Operators and Expressions in Python

- Operators are special symbols in Python that carry out arithmetic or logical computation.
- If any two or more Objects (or) Variables connected with an Operator then it is called Expression.

Operators Types :

1. Arithmetic Operators
2. Assignment Operator
3. Comparison (Relational) Operators
4. Logical (Boolean) Operators
5. Membership Operators

```
a) in
b) not in
```

6. Identity Operators

```
a) is
b) is not
```

1. Arithmetic Operators

- The purpose of Arithmetic Operators is that "To Perform Arithmetic Operations such Addition, Subtraction, Multiplication etc".
- If any two or more objects or variables connected with Arithmetic Operators then it is called Arithmetic Expression.
- Python programming contains 7 Arithmetic Operators and they are given in the following Table.

- "+","-","*","/","//","%","**" are arithmetic operators.

In [1]:
```
#Addition
3+5
```

Out[1]:
```
8
```

In [2]:
```
#Subtraction
50-33
```

Out[2]:
```
17
```

```
In [3]: #Multiplication
        11*10
```

```
Out[3]: 110
```

- **Division always gives output as float**

```
In [4]: #Division
        120/4
```

```
Out[4]: 30.0
```

```
In [6]: #Floor Division-----quatient
        14//4
```

```
Out[6]: 3
```

```
In [7]: #Modulo Division----remainder
        14%4
```

```
Out[7]: 2
```

```
In [11]: #Power/Exponent
         3**4
```

```
Out[11]: 81
```

```
In [12]: # Paranthesis
         (2+4) * (10+12)
```

```
Out[12]: 132
```

Arithmetic Operators Precedence

1. Paranthesis
2. Exponents
3. Floor Division
4. Division / Multiplication
5. Modulus Division
6. Addition / Subtraction

```
In [13]: 5*5+5/5-5
```

```
Out[13]: 21.0
```

- **When we use arithmetic operators, the boolean values will be automatically converted into int.**

In [15]:

```
True + True
```

Out[15]:

```
2
```

In [19]:

```
b=3.9
c=False
b+c
```

Out[19]:

```
3.9
```

In [20]:

```
b=3.9
c=False
b/c
```

```
---------------------------------------------------------------------------
ZeroDivisionError                         Traceback (most recent call last)
~\AppData\Local\Temp\ipykernel_4160\3526177587.py in <module>
      1 b=3.9
      2 c=False
----> 3 b/c

ZeroDivisionError: float division by zero
```

In [22]:

```
#Program for demonstrating functionality of Arithmetic Operators

# Ask user to enter two integer values
a=int(input("Enter Value of a:"))
b=int(input("Enter Value of b:"))

# Print  Text "Arithmetic Operators Result".
print("\tArithmetic Operators Result")

# Addition of two values
print("\tsum({},{})={}".format(a,b,a+b))
# Subtraction of two values
print("\tsub({},{})={}".format(a,b,a-b))
# Multiplication of two values
print("\tmul({},{})={}".format(a,b,a*b))
# Division of two values
print("\tDiv({},{})={}".format(a,b,a/b))
# Floor Division of two values
print("\tFloorDiv({},{})={}".format(a,b,a//b))
# Modulo Division of two values
print("\tMod({},{})={}".format(a,b,a%b))
# Exponent of two values
print("\tExpo({},{})={}".format(a,b,a**b))
```

```
Enter Value of a:10
Enter Value of b:3
        Arithmetic Operators Result
        sum(10,3)=13
        sub(10,3)=7
        mul(10,3)=30
        Div(10,3)=3.3333333333333335
        FloorDiv(10,3)=3
        Mod(10,3)=1
        Expo(10,3)=1000
```

2. Assignment Operator

- Assignment operators are used in Python to asign values to variables.
- The Purpose of Assignment Operator is that "To Transfer Right Hand Side Value / Expression to the Left Hand Side Variable / Object".

- (=, +=, -=, /=, //=, %=, **=) are assignment operators.

