Chapter No2: Variables and Data Types/Operators

Variables:

Variable is a name that is used to refer to memory location. Python variable is also known as an identifier and used to hold value.

In python we don't need to specify the type of variable because python language is smart enough to get variable type.

Example:

```
price = 100
tax = 18
total-price = price + tax
print (total-price)
```

Output:

```
118
```

What we did here is that we asigned "100" a name i.e price which now became a variable and now in our program we will use price that has the location of "100" to use the value.

As we know Python is dynamically typed, we can change the value residing on the location of a variable.

Example:

```
x = 100
print(x)
x = "What's your name"
print(x)
```

Output:

01_Variable.ipynb

Note!!

Assignmen 2.1

Try to create 10 different value variables, then asign them to each other in such a way that all of them have same value. One variable must not be asigned twice.

See 01_Variab.ipynb

Operators In Python

Variables that we've discussed are used for only one thing, i.e for operations. To perform any kind of operation we need operators and python provide these operators.

These are standard symbol used for logical and arithimetic operations.

Types of Operators:

1. Arithimetic
2. Comparison
3. Logical
4. Bitwise
5. Assignment
6. Identity and Membership.

1 Arithimetic Operator:

These are used to perform basic mathematical operations.

Operator	Description	Syntax
+	Addition	x+y
-	Subtraction	x-y
*	Multiplication	x*y
/	float Division	x/y
//	floor Division	x//y
%	Modulus	x%y
**	Power	x**y

Assignment 2.2

(i) What is the difference between floor and float division.

(ii) Why are there 2 operators for division.

Examples: 02_Operator

2 Comparison Operators:

Relational operators are used for comparing values. It returns either "true" or "false".

Operator	Description	Syntax
>	Greater than	x>y
<	Less than	x<y
==	Equal to	x=y
!=	Not Equal to	x!=y

>=	Greater than or equal to
<=	Less than or equal to
Example	See in 02_ Operator.ipynb

3 Logical Operators :-

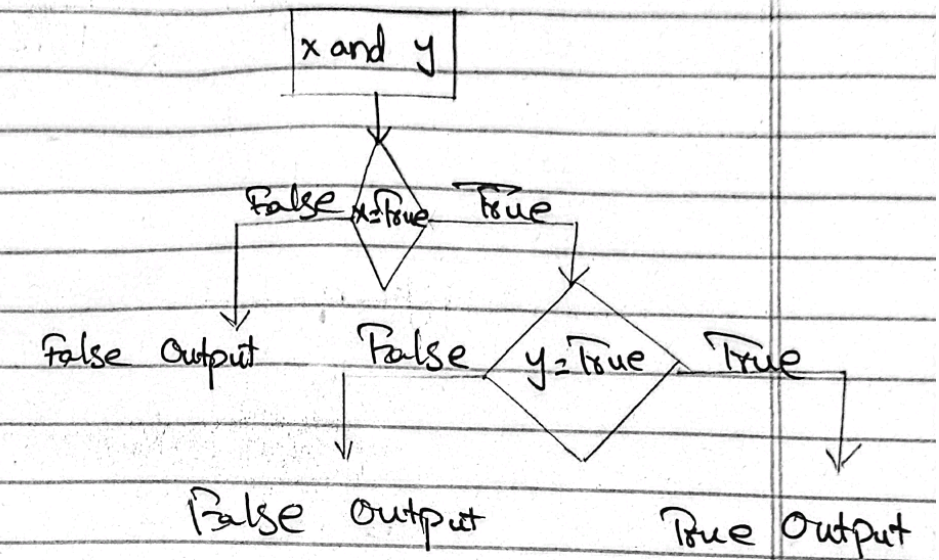
In python logical operators are used on conditional statements (either true or false).

Operator	Description
and	Logical and: True if both are true
or	Logical or: True if either is true
not	Logical not: True if operand is false

a Logical AND Operator:

AND returns true if both are true.

