent Assignment	a **= 10. a = a ** 10.

Example :>

```
a = 10
b = 5
a += b
Print(a)
```

Output :> 15.

*(3) Comparison Operators :>

Comparison operators compare two values/variables and return a boolean result : True or false.

Example :>

```
a = 5
b = 2
Print (a > b)
```

Output :> True.

Output :> a == b = False
a != b = True, a > b = True
a < b = False
a >= b = True
a <= b = False

Operator	Meaning	Example.
==	is equal To	3==5 false
!=	Not Equal to	3!=5 True
>	Greater Than	3>5 false
<	Less Than	3<5 True
>=	Greater Than Or Equal to	3>=5 gives us false
<=	Less Than Or Equal to	3<=5 gives us True

Example:-

```
a=5
b=2
Print('a==b=', a==b)
Print('a!=b =', a!=b)
Print('a>b=', a>b)
Print('a<b =', a<b)
Print('a>=b=', a>=b)
Print('a<=b=', a<=b)
```

*(4) Logical Operators :-

Logical operators used to check whether an expression is 'True' or 'false'.

They are used in decision-making.

Operator	Example	Meaning
and	a and b	Logical AND: True only if both the operands are true.
or	a or b	Logical OR: True if at least one of the operands is True.
Not	not a	Logical NOT: True if the operands is false and vice-versa.

Example of Logical Operators :→

```
Print (True and True) # True
Print (True and false) # false
Print (True or false) # True
Print (Not True)      # false.
```

*(5) Bitwise operators :→

Bitwise operator act on operands as if they were strings of binary digits. They operate bit by bit, hence the name.

Ex :→ 2 is 10 and 7 is 111.

Operator	Meaning	Example.
&	Bitwise AND	x&y = 0 (00000000)
|	Bitwise OR	x|y = 14 (00001110)
~	Bitwise Not	~x = -11 (11110101)
^	Bitwise XOR	x^y = 14 (00001110)
>>	Bitwise right shift	x>>2 = 2 (00000010)
<<	Bitwise Left shift	x<<2 = 40 (00101000)

* (6) Special operators :-

Python language offers some special types of operators like the identity operator and the membership operator.

* • Identity Operators :-

In python, 'is' and 'is not' are used to check if two values are located on the same part of the memory. Two variables that are equal does not imply that they are identical.

Operator	Meaning	Example.
is	True if the operands are identical	x is True
is not	True if the operands are Not identical	x is not True

Example :>

```
x1 = 5
y1 = 5
x2 = 'Hello'
y2 = 'Hello'
x3 = [1, 2, 3]
y3 = [1, 2, 3]
Print (x1 is not y1)
Print (x2 is y2)
Print (x3 is y3)
```

Output :>

```
False
True
False.
```

- Membership Operators :>

In python, 'in' and 'not in' are the membership operator. They are used to test whether a value or variables is found in a sequance (string, list, tuple, set and dictionary).

In a dictionary we can only test for presence of key, not the value.

Operator	Meaning	Example
in	True if value /variable is ~~false~~ found in the sequance	5 not in x
not in	True if value/ variables is not found in the sequance	5 not in x

Example :-

```
massage = 'Hello world'
dict1 = {1 : 'a', 2 : 'b'}
Print ('H' in massage)
Print ('hello' not in massage)
Print (1 in dict1 )
Print ('a' in dict1)
```

Output :-

True
True
True
false

Precedence and Associativity

(XI) Precedence and Associativity of Operators in python :→

(i) **Precedence Operator** :→ This is used in an expression with more than one operator with different precedence to determine which operator to perform first.

Example :→ 10 + 20 * 30

addition operator ← Lower Precedence

higer precedence → multiplication operator

10 + 20 * 30

multiply will happen first as * has higher precedence.

10 + 600

Now addition will happen as + has lower precedence.

610

10 + 20 * 30 is calculated as 10 + (20 * 30) and not as (10 + 20) * 30.

Code :-

```
expr = 10 + 20 * 30
Print (expr)
```

Output :- 610

(ii) Associativity operator :-

If an expression contains two or more operators with the same precedence then operators Associativity is used to determine. It can eighter be Left to Right or from Right to Left.

Example :- '*' and '/' have the same precedence and their associativity is Left to Right, so the expression "100 / 10 * 10" is treated as "(100 / 10) * 10".

Code :-

```
Print (100 / 10 * 10)
Print ( 5 - 2 + 3 )
Print ( 5 - (2+3))
Print ( 2 ** 3 ** 2)
```

Output :-

100
6
0
512

Operator Associativity

division operator ← high associativity

lower associativity → multiplication operator

100 / 10 * 10

divide will happen first as / has higher associativity

10 * 10

100

Now multiply will happen as * has lower associativity.

