**Python's features<learn any 5> include:**

• Easy-to-learn: Python has few keywords, simple structure, and a clearly defined syntax. This allows the student to pick up the language quickly.

•Python Supports functional and structured programming methods as well as OOP

• Easy-to-maintain: Python's source code is fairly easy-to-maintain.

• A broad standard library: Python's bulk of the library is very portable and cross-platform compatible on UNIX, Windows, and Macintosh.

• Interactive Mode: Python has support for an interactive mode which allows interactive testing and debugging of snippets of code. .

• Databases: Python provides interfaces to all major commercial databases.

Python can be dynamically typed which means we don’t need to declare data types in variables in advance like we do in C programming

## What is debugging?

- Programming is error-prone. For whimsical reasons, programming errors are called bugs and the process of tracking them down is called debugging.
- Three kinds of errors can occur in a program: syntax errors, runtime errors, and semantic errors.
- Syntax errors:
- Python can only execute a program if the syntax is correct; otherwise, the interpreter displays an error message.
- Syntax refers to the structure of a program and the rules about that structure.
- For example, parentheses have to come in matching pairs, so (1 + 2) is legal, but 8) is a syntax error.

- Runtime errors:

1. The second type of error is a **runtime error**.
2. A program with a runtime error is one that passed the interpreter’s syntax checks, and started to execute
3. However, during the execution of one of the statements in the program, an error occurred that caused the interpreter to stop executing the program and display an error message
4. Runtime errors are also called **exceptions** because they usually indicate that something exceptional (and bad) has happened.
5. Here are some examples of common runtime errorsr:
6. Misspelled or incorrectly capitalized variable and function names
7. Attempts to perform operations (such as math operations) on data of the wrong type (ex. attempting to subtract two variables that hold string values)

Semantic errors:

The third type of error is the semantic error.

- If there is a semantic error in your program, it will run successfully in the sense that the computer will not generate any error messages, but it will not do the right thing.
- It will do something else. The problem is that the program you wrote is not the program you wanted to write.
- The meaning of the program (its semantics) is wrong.
- Identifying semantic errors can be tricky because it requires you to work backward by looking at the output of the program and trying to figure out what it is doing

**Difference between Formal and Natural languages**

| **Natural  Language** | **Formal Language** |
|---|---|
| **Natural languages** are the languages that people speak, such as English, Spanish, Korean, and Mandarin Chinese. They were not designed by people (although people try to impose some order on them); they evolved naturally. | **Formal languages** are languages that are designed by people for specific applications. For example, the notation that mathematicians use is a formal language that is particularly good at denoting relationships among numbers and symbols. Chemists use a formal language to represent the chemical structure of molecules |
| Natural languages are full of ambiguity, which people deal with by using contextual clues and other information | Formal languages are designed to be nearly or completely unambiguous, which means that any statement has exactly one meaning, regardless of context. |
| In order to make up for ambiguity and reduce misunderstandings, natural languages employ lots of redundancy. As a result, they are often verbose (lengthy/wordy). | Formal languages are less redundant and more concise. |
| Natural languages are full of idiom and metaphor. If I say, “The penny dropped,” there is probably no penny and nothing dropping (this idiom means that someone realized something after a period of confusion). | Formal languages mean exactly what they say. |

Explain Python Data types:

A data type describes the characteristic value of a variable

Python has 6 standard data types

1. Numbers
2. String
3. List
4. Tuple
5. Set
6. Dictionary

1.Number

- ☞ In numbers there are mainly three types which include Integer, Float and Complex
- ☞ Number data types store numeric values. Number objects are created when you assign a value to them. For example −

```
☞ var1 = 1
☞ var2 = 10
```

- ☞ These 3 are defined in class python.In order to fid whuch class these variables belong to you can us the type function

```
a=5
print(a,is of type,type(a))

Output:
5 is of type<class ‘int’>
```

2. String

A string Is an ordered sequence of characters.

We can use single quotes or double quotes to represent a string

Strings are immutable which means once we declare a string we cant update the already declared string

```
Single=’Welcome’
Or
Multiple=”Welcome”
```

3.List

A list contains items separated by commas and enclosed within square brackets ([]).

To some extent, lists are similar to arrays in C.