Flowchart:



Example:

```
a = 10
b = 10
c = -10
d = 0

if a>0 and b>0:
        print("Both are greater
        than (0)")
if a>0 and d>0:
        print("Both are greater
        than (0)")
else:
        print("~~Either~~ Both are not
                greater than (0)")
```

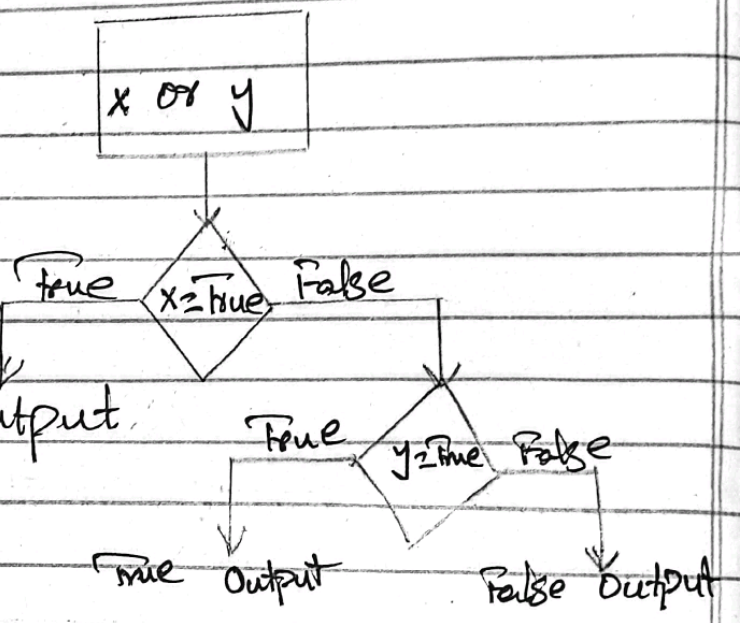
Output:

See 02_ Operators.ipynb

b Logical OR:

It returns true if either of operands is true.

Flowchart:



Example:

```
if a>0 or c>-10:
    print("Either of the
    number is greater than
    0")
else:
    print("Neither is
    greater than 0"
```

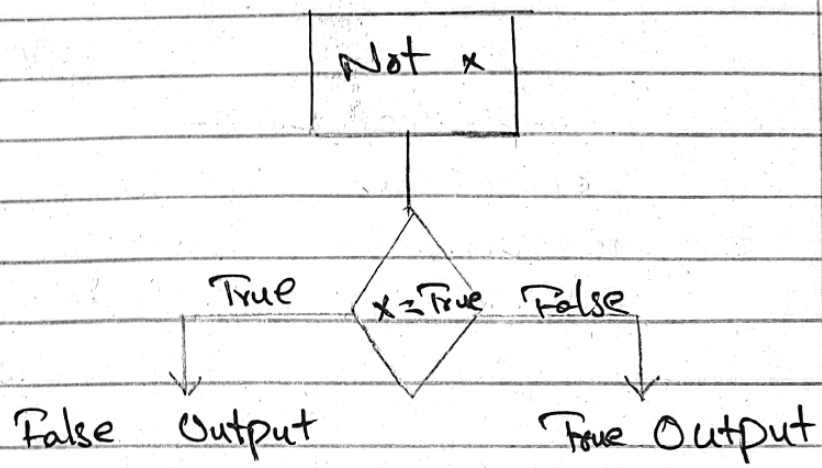
Output:

See in 02_Operator.

c Logical Not:

It operates with single boolean value. If boolean is "True" ;it returns False and vice-versa.

Flowchart:



Example:

```
if not a:                    of a
    print("Boolean value is True")
if not d:                    of d
    print("Boolean value is True")

if not (a%3==0 or a%5==0)
    print("10 is not divisible by
    either 3 or 5.")
```

```
    else:
        print("%d is divisible by either
    3 or 5.")
```

Output:

See in 02_Operators.ipynb

4 Bitwise Operators:

In python, bitwise operators are used to perform bitwise operations/calculations on integer. The integers are first converted into binary and then operations are performed on each bit or coressponding pair of bits. The result is given in decimal form.

Name	Operator	Description	Syntax
Bitwise AND	&	Result bit is 1, if both are 1. Otherwise 0.	x & y
Bitwise OR	|	Result bit 1, if any operand bit is 1; otherwise result bit 0	x | y
Bitwise NOT	~	Inverts individual bits	~x
Bitwise XOR	^	Result bit 1, if any of the operand bit is 1, but not both, otherwise 0.	x ^ y
Bitwise Right Shift	>>	Left operand's value is moved toward right by the number of bits specified by right operand.	x >> y
Bitwise Left Shift	<<	Does same operation as right shift but in opposite direction	x << y

Examples are given in 03_Bitwise operators.ipynb.

5. Assignment operators:

Assignment operators are used to assigning values to variables.

operator	Description	Syntax
=	Assign value of right side to left	x = y+z
+=	First add and then asign	x += y
-=	Subtract and assign	x -= y
*=	Multiply and assign	x *= y
/=	Divide and assign	x /= y
%=	Modulus and assign	x %= y
//=	Divide and assign	x //= y
**=	Exponent and assign	x **= y
&=	Bitwise And and asign	x &= y
|=	Bitwise OR and assign	x |= y
>>=	Bitwise right shift and assign	x >>= y
<<=	Bitwise left shift and assign	x <<= y

Examples

see 04_Assignment.ipynb

Note: Assignment is always from right to left.

