Operator and operand:

Based on operands:

Unary operator

Binary operator

Unary operator:

single operand is present that operator is called as unary operator eg: -4

Binary operator:it contains two operands eg: 4+6 in this + is called binary operator

Based on operator:

Arithmatic operator

Relational operator

Membership operator

Boolean operator,

Relational operator, Membership operator,Boolean operator are also called as Boolean expression because the result of the expression is true or false

Arithmetic operators:

1.Negation

2.Addition

3.subtraction

4.multiplication

5.division

6.truncative div

7.modulus

8.Exponential

1.Negation:

It is the unary operatorit is use to negate the value

Eg: >>> -5

-5

2.Addition

>>> 5+4

9

3.subtraction

>>> 6-3

3

4.multiplication

>>> 3*3

9

5.division

>>> 9/3

3.0

>>> 9/2

4.5

6.truncative div

>>> 9//2

4

7.modulus

>>> 5%4

1

8.Exponential: 2 is the base and 5 is Exponetial

>>> 2**5

32

Relatinal operator:

EQUAL: ==

NOT EQUAL !=

LESS THEN <

GREATER THEN >

LESSTHEN OR EQUAL TO <=

GREATEERTHEN OR EQUAL TO

Membership operator:

In:

>>> 5 in [1,2,3,4,5,6]

True

Not in:

>>> 5 not in [1,2,3,4,5,6]

False

Boolean operator:

And:

```
>>> 5<6 and 4==4
True
>>> 5<6 and 4==5
False
```

Or:

```
>>> 5<6 or 4==5
True
>>> 5>6 or 4==5
False
```

Not:

```
>>> 10>5
True
>>> not (10>5)
False
```

operator precedence:

precedence and associativity:

2+5*4=28 but 22 is output

evaluating an expression which contains multiple operator. We need to follow priority. there is some operator they have highest priority. This case there is two operators + and multiplication. multiplication symbol has highest priority .so here multiplication symbol has highest priority then addition.

So 5*4=20 then add 2 that is 22

This ruling priority is called as precedence.

(2+5)*4=28

Some cases addition and subtraction is same precedence so we apply associative rule

There is two type of evaluation

1. Left to right evaluation
2. Right to left evaluation.

Almost all evaluation follows left to right evaluation.

But exponential evaluation follow right to left

Highest priority wise:

1.() parantasis symbol

2.**

3.unary operator

4.*,/,//.%

5.+,_

6.<<,>>

7.&

8.^ bitwise ex-or operator

9.| botwise or operator

10.relatinal and membership operator

11.not

12.and

(Assignment statement):

Why I want to copy any data?

Because I want to store original data or I change same data then I need a copy of that so assignment operator to copy the data

X=4

Y=X Here we are altering X .original value is Y. but here assignment operator is not copying the data. Y is the another name given the value for.

S it allows the original data and it allows modified data

X=X+1

We can't alter immutable objects. but reference can be changed and we may loose that data. To stop that we can use assignment operator.

```
>>> list1=[1,2,3,4]
>>> list2=list1
>>> list2
[1, 2, 3, 4]
>>> list2.append(10)
>>> list2
[1, 2, 3, 4, 10]
>>> list1
[1, 2, 3, 4, 10]
>>> id(list1)==id(list2)
True
```

List1= list2 so this is not copying purpose that why we are going to

1) Shallow purpose
2) Deep purpose

Shallow purpose: create a new object which store the reference of the original elements.