One difference between them is that all the items belonging to a list can be of different data type

.A list is mutable which means we can modify the list.

```
List=[2,3,4,5.5,”Hi”]
Print(“List[2]= ”List[2])


Output:
List[2]=4
```

4.Tuple

A tuple is a sequence of Python objects defined using parentheses() and are separated by Commas

Tuples are immutable,which means once created they cannot be modified.

```
Tuple=(2,3,4,5.5,”Hi”)
Print(“Tuple[2]= ”Tuple[2])


Output:
Tuple[2]=4
```

5.Set

A set is an unordered collection of items separated by commas inside curly braces{}.

A set is an unordered collection of items. Every set element is unique (no duplicates) and must be immutable (cannot be changed).

However, a set itself is mutable. We can add or remove items from it

```
my_set = {1, 2, 3}
print(my_set)
my_set = {1.0, "Hello", (1, 2, 3)}
print(my_set)
```

Output:

```
{1, 2, 3}
{1.0, (1, 2, 3), 'Hello'}
```

## 6.Dictionary

Python Dictionary is used to store the data in a key-value pair format.

It is the mutable data-structure.

The dictionary is defined into element Keys and values.

o Keys must be a single element

o Value can be any type such as list, tuple, integer, etc.

In other words, we can say that a dictionary is the collection of key-value pairs where the value can be any Python object.

In contrast, the keys are the immutable Python object, i.e., Numbers, string, or tuple.

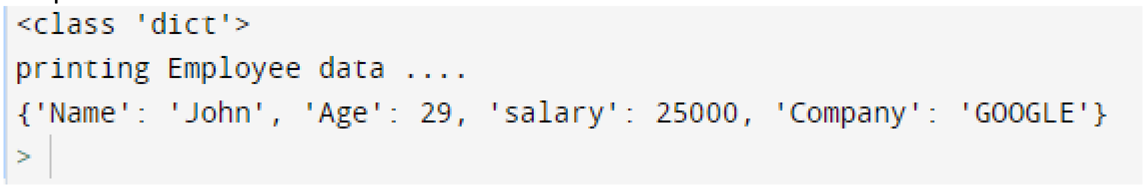
The dictionary can be created by using multiple key-value pairs enclosed with the curly brackets {}, and each key is separated from its value by the colon (:).