In [23]:

```
a=10
a
```

Out[23]:

```
10
```

In [24]:

```
id(a)
```

Out[24]:

```
3053133982288
```

In [25]:
```
a+=1        # a=a+1
a
```

Out[25]:
```
11
```

In [26]:
```
id(a)
```

Out[26]:
```
3053133982320
```

In [27]:
```
a-=4    #a=a-4
a
```

Out[27]:
```
7
```

In [28]:
```
a/=2    #a=a/2
a
```

Out[28]:
```
3.5
```

In [36]:
```
a=20
a//=3     #a=a//3
a
```

Out[36]:
```
6
```

In [37]:
```
a=20
a%=3     #a=a%3
a
```

Out[37]:
```
2
```

In [38]:
```
a=20
a**=3     #a=a**3
a
```

Out[38]:
```
8000
```

In [163]:

```
''' Program for cal all type of Arithmetic Operations by using
Multi line assigments.'''

# Ask user to enter two values.
a=int(input("Enter First Value:"))
b=int(input("Enter Second Value:"))

#Calculations - Multiline assignment
aop,sop,mop,dop,fdop,mdop,exop=a+b,a-b,a*b,a/b,a//b,a%b,a**b

# Addition of two values
print("sum={}".format(aop))

# Subtraction of two values
print("sub={}".format(sop))

# Multiplication of two values
print("mul={}".format(mop))

# Division of two values
print("division={}".format(dop))

# Floor Divisionn of two values
print("floor div={}".format(fdop))

# Modulo Division of two values
print("Mod={}".format(mdop))

# Exponentiation of two values
print("Exp={}".format(exop))
```

```
Enter First Value:10
Enter Second Value:3
sum=13
sub=7
mul=30
division=3.3333333333333335
floor div=3
Mod=1
Exp=1000
```

In [39]:

```
#Program for cal Swapping any two by using Multi line assigments.

# Ask user to enter two values.
a,b=input("Enter First Value:") , input("Enter Second Value:")
print("-"*50)
print("Original Values:")
print("Value of a={}\tValue of b={}".format(a,b))
print("-"*50)
#Swapping Logic
a,b,=b,a # Multi Line assigment
print("Swapped Values:")
print("Value of a={}\tValue of b={}".format(a,b))
print("-"*50)
```

```
Enter First Value:10
Enter Second Value:20
--------------------------------------------------
Original Values:
Value of a=10   Value of b=20
--------------------------------------------------
Swapped Values:
Value of a=20   Value of b=10
--------------------------------------------------
```

3. Comparison (Relational) Operators

- The Purpose of Relational Operators is that "To Compare Two or More Values".
- If two or more variables / Values / objects connected with Relational Operator then it is called Relational Expression.
- Relational Expressions are also called Test Conditions and whose result is always either to be True or False (bool type result).

- **(>, <, >=, <=, ==, !=) are comparison operators.**

In [40]:

```
# is greater than
10>8
```

Out[40]:

```
True
```

In [41]:

```
# is less than
45<12
```

Out[41]:

```
False
```

In [42]:

```
# is less than
11<12
```

Out[42]:

```
True
```

```
In [43]: 3*3 < 3*4

Out[43]: True
```

```
In [44]: # is equal to
         22==22

Out[44]: True
```

```
In [45]: # is equal to
         20==12

Out[45]: False
```

```
In [46]: # is not equal to
         22!=22

Out[46]: False
```

```
In [48]: # is not equal to
         212!=122

Out[48]: True
```

```
In [49]: # greater than or equal to
         100>=90

Out[49]: True
```

```
In [50]: # greater than or equal to
         29>=29

Out[50]: True
```

```
In [51]: # greater than or equal to
         100>=120

Out[51]: False
```

```
In [52]: # less than or equal to
         100<=90

Out[52]: False
```

```
In [53]: # less than or equal to
         90<=90

Out[53]: True
```

```
In [54]: # less than or equal to
         45<=50

Out[54]: True
```

```
In [55]: "hi"=="HI"

Out[55]: False

In [56]: "H"=="H"

Out[56]: True

In [57]: "a"!="A"

Out[57]: True

In [58]: "B "=="B"  # Space added

Out[58]: False

In [59]: "a"<"A"

Out[59]: False

In [60]: "b">"B"

Out[60]: True

In [61]: None==None

Out[61]: True

In [62]: "  "==None

Out[62]: False

In [63]: "  "==False

Out[63]: False
```

In [65]:

```
# program for demonstarting relational operators.

#Ask user to enter two values.
a,b=float(input("Enter Value of a:")),float(input("Enter Value of b:"))

# Print "Result of Various Relational Operators" for display purpose.
print("Result of Various Relational Operators:")

# a is greate than b
print("\t{} > {}={}".format(a,b,a>b))

# a is less than b
print("\n\t{} < {}={}".format(a,b,a<b))

# a is equal to b
print("\n\t{} == {}={}".format(a,b,a==b))

# a is not equal to b
print("\n\t{} != {}={}".format(a,b,a!=b))

# a is greater than or equal to b
print("\n\t{} >= {}={}".format(a,b,a>=b))

# a is less than or equal to b
print("\n\t{} <= {}={}".format(a,b,a<=b))
```

```
Enter Value of a:10
Enter Value of b:20
Result of Various Relational Operators:
        10.0 > 20.0=False

        10.0 < 20.0=True

        10.0 == 20.0=False

        10.0 != 20.0=True

        10.0 >= 20.0=False

        10.0 <= 20.0=True
```

