In [ ]:

```
Python Operators

Operators are used to perform operations on variables and values.

Python divides the operators in the following groups:

1. Arithmetic operators

2. Assignment operators

3. Comparison operators

4. Logical operators

5. Identity operators

6. Membership operators

7. Bitwise operators
```

In [ ]:

```
Python Arithmetic Operators
```

In [ ]:

```
Python Arithmetic Operators

Arithmetic operators are used with numeric values to perform common mathematical operations

Operator    Name    Example
+   Addition    x + y
-   Subtraction x - y
*   Multiplication  x * y
/   Division    x / y
%   Modulus x % y
**  Exponentiation  x ** y
//  Floor division  x // y
```

In [1]:

```
x = 5
y = 3

print(x + y)
```

```
8
```

In [1]:

```
x=int(input("Enter the X Value"))
y=int(input("Enter the Y Value"))
z=x+y
print(z)
```

```
Enter the X Value5
Enter the Y Value4
9
```

In [2]:

```
x = 5
y = 3

print(x - y)
```

```
2
```

In [3]:

```
x = 5
y = 3

print(x * y)
```

```
15
```

In [4]:

```
x = 12
y = 3

print(x / y)
```

```
4.0
```

In [4]:

```
x = 30
y = 4

print(x % y)
```

```
2
```

In [6]:

```
x = 2
y = 5

print(x ** y) #same as 2*2*2*2*2
```

32

In [7]:

```
x = 15
y = 2

print(x // y)

#the floor division // rounds the result down to the nearest whole number
```

7

In [ ]:

```
Python Assignment Operators
```

In [ ]:

```
Python Assignment Operators

Assignment operators are used to assign values to variables:

Operator    Example Same As
=   x = 5   x = 5
+=  x += 3  x = x + 3
-=  x -= 3  x = x - 3
*=  x *= 3  x = x * 3
/=  x /= 3  x = x / 3
%=  x %= 3  x = x % 3
//= x //= 3 x = x // 3
**= x **= 3 x = x ** 3
&=  x &= 3  x = x & 3
|=  x |= 3  x = x | 3
^=  x ^= 3  x = x ^ 3
>>= x >>= 3 x = x >> 3
<<= x <<= 3 x = x << 3
```

In [8]:

```
x = 5

print(x)
```

5

In [5]:

```
x = 12

x += 3

print(x)
```

```
15
```

In [7]:

```
x = 8

x -= 3

print(x)
```

```
5
```

In [11]:

```
x = 5

x *= 3

print(x)
```

```
15
```

In [12]:

```
x = 5

x /= 3

print(x)
```

```
1.6666666666666667
```

In [13]:

```
x = 5

x%=3

print(x)
```

```
2
```

In [14]:

```
x = 5

x//=3

print(x)
```

1

In [10]:

```
x = 3

x **= 3

print(x)
```

27

In [16]:

```
x = 5

x &= 3

print(x)
```

1

In [17]:

```
x = 5

x |= 3

print(x)
```

7

In [18]:

```
x = 5

x ^= 3

print(x)
```

6

In [13]:

```
x = 10

x >>= 10

print(x)
```

```
0
```

In [20]:

```
x = 5

x <<= 3

print(x)
```

```
40
```

In [ ]:

```
Python Comparison Operators
```

In [ ]:

```
Python Comparison Operators

Comparison operators are used to compare two values:

Operator    Name    Example
==  Equal   x == y
!=  Not equal   x != y
>   Greater than    x > y
<   Less than   x < y
>=  Greater than or equal to    x >= y
<=  Less than or equal to   x <= y
```

In [2]:

```
x = 5
y = 3

print(x == y)

# returns False because 5 is not equal to 3
```

```
False
```

In [22]:

```
x = 5
y = 3

print(x != y)

# returns True because 5 is not equal to 3
```

```
True
```

In [23]:

```
x = 5
y = 3

print(x > y)

# returns True because 5 is greater than 3
```

```
True
```

In [24]:

```
x = 5
y = 3

print(x < y)

# returns False because 5 is not less than 3
```

```
False
```

In [25]:

```
x = 5
y = 3

print(x >= y)

# returns True because five is greater, or equal, to 3
```

```
True
```

In [26]:

```
x = 5
y = 3

print(x <= y)

# returns False because 5 is neither less than or equal to 3
```

```
False
```

In [ ]:

```
Python Logical Operators
```

In [ ]:

```
Python Logical Operators

Logical operators are used to combine conditional statements:

Operator    Description Example
and     Returns True if both statements are true    x < 5 and  x < 10
or  Returns True if one of the statements is true   x < 5 or x < 4
not Reverse the result, returns False if the result is true not(x < 5 and x < 10)
```

In [27]:

```
x = 5

print(x > 3 and x < 10)

# returns True because 5 is greater than 3 AND 5 is less than 10
```

```
True
```

In [28]:

```
x = 5

print(x > 3 or x < 4)

# returns True because one of the conditions are true (5 is greater than 3, but 5 is not le
```

```
True
```

In [29]:

```
x = 5

print(not(x > 3 and x < 10))

# returns False because not is used to reverse the result
```

```
False
```

In [ ]:

```
Python Identity Operators
```

In [ ]:

```
Python Identity Operators

Identity operators are used to compare the objects, not if they are equal, but if they are

Operator    Description Example
is  Returns True if both variables are the same object  x is y
is not  Returns True if both variables are not the same object  x
```

In [30]:

```
x = ["apple", "banana"]
y = ["apple", "banana"]
z = x

print(x is not z)

# returns False because z is the same object as x

print(x is not y)

# returns True because x is not the same object as y, even if they have the same content

print(x != y)

# to demonstrate the difference betweeen "is not" and "!=": this comparison returns False b
```

```
False
True
False
```

In [3]:

```
x = ["apple", "banana"]
y = ["apple", "banana"]
z = x

print(x is z)

# returns True because z is the same object as x

print(x is y)

# returns False because x is not the same object as y, even if they have the same content

print(x == y)

# to demonstrate the difference betweeen "is" and "==": this comparison returns True becaus
```

```
True
False
True
```

In [ ]:

```
Python Membership Operators
```

In [ ]:

```
Python Membership Operators

Membership operators are used to test if a sequence is presented in an object:

Operator    Description Example
in  Returns True if a sequence with the specified value is present in the object    x in y
not in  Returns True if a sequence with the specified value is not present in the object
```

In [32]:

```
x = ["apple", "banana"]

print("pineapple" not in x)

# returns True because a sequence with the value "pineapple" is not in the list
```

```
True
```

In [33]:

```
x = ["apple", "banana"]

print("banana" in x)

# returns True because a sequence with the value "banana" is in the list
```

```
True
```

In [ ]:

```
Python Bitwise Operators
Bitwise operators are used to compare (binary) numbers:

Operator    Name    Description
&   AND Sets each bit to 1 if both bits are 1
|   OR  Sets each bit to 1 if one of two bits is 1
 ^  XOR Sets each bit to 1 if only one of two bits is 1
~   NOT Inverts all the bits
<<  Zero fill left shift    Shift left by pushing zeros in from the right and let the leftm
>>  Signed right shift  Shift right by pushing copies of the leftmost bit in from the left,
```