The syntax to define the dictionary is given below

Dict = {"Name": "Tom", "Age": 22}

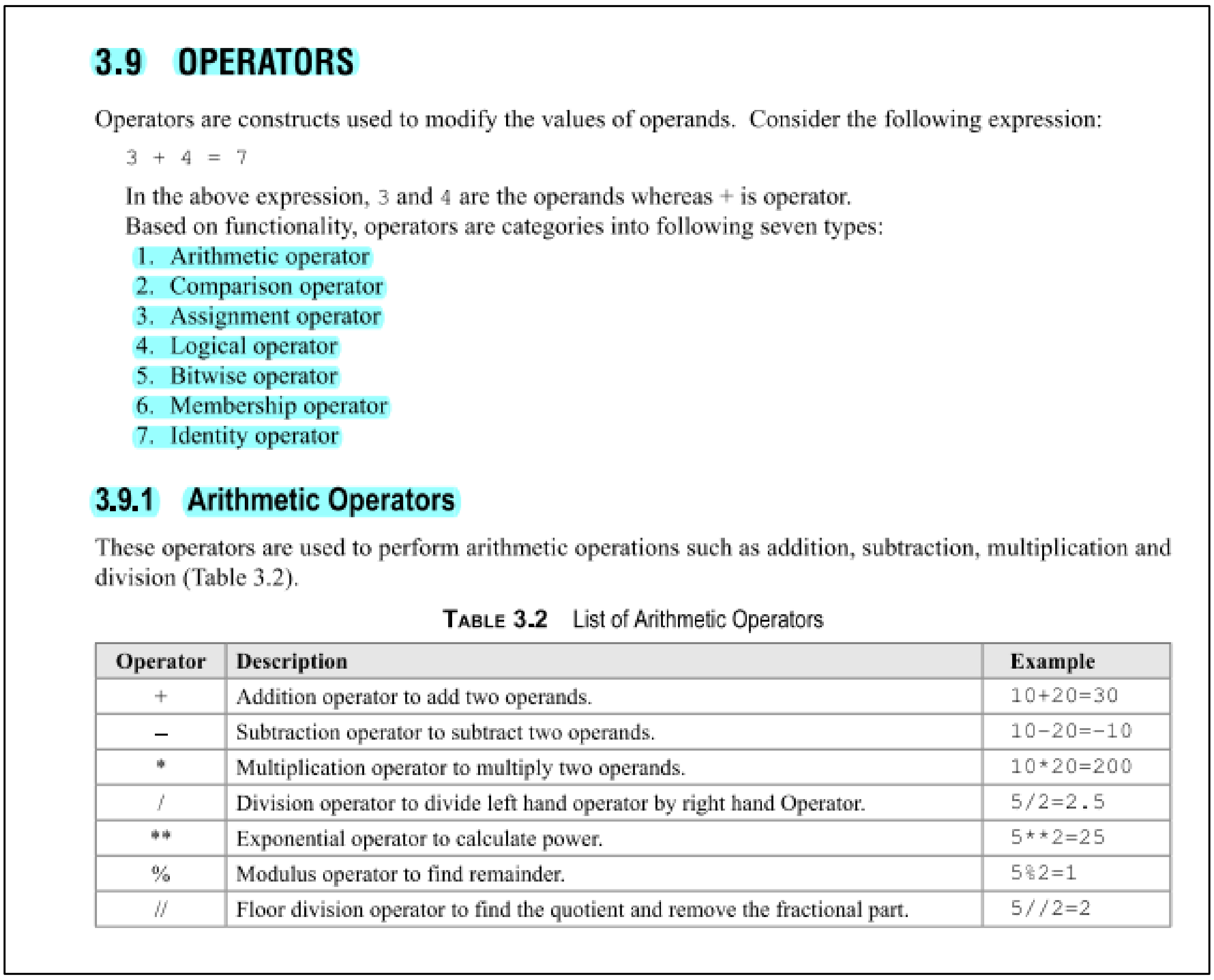
```
Employee = {"Name": "John", "Age": 29, "salary":25000,"Company":"GOOGLE"}

print(type(Employee))

print("printing Employee data .... ")

print(Employee)
```

Output:



```
<class 'dict'>
printing Employee data ....
{'Name': 'John', 'Age': 29, 'salary': 25000, 'Company': 'GOOGLE'}
>
```

Explain operators in python with examples



## 3.9 OPERATORS

Operators are constructs used to modify the values of operands. Consider the following expression:

```
3 + 4 = 7
```

In the above expression, 3 and 4 are the operands whereas + is operator.
Based on functionality, operators are categories into following seven types:

1. Arithmetic operator
2. Comparison operator
3. Assignment operator
4. Logical operator
5. Bitwise operator
6. Membership operator
7. Identity operator

### 3.9.1 Arithmetic Operators

These operators are used to perform arithmetic operations such as addition, subtraction, multiplication and division (Table 3.2).

**Table 3.2** List of Arithmetic Operators

| Operator | Description | Example |
|---|---|---|
| + | Addition operator to add two operands. | 10+20=30 |
| – | Subtraction operator to subtract two operands. | 10-20=-10 |
| * | Multiplication operator to multiply two operands. | 10*20=200 |
| / | Division operator to divide left hand operator by right hand Operator. | 5/2=2.5 |
| ** | Exponential operator to calculate power. | 5**2=25 |
| % | Modulus operator to find remainder. | 5%2=1 |
| // | Floor division operator to find the quotient and remove the fractional part. | 5//2=2 |

**Example**

```
>>> x = 10
>>> y = 12
>>> z = 0

>>> z = x + y
>>> print z
22                              #Ouput

>>> z = x - y
>>> print z
-2                              #Ouput

>>> z = x * y
>>> print z
120                             #Ouput

>>> z = x / y
>>> print z
0                               #Ouput

>>> z = x % y
>>> print z
10                              #Ouput

>>> z = x ** y
>>> print z
1000000000000                   #Ouput

>>> z = x // y
>>> print z
0                               #Ouput
```

### 3.9.2 Comparison Operators

These operators are used to compare values. Comparison operators are also called relational operators. The result of these operators is always a Boolean value, that is, either true or false. Table 3.3 provides a list of comparison operators.