6 Membership and Identity Operators

Membership:

Python offers two membership operators to check or validate if a value is a member or not. It tests for membership in sequences like, strings, lists or tuples.

(i) in: The 'in' operator is used to check if a character/substring/elements exists in a sequence or not. True if it's in and False if it's not in sequence.

Example:

```
list1 = [1,2, 16,17, 18, 26]
list2 = [32, 6, 19, 18,26,2, 35]
for item in list1:
    if item in list2:
        print("overlapping")
```

```
else:
    print ("Not Overlapping")
```

(ii) Not in: Evaluates true if it does not finds a variable in the specified sequence and false otherwise.

Example:

```
x = 13
y = 10
lists = [3, 6, 9, 13, 18, 20]
if x not in lists:
    print("True")
else:
    print ("False")
if y not in lists:
    print("True")
else:
    print ("False")
```

Identity Operators:

These are used to check if two objects are identical or not. Are they of same data type? Do they share same memory location? These two questions are asked by identity operators.

(i) is: Evaluates to True if the variables on either side point to same object and false otherwise.

Example:

```
x = 5
y = 5
print (x is y)
id (x)
id (y)
```

(ii) is not: Evaluates to True if the variables on either side does not point to same variable and false otherwise.

Example:

See this example and all others in 05_ID_And_Membership.ipynb.

Ternary Operators

Conditional expressions which will be discussed in upcoming chapters are sometimes called ternary operators.

- They have the lowest priority in python operations
- It was added to python version 2.5.
- It alows testing a condition in single line!

Note: Operator overloading after OOPs

Operator Functions after Functions. with Any/All.

Data Types

Data types are the classification or categorization of data items. It represents the kind of ~~data~~ value that tells what operations can be performed on a certain data. In Python everything is an object. Data types are actually classes and variables are instances (object) of these classes.

Following are built in data types in python.

1. Numeric
2. Sequence Type
3. Boolean
4. Set
5. Dictionary
6. Binary Types (memoryview, bytearray, byte

To find the type of variables we use type() function in python.

Example:

```
# Data Type string
    x = "Hello World"
# Data Type integer
    y = 50
print(type(x))
print(type(y))
```

Output:

```
<class 'str'>
<class 'int'>
```

1 Numeric:

Numeric data type in python represents three data types which are as follows

(i) Integer:
(ii) float:
(iii) Complex Number:

Integer - is represented by class int. It contains positive or negative whole numbers. In Python there is no limit to how long an integer can be.

Float is represented by class 'float'. It is a number with a floating point, specifed by decimal point.

Complex Numbers: are represented by complex class. It has (real part) + (Imaginary part)j.

Example: See 06-Numeric ---.ipynb

2 Sequence Type:

Sequence data type is the ordered collection of similar or different data types. These allow storing of multiple values an organized and efficient way. Following are sequence type in Python.

(i) Python String
(ii) Python List
(iii) Python Tuple

Note: All of these will be discussed in Data Structure chapter included in Python Basics

3 Boolean Data Type:

Data type with one of two built in values, True or False. It is denoted by class bool.

Note:

True and False with capital 'T' and 'F' are valid booleans otherwise python will throw an error.

Example:

```
print (type (True))
print (type (False))
print (type (toue))
```

Output:

See in 07_ Boolean Data Type.ipynb

4 Set Data Type:

In Python, a set is an unordered collection of data types that is iterable, mutable and has no duplicate elements.

Create a set in Python

Sets can be created by using built-in set() function with an iterable object or a sequence, by placing the sequence inside curly braces, separated by 'comma'.

Example:

```
set_1 = set ("I am a set")
print (type(set_1))
```

Output:

See in
08_Set. Data Type.ipynb.

Note: Sets will be further discussed in data structures chapter.

5 Dictionary Data type:

A dictionary in python is an unordered collection of data values, used to store data values like a map. A dictionary holds a key: value pair. It is provided in dictionary to make it more optimized. Each pair is separated by a comma and each key and value by colon ':'.

Creating:

A dictionary can be created by placing a sequence of elements within curly braces {}. Values can be of any data type and can be duplicated but keys must not be repeated and are immutable.

Example:

see 09_Dictionary.ipynb.

Note: Further Dictionaries will be discussed in Data Structures