Data type

m. Imp

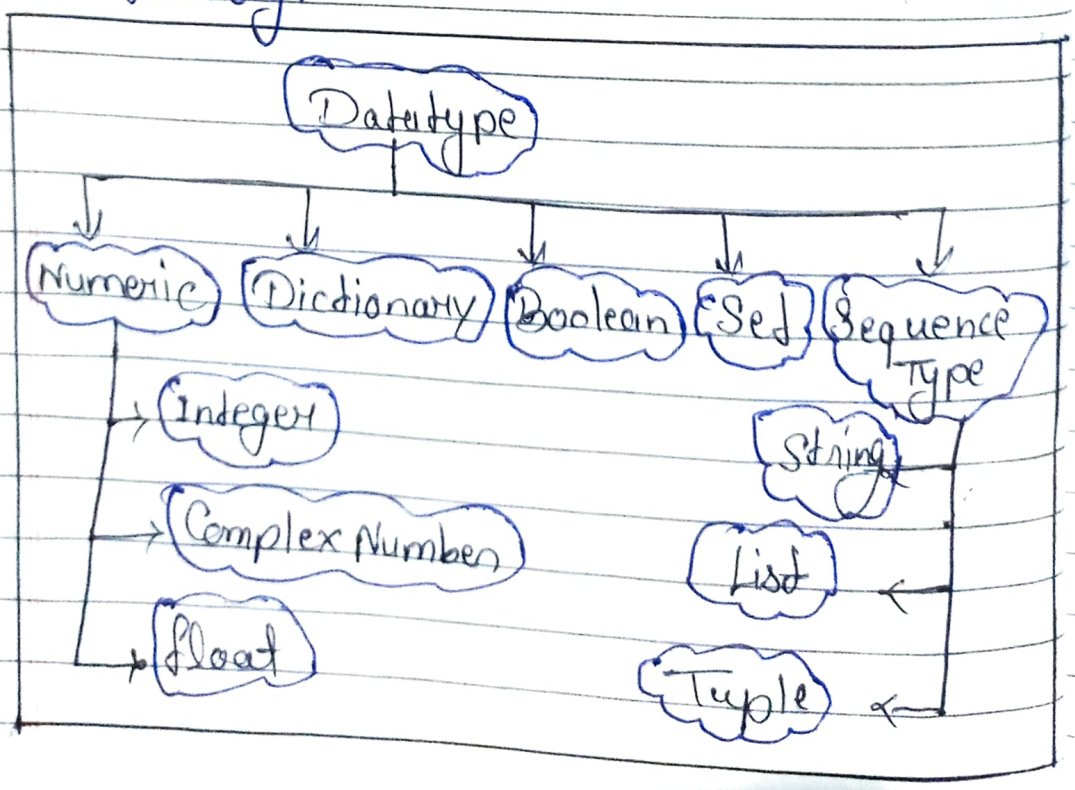
(X) Data type :-

Data type are the classification or categorization of data items. It represents the kind of value that tells what operations can be performed on a particular data.

"Data types are actually classes and variables are instance (object) of these classes."

⇒ Standard or Build-in Data type :-

(i) Numeric.
(ii) Sequence Type.
(iii) Boolean.
(iv) Set.
(v) Dictionary.



Numeric (Integer, float, Complex)

M. Imp

* (1) **Numeric Data type** :> In python, Numeric data type represend the data which has numeric value. Numeric value can be integer, float, and complex number.

* • **Integers** :> This value is represented by int class. It contains positive or negative whole numbers. In python there is no limit to how long an integer value can be.

* • **float** :> This value is represented by float class. It is a real number with floating point representation. It is specified by a decimal point. Optionally, the character e or E followed by a positive or negative integer may be append to specify scientific notation.

* • **Complex Numbers** :> Complex number is represented by complex class. It is specified as (real part) + (imaginary part)j.
Ex: 2+3j.

Program :-

```
a = 5
Print ("Type of a :" type(a))
b = 5.0
Print ("\n Type of b :" type(b))
c = 2+4j.
Print ("\n Type of c :" type(c))
```

Output :-

```
Type of a : 'int'
Type of b : 'float'
Type of c : 'Complex'
```

m.Imp

* (ii) **Sequence Type :-** Sequance is the ordered Collection of similar or different data types. Sequances allows to store multiple values in an organized and efficiend fashion.

Several sequance types in python :-

- String.
- List.
- Tuple.

* (1). **String :-** Strings are arrays of bytes representing Unicode characters.