**Table 3.3** List of Comparison Operators

| Operator | Description | Example |
|---|---|---|
| == | Operator to check whether two operands are equal. | 10 == 20, false |
| != or <> | Operator to check whether two operands are not equal. | 10 !=20, true |
| > | Operator to check whether first operand is greater than second operand. | 10 > 20, false |
| < | Operator to check whether first operand is smaller than second operand. | 10 < 20, true |
| >= | Operator to check whether first operand is greater than or equal to second operand. | 10 >= 20, false |
| <= | Operator to check whether first operand is smaller than or equal to second operand. | 10 <= 20, true |

### 3.9.3 Assignment Operators

This operator is used to store right side operand in the left side operand. Table 3.4 provides a list of assignment operators.

**Table 3.4** List of Assignment Operators

| Operator | Description | Example |
|---|---|---|
| = | Store right side operand in left side operand. | `a=b+c` |
| += | Add right side operand to left side operand and store the result in left side operand. | `a+=b or a=a+b` |
| – = | Subtract right side operand from left side operand and store the result in left side operand. | `a-=b or a=a-b` |
| * = | Multiply right side operand with left side operand and store the result in left side operand. | `a*=b or a=a*b` |
| / = | Divide left side operand by right side operand and store the result in left side operand. | `a/b or a=a/b` |
| % = | Find the modulus and store the remainder in left side operand. | `a%=b or a=a%b` |
| ** = | Find the exponential and store the result in left side operand. | `a**=b or a=a**b` |
| // = | Find the floor division and store the result in left side operand. | `a//=b or a=a// b` |

### 3.9.4 Bitwise Operators

These operators perform bit level operation on operands. Let us take two operands x = 10 and y = 12. In binary format this can be written as x = 1010 and y = 1100. Table 3.5 presents a list of bitwise operators.

**Table 3.5** List of Bitwise Operators

| Operator | Description | Example |
|---|---|---|
| & Bitwise AND | This operator performs AND operation between operands. Operator copies bit if it exists in both operands. | x & y results 1000 |
| \| Bitwise OR | This operator performs OR operation between operands. Operator copies bit if it exists in either operand. | x \| y results 1110 |
| ^ Bitwise XOR | This operator performs XOR operation between operands. Operator copies bit if it exists only in one operand. | x ^ y results 0110 |
| ~ bitwise inverse | This operator is a unary operator used to opposite the bits of operand. | ~ x results 0101 |
| << left shift | This operator is used to shift the bits towards left | x << 2 results 101000 |
| << right shift | This operator is used to shift the bits towards right | x >> 2 results 0010 |

**Example**

```
>>> x = 10               # 10 = 0000 1010
>>> y = 12               # 12 = 0000 1100
>>> z = 0

# Bitwise AND
>>> z = x & y
>>> print z
8                              # 8 = 0000 1000

# Bitwise OR
```

### 3.9.5 Logical Operators

These operators are used to check two or more conditions. The resultant of this operator is always a Boolean value. Here, x and y are two operands that store either true or false Boolean values. Table 3.6 presents a list of logical operators. Assume x is true and y is false.

TABLE 3.6 List of Logical Operators

| Operator | Description | Example |
|---|---|---|
| and logical AND | This operator performs AND operation between operands. When both operands are true, the resultant become true. | x and y results false |
| or logical OR | This operator performs OR operation between operands. When any operand is true, the resultant becomes true. | x or y results true |
| not logical NOT | This operator is used to reverse the operand state. | not x results false |

**Example**

```
>>> x = True
>>> y = False

>>> print (x and y)
```

,

```
False                          #Output

>>> print (x or y)
True                           #Output

>>> print (not x)
False                          #Output

>>> print (not y)
True                           #Output
```

### 3.9.6 Membership Operators

These operators are used to check an item or an element that is part of a string, a list or a tuple. A membership operator reduces the effort of searching an element in the list. Suppose, *x* stores a value 20 and *y* is the list containing items 10, 20, 30, and 40. Then, *x* is a part of the list *y* because the value 20 is in the list *y*. Table 3.7 gives a list of membership operators.