4. Logical (Boolean) Operators

- The purpose of Logical Operators is that "To Compare the result of Two or more Relational Expressions".
- If two or more Relational Expressions or Test Conditions connected with Logical Operators then it is called Logical Expression.
- Logical Expressions are also called Compound Test Conditions and whose result is always either to be True or False (bool type result).
- In Python Programming, we have 3 types of Logical Operators. They are

```
1) and operator
2) or operator
3) not operator
```

1) and operator

- When we have "and" operator -- all conditions should be True then only overall output will be True.

```
In [66]: 10>5 and 20>11

Out[66]: True
```

Short Circuit Evaluation in the case "and" operator:

- If "and" operator connected with Two Or More Relational Expressions and if the First Relational Expression is False then PVM never evaluate Second and Sub-Sequent Relational Expressions and Result of the Entire Test Conditions is Taken as FALSE. This process is called Short Circuit Evaluation.

```
In [67]: 10>15 and 13>11    # Short Circuit Evaluation

Out[67]: False

In [68]: 30<10 and 20>10 and 2<4     # Short Circuit Evaluation

Out[68]: False

In [69]: 30>10 and 20<10 and 2<4     # Short Circuit Evaluation

Out[69]: False

In [70]: True and True

Out[70]: True

In [71]: False and False

Out[71]: False

In [72]: False and True

Out[72]: False

In [73]: True and False

Out[73]: False
```

2) or operator

- When we have "or" operator -- atleast one condition should be True then only overall output will be True.

```
In [75]: 100>50 or 20>10

Out[75]: True
```

```
In [76]: 10<1 or 20>10

Out[76]: True
```

Short Circuit Evaluation in the case "or" operator:

- If "or" operator connected with Two Or More Relational Expressions and if the First Relational Expression is True then PVM never evaluate Second and Sub-Sequent Relational Expressions and Result of the Entire Test Conditions is Taken as TRUE. This process is called Short Circuit Evaluation.

```
In [77]: 30>22 or 10>300      # Short circuit evaluation

Out[77]: True
```

```
In [79]: 100>29 or 20>100 or 3>9     # Short circuit evaluation

Out[79]: True
```

```
In [80]: True or True

Out[80]: True
```

```
In [81]: True or False

Out[81]: True
```

```
In [82]: False or False

Out[82]: False
```

3) not operator

- NOT operator is a Boolean operator that returns TRUE or 1 when the operand is FALSE or 0, and returns FALSE or 0 when the operand is TRUE or 1.
- Essentially, the operator reverses the logical value associated with the expression on which it operates.

```
In [83]: 10>5 and 20<100
```

```
Out[83]: True
```

```
In [84]: not (10>5 and 20<100)
```

```
Out[84]: False
```

```
In [85]: 100>50 and 200<100
```

```
Out[85]: False
```

```
In [86]: not (100>50 and 200<100)
```

```
Out[86]: True
```

```
In [87]: not True
```

```
Out[87]: False
```

```
In [88]: not False
```

```
Out[88]: True
```

```
In [89]: 10>5 or 20>100
```

```
Out[89]: True
```

```
In [90]: not (10>5 or 20>100)
```

```
Out[90]: False
```

Logical Operators Precedence

- Logical not
- Logical and
- Logival or

```
In [91]: # "Logical and" followed by "Logical or"
         (2==2) or (3==3) and (3==4)
```

```
Out[91]: True
```

```
In [92]: # "Logical and" followed by "Logical or"
         (5==2) or (3==3) and (3==4)
```

```
Out[92]: False
```

5. Membership Operators

- The purpose of Membership operators is that " To Check the existence of a Particular value in Iterable Object".
- An Iterable Object is one, which contains More Number of Values (Sequence Type, List Type, Set Type, dict Type).
- In Python Programming, we have 2 Types of Membership Operators. They are

```
1. in
2. not in
```

1. in

- The "in" Operator Returns True provided The "value" present in IterableObject.
- The "in" Operator Returns False provided The "value" not present in IterableObject.