Shallow copy:

1) Built-in function list(), set(),dict()
2) Slicing operator
3) List comprehension method
4) Copy function for copy module

1)Built-in function list():

```
>>> list1=[1,2,3,4]
>>> list2=list(list1)
>>> list2
[1, 2, 3, 4]
>>> list2.append("a")
>>> list2
[1, 2, 3, 4, 'a']
>>> list1
[1, 2, 3, 4]
```

2)Slicing operator:

```
>>> list1=[1,2,3,4]
>>> list2=list1[:]
>>> list2[0]="amul"
>>> list2
['amul', 2, 3, 4]
>>> list1
[1, 2, 3, 4]
```

3)List comprehension method:

```
>>> list1=[1,2,3,4]
>>> list2=[x for x in list1]
>>> list2
[1, 2, 3, 4]
>>> list2[3]='a'
>>> list2
[1, 2, 3, 'a']
>>> list1
[1, 2, 3, 4]
```

4)Copy function for copy module:

```
>>>Import copy
>>> list1=[1,2,3,4]
>>> list2=copy.copy(list1)
>>> list2
[1, 2, 3, 4]
```

```
>>> list2.append("hellow")
>>> list2
[1, 2, 3, 4, 'hellow']
>>> list1
[1, 2, 3, 4]
```

Deep copy:

Creates a new object and recursively adds the copies of nested objects present in the original elements.

```
>>> import copy
>>> list1=[1,2,3,4]
>>> list2=copy.deepcopy(list1)
>>> list2
[1, 2, 3, 4]
>>> id(list1)==id(list2)
False
>>> list2.append(10)
>>> list2
[1, 2, 3, 4, 10]
>>> list1
[1, 2, 3, 4]
```

If we take nested list:

```
>>> import copy
>>> list1=[[1,2],3,4]
>>> list2=copy.deepcopy(list1)
>>> list2
[[1, 2], 3, 4]
>>> id(list1)==id(list2)
```

False

>>> list2[0][0]="sun"

>>> list2

[['sun', 2], 3, 4]

>>> list1

[[1, 2], 3, 4]

Immutable object -------- assignment operator

Mutable object -------- shallow copy

Immutable --------- deep copy

Regular expression:

Regular expression in string

```
>>> str1="my name is suneel"
>>> str1.replace("suneel","aneeel")
'my name is aneeel'
>>> str2="main street broad road"
>>> str2.replace("road","rd")
'main street brd rd'
>>> str2[0:17]+str2[17:].replace("road","rd")
'main street broad rd'
```

Taking input (using raw_input() and input()) and displaying output (print statement); Putting Comments.

we are discussing about input and output functions before going to input and output function

first we need to know about function

function are nothing but these are the statements which are group together to do some particular task.

In our program we can used to enter information. But can be done by the input function.

```
>>> input()
10
'10'
>>> input("please enter your name:")
please enter your name:suneel
'suneel'
>>> print("hello welcome")
hello welcome
>>> var=10
>>> print(var)
10
```

Then go to file option

Go to new file:

```
name=input("enter the name:")
print("WELCOME")
print("hello",name)
```

save the file sun1.py

```
enter the name:suneel
WELCOME
hello Suneel
```

comments:

comment are nothing but which are written in our program these statement are equal lent to during execution of the program.

Why we use the comments because imagine that we wrote the program.

After one year or two year you want use that same code. but after the one or two years you may not remember. why you use some variable how the part of the program has execution. Etc.

If you write the comments writing your code then after one year or two years while re-referring the code you can read the description and you can understand why we use that variable how the part of the working and all

Imagine that you are referring that codewhich are written by other developer . so you may not understand why he use the variable. How the cods is running. if you wrote the description then you have underdstand.

Now we will see the how to use the command in our program for that we need to use # symbol. any thing here it will be treated as comment. It will be ignored execution of the program.

Output we will give

"""this is my first program

print("Hello")"""

print("welcome")

print("Have a nice day")

input: enter the name:suneel

WELCOME

hello suneel.

Types of error:

Syntax errors:

Syntax errors are the most basic **type** of **error**. They arise when the **Python** parser is unable to understand a line of code. ... In IDLE, it will highlight where the **syntax error** is. Most **syntax errors** are typos, incorrect indentation, or incorrect arguments.

Runtime error: A syntax **error** happens when **Python** can't understand what you are saying. A **run-time error** happens when **Python** understands what you are saying, but runs into trouble when following your instructions. ... This is called a **run-time error** because it occurs after the program starts running

Semantic Error:. 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. ... The meaning of the program (its **semantics**) is wrong.