TABLE 3.7 List of Membership Operators

| Operator | Description | Example |
|---|---|---|
| in | Return true, if item is in list or in sequence. Return false, if item is not in list or in sequence. | x in y, results true |
| not in | Return false, if item is in list or in sequence. Return true, if item is not in list or in sequence. | x not in y, results false |

**Example**

```
>>> x = 10
>>> y = 12
>>> list = [21, 13, 10, 17]

>>> if (x in list):
        print "x is present in the list"
    else:
        print "x is not present in the list"

x is present in the list                    #Output

>>> if (y not in list):
        print "y is not present in the list"
    else:
        print "y is present in the list"

y is not present in the list                #Output
```

## 3.9.7 Identity Operators

These operators are used to check whether both operands are same or not. Suppose, x stores a value 20 and y stores a value 40. Then `x is y` returns `false` and `x not is y` returns `true`. Table 3.8 provides a list of identity operators

TABLE 3.8 List of Identity Operators

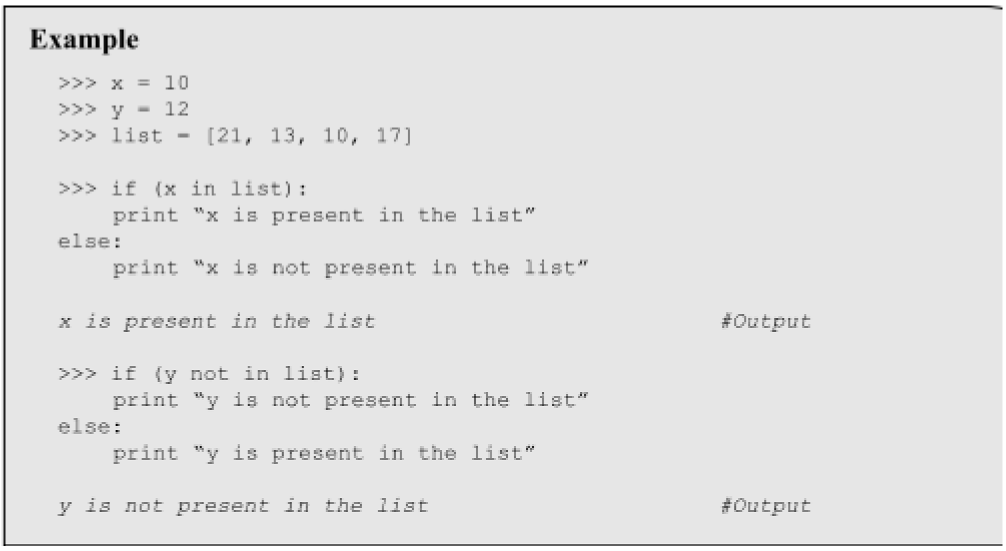
| Operator | Description | Example |
|---|---|---|
| is | Return true, if the operands are same. Return false, if the operands are not same. | x is y, results false |
| not is | Return false, if theoperands are same. Return true, if the operands are not same. | x not is y, results true |

**Example**

```
>>> x = 12
>>> y = 12

>>> if ( x is y):
            print "x is same as y"
    else:
            print "x is not same as y"

x is same as y                    #Output

>>> y = 10

>>> if ( x is not y):
            print "x is not same as y"
    else:
            print "x is same as y"

x is not same as y                #Output
```

**Write a program to use membership and and identity operarators**

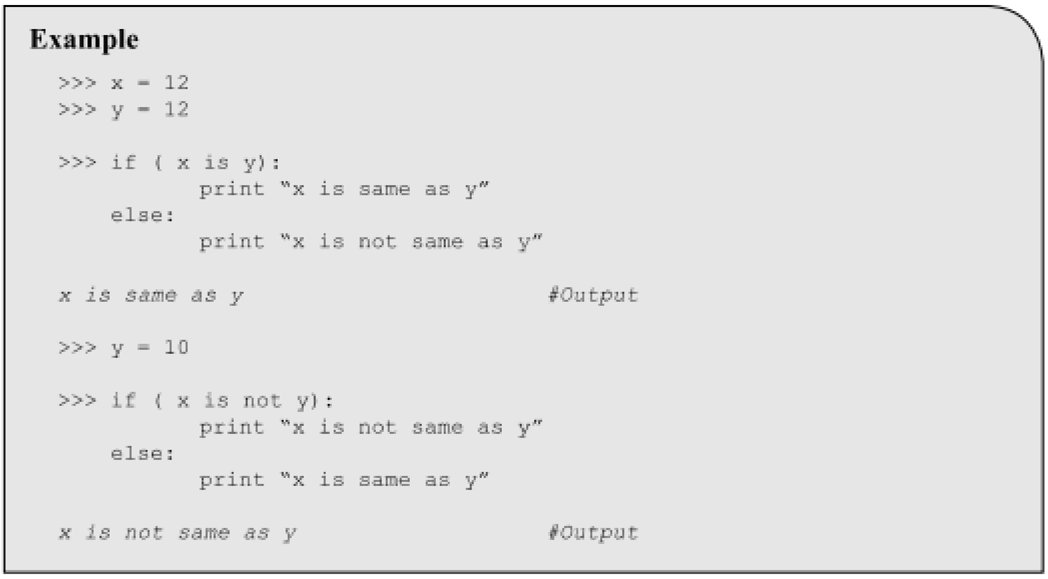
Using membership operators:1.in operator 2:not in operator