- **Syntax: - Value in IterableObject**

In [95]:
```
s="PYTHON"
print(s,type(s))
```

```
PYTHON <class 'str'>
```

In [96]:
```
"P" in s
```

Out[96]:
```
True
```

In [97]:
```
"p" in s
```

Out[97]:
```
False
```

In [98]:
```
"on" in s
```

Out[98]:
```
False
```

In [99]:
```
"noh" in s
```

Out[99]:
```
False
```

In [101]:
```
l1=[10,"Rossum",23.45,2+3j]
print(l1,type(l1))
```

```
[10, 'Rossum', 23.45, (2+3j)] <class 'list'>
```

In [102]:
```
10 in l1
```

Out[102]:
```
True
```

In [103]:
```
"rossum" in l1
```

Out[103]:
```
False
```

```
In [104]: 2 in l1

Out[104]: False
```

```
In [105]: 10 in l1[0::]

Out[105]: True
```

```
In [106]: "MADAM" in "MADAM"[::-1]

Out[106]: True
```

2. not in

- The "not in" Operator Returns True provided The "value" not present in IterableObject.
- The "not in" Operator Returns False provided The "value" present in IterableObject.

- **Syntax: - Value not in IterableObject**

```
In [107]: l1=[10,"Rossum",23.45,2+3j]
          print(l1,type(l1))

          [10, 'Rossum', 23.45, (2+3j)] <class 'list'>
```

```
In [108]: "Rossum" not in l1

Out[108]: False
```

```
In [111]: s="Python"
          print(s)

          Python
```

```
In [112]: "P" not in s

Out[112]: False
```

```
In [113]: "om" not in s

Out[113]: True
```

```
In [114]: "MADAM"  not in "MADAM"[::-1]

Out[114]: False
```

```
In [115]: "RACECAR" not in "RACECAR"[::-1]

Out[115]: False
```

```
In [116]: s1={10,20,30,40,50,60}
          print(s1,type(s1))

          {50, 20, 40, 10, 60, 30} <class 'set'>
```

```
In [117]: s1[0] in s1

          ---------------------------------------------------------------------------
          TypeError                                 Traceback (most recent call last)
          ~\AppData\Local\Temp\ipykernel_4160\655467604.py in <module>
          ----> 1 s1[0] in s1

          TypeError: 'set' object is not subscriptable
```

```
In [118]: d1={10:"Apple",20:"Kiwi",30:"Banana"}
          print(d1,type(d1))

          {10: 'Apple', 20: 'Kiwi', 30: 'Banana'} <class 'dict'>
```

```
In [119]: d1.get(10) in d1

Out[119]: False
```

```
In [120]: d1.get(10) not in d1[10][::-1]

Out[120]: True
```

```
In [121]: d1.items() in d1

          ---------------------------------------------------------------------------
          TypeError                                 Traceback (most recent call last)
          ~\AppData\Local\Temp\ipykernel_4160\3736688872.py in <module>
          ----> 1 d1.items() in d1

          TypeError: unhashable type: 'dict_items'
```

6. Identity Operators

- The purpose of Identity Operators is that " To Check or Compare the memory addresses of Two Objects".
- In Python, we have two Identity Operators. They are

```
          1. is
          2. is not
```

1. is

- The "is" operator returns True provided The memory address of Object1 and Object2 must be Same.

- The "is" operator returns False provided The memory address of Object1 and Object2 must be Different.

- **Syntax: object1 is object2**

2. is not

- The "is not" operator returns True provided The memory address of Object1 and Object2 must be Different.
- The "is not " operator returns False provided The memory address of Object1 and Object2 must be Same.

- **Syntax: - object1 is not object2**

In [122]:

```python
a=5
b=5
print(a,type(a),id(a))
print(b,type(b),id(b))
```

```
5 <class 'int'> 3053133982128
5 <class 'int'> 3053133982128
```

In [124]:

```python
a is b
```

Out[124]:

```
True
```

In [125]:

```python
a is not b
```

Out[125]:

```
False
```

In [126]:

```python
s1="PYTHON"
s2="PYTHON"
print(s1,type(s1),id(s1))
print(s2,type(s2),id(s2))
```

```
PYTHON <class 'str'> 3053207796592
PYTHON <class 'str'> 3053207796592
```

In [127]:

```python
s1 is s2
```

Out[127]:

```
True
```

In [128]:

```python
s1 is not s2
```

Out[128]:

```
False
```

In [129]:

```
a=None
b=None
print(a,type(a),id(a))
print(b,type(b),id(b))
```

```
None <class 'NoneType'> 140732990930136
None <class 'NoneType'> 140732990930136
```