**Example**

```
>>> x = 10
>>> y = 12
>>> list = [21, 13, 10, 17]

>>> if (x in list):
    print "x is present in the list"
else:
    print "x is not present in the list"

x is present in the list                          #Output

>>> if (y not in list):
    print "y is not present in the list"
else:
    print "y is present in the list"

y is not present in the list                      #Output
```

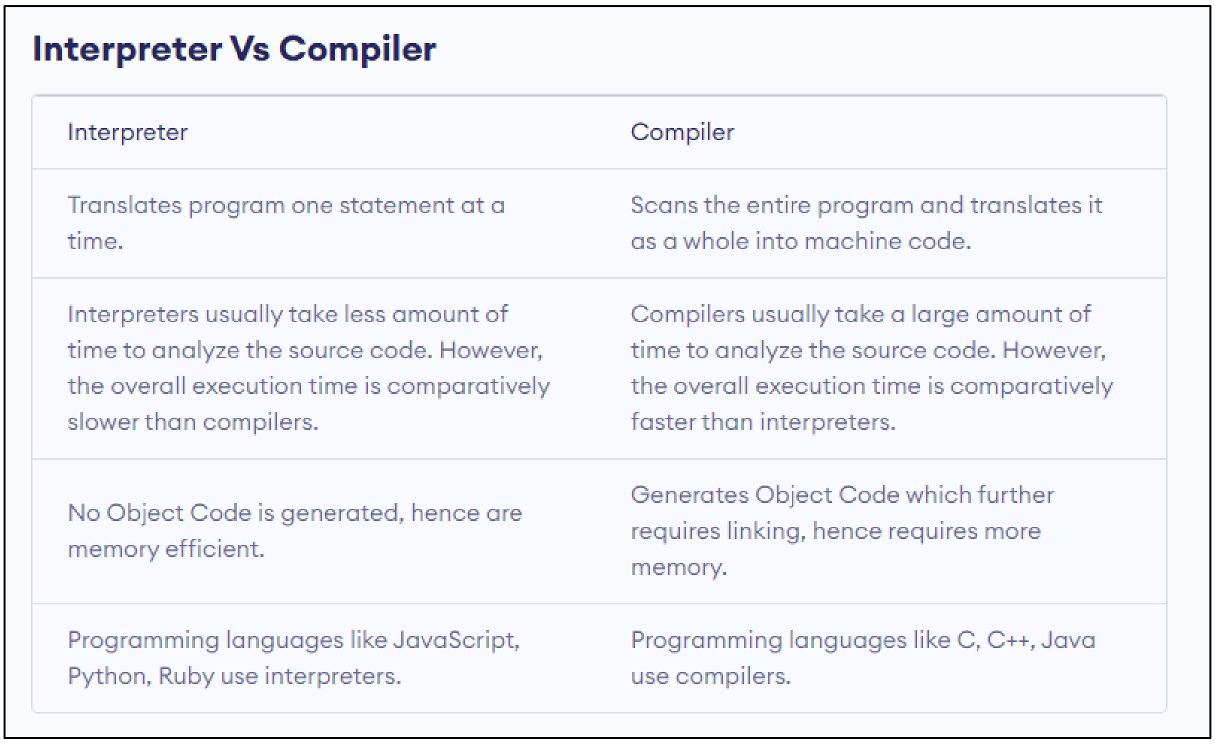
Using identity operator 1:is operator 2:is not operator



**Example**

```
>>> x = 12
>>> y = 12

>>> if ( x is y):
         print "x is same as y"
    else:
         print "x is not same as y"

x is same as y                   #Output

>>> y = 10

>>> if ( x is not y):
         print "x is not same as y"
    else:
         print "x is same as y"

x is not same as y               #Output
```

## Compiler and interpreter



### Interpreter Vs Compiler

| Interpreter | Compiler |
|---|---|
| Translates program one statement at a time. | Scans the entire program and translates it as a whole into machine code. |
| Interpreters usually take less amount of time to analyze the source code. However, the overall execution time is comparatively slower than compilers. | Compilers usually take a large amount of time to analyze the source code. However, the overall execution time is comparatively faster than interpreters. |
| No Object Code is generated, hence are memory efficient. | Generates Object Code which further requires linking, hence requires more memory. |
| Programming languages like JavaScript, Python, Ruby use interpreters. | Programming languages like C, C++, Java use compilers. |

## Conditional Statements in python

Conditional statements are also called decision-making statements. We use those statements while we want to execute a block of code when the given condition is true or false.

Type of condition statement in Python:

- If statement.
- If Else statement.
- Elif statement.

IF statement
In python if statement is used for decision making. It will run the body of the code only when the if statement is true
Syntax:

```
If condition:
        Body of if
```

## Example:

```python
a = 10
b = 20
if a<b:
  print("a is less than b")
```

### If else statement

If else is a conditional statement.The body of if statement executes if the condition in the if statement is true else if the condition is false the body of else statement is executed.

Syntax:

```python
if(condition):
   # if statement
else:
   # else statement
```

Program for if else statement,

```python
a = 10
b = 20
if a==b:
  print("a and b are equal")
else:
  print("a and b are not equal")
```

## if....elif....else statement

The elif is short form of else if. It allows to check for multiple expressions

If the condition in if is false then it checke the condition for elif block and so on

If all the conditions are false then the body of else

Is executed

The if block can have only one else block but it can help multiple elif blocks hence it allows to check multiple conditions.

## Example

```
if(condition):

    # if statement

elif(condition):

    # elif statement

else:

     # else statement

```

## Write a program to demonstrate the use of conditional statements

```
a = 10
b = 10
if a < b:
  print("a is greater than b")
elif a == b:
  print("a and b are equal")
else:
  print("b is greater than a")
```

## or the below program can also be used

```
#FIBONACCI SERIES O 1 1 2 3 5...

num=int(input("Enter the number of terms: "))

x=0

y=1

sum=0

if num<=0:

    print("Please enter a positive integer")

elif num==1:

    print("The fibonacci series upto ",num," terms is: ",x)

else:

    print("The fibonacci series upto ",num," terms is: ")

    while(sum<=num):

        print(sum)

        x=y

        y=sum

        sum=x+y
```

### Looping Statements

## Python for Loop

The for loop in Python is used to iterate over a sequence (list, tuple, string) or other iterable objects. Iterating over a sequence is called traversal.

### Syntax of for Loop

```
for val in sequence:
    loop body
```

Here, `val` is the variable that takes the value of the item inside the sequence on each iteration.
Loop continues until we reach the last item in the sequence. The body of for loop is separated from the rest of the code using indentation.

### Example : for loop in python

```
# Program to find the sum of all numbers stored in a list

# List of numbers
numbers = [6, 5, 3, 8, 4, 2, 5, 4, 11]

# variable to store the sum
sum = 0

# iterate over the list
for val in numbers:
    sum = sum+val

print("The sum is", sum)
```

## While loop in python

The while loop in Python is used to iterate over a block of code as long as the test expression (condition) is true.

We generally use this loop when we don't know the number of times to iterate beforehand.

### Syntax of while Loop in Python

```
while test_expression:

    Body of while
```

In the while loop, test expression is checked first. The body of the loop is entered only if the `test_expression` evaluates to `True`. After one iteration, the test expression is checked again. This process continues until the `test_expression` evaluates to `False`.

In Python, the body of the while loop is determined through indentation.

The body starts with indentation and the first unindented line marks the end.

Python interprets any non-zero value as `True`. `None` and `0` are interpreted as `False`.

Example of while loop

```python
n = int(input("Enter n: "))
sum = 0
i = 1

while i <= n:
    sum = sum + i
    i = i+1    # update counter

# print the sum
print("The sum is", sum)
```