In [130]:

```
a is not b
```

Out[130]:

```
False
```

In [131]:

```
a is b
```

Out[131]:

```
True
```

In [132]:

```
a=True
b=True
print(a,type(a),id(a))
print(b,type(b),id(b))
```

```
True <class 'bool'> 140732990879848
True <class 'bool'> 140732990879848
```

In [133]:

```
a is b
```

Out[133]:

```
True
```

In [134]:

```
a=False
b=False
print(a,type(a),id(a))
print(b,type(b),id(b))
```

```
False <class 'bool'> 140732990879880
False <class 'bool'> 140732990879880
```

In [135]:

```
a is not b
```

Out[135]:

```
False
```

In [136]:

```
a is b
```

Out[136]:

```
True
```

In [137]:

```
d1={10:"Apple",20:"Mango"}
d2={10:"Apple",20:"Mango"}
print(d1,type(d1),id(d1))
print(d2,type(d2),id(d2))
```

```
{10: 'Apple', 20: 'Mango'} <class 'dict'> 3053217406272
{10: 'Apple', 20: 'Mango'} <class 'dict'> 3053218386496
```

In [138]:

```
d1 is d2
```

Out[138]:

```
False
```

In [139]:

```
d1 is not d2
```

Out[139]:

```
True
```

In [140]:

```
s1={10,"Deepthi"}
s2={10,"Deepthi"}
print(s1,type(s1),id(s1))
print(s2,type(s2),id(s2))
```

```
{10, 'Deepthi'} <class 'set'> 3053218287168
{10, 'Deepthi'} <class 'set'> 3053218480416
```

In [141]:

```
s1 is s2
```

Out[141]:

```
False
```

In [142]:

```
s1 is not s2
```

Out[142]:

```
True
```

In [143]:

```
l1=[10,"Rossum"]
l2=[10,"Rossum"]
print(l1,type(l1),id(l1))
print(l2,type(l2),id(l2))
```

```
[10, 'Rossum'] <class 'list'> 3053217464384
[10, 'Rossum'] <class 'list'> 3053218447936
```

In [144]:

```
l1 is l2
```

Out[144]:

```
False
```

In [145]:

```
l2 is not s1
```

Out[145]:

```
True
```

In [146]:

```
l1=[10,"Rossum"]
l2=l1
print(l1,type(l1),id(l1))
print(l2,type(l2),id(l2))
```

```
[10, 'Rossum'] <class 'list'> 3053218370432
[10, 'Rossum'] <class 'list'> 3053218370432
```

In [147]:

```
l1 is l2
```

Out[147]:

```
True
```

In [148]:

```
l1=[10,"Rossum"]
l2=l1.copy()
print(l1,type(l1),id(l1))
print(l2,type(l2),id(l2))
```

```
[10, 'Rossum'] <class 'list'> 3053217406912
[10, 'Rossum'] <class 'list'> 3053216633664
```

In [149]:

```
l1 is l2
```

Out[149]:

```
False
```

In [150]:

```
l1 is not l2
```

Out[150]:

```
True
```

In [151]:

```
r1=range(10,20)
r2=range(10,20)
print(r1,type(r1),id(r1))
print(r2,type(r2),id(r2))
```

```
range(10, 20) <class 'range'> 3053218343648
range(10, 20) <class 'range'> 3053218342208
```

In [152]:

```
r1 is r2
```

Out[152]:

```
False
```

In [153]:

```
s1="INDIA"
s2="INDIA"
print(s1,type(s1),id(s1))
print(s2,type(s2),id(s2))
```

```
INDIA <class 'str'> 3053218450992
INDIA <class 'str'> 3053218450992
```

In [154]:

```
s1 is s2
```

Out[154]:

```
True
```

In [155]:

```
s1 is not s2
```

Out[155]:

```
False
```

In [156]:

```
a=2+3j
b=2+3j
print(a,type(a),id(a))
print(b,type(b),id(b))
```

```
(2+3j) <class 'complex'> 3053218410384
(2+3j) <class 'complex'> 3053218409808
```

```
In [157]: a is not b
```

```
Out[157]: True
```

```
In [158]: a is b
```

```
Out[158]: False
```

Operators Precedence

- Arithmetic Operators
- Comparison (Relational) Operators
- Membership Operators
- Identity Operators
- Logical Operators
- Assignment Operator

```
In [161]: b=(1>2) or (3>=3)
          b
```

```
Out[161]: True
```