Operators in Python

Python divides the operators into the following seven groups:

- Arithmetic operators
- Assignment operators
- Comparison operators
- Logical operators
- Identity operators
- Membership operators
- Bitwise operators

Seven Arithmetic operators

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

Floor Division

```
x = 15
y = 2

print(x // y)

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

```
7
```

Floor: next lower int
Ceil: next higher int
Round: nearest int

```
>>> round(5.2)
5
>>> round(5.9)
6
>>> math.floor(5.9)
5
>>>
>>> round(5.8)
6
>>> math.floor(5.8)
5
>>>
>>> math.ceil(5.8)
6
>>> math.ceil(5.2)
6
```

+ And * Operators Can Act on Strings

String Concatenation using + Operator

```
>>> x = 'johanth'
>>> y = 'yada'
>>> x + y
'johanthyada'
```

String repetition using * Operator

```
>>> x*3
'johanthjohanthjohanth'
```

This phenomenon is over-loading of operator.

```
>>> a = 5
>>> b = 6
>>> a + b
11
>>> a = "rakesh"
>>> b = "vikash"
>>> a + b
'rakeshvikash'
>>>
>>> a = 5
>>> b = 6
>>> a*b
30
>>> a = "rakesh"
>>> b = 'vikash'
>>> a * 3
'rakeshrakeshrakesh'
>>> b * 2
'vikashvikash'
>>>
>>> l = ['rakesh', 'vikash', 'ashish']
>>> l * 3
['rakesh', 'vikash', 'ashish', 'rakesh', 'vikash', 'ashish', 'rakesh',
'vikash', 'ashish']
>>>
```

+ And * Operators Can Act on Lists

```
>>> l = ['rakesh', 'vikash', 'ashish']
>>> l * 3
['rakesh', 'vikash', 'ashish', 'rakesh', 'vikash', 'ashish', 'rakesh', 'vikash', 'ashish']
>>> k = ['johanth', 'sonia']
>>> l + k
['rakesh', 'vikash', 'ashish', 'johanth', 'sonia']
>>>
```

Assignment Operators

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

X+=5: shorthand expression for x=x+5

Python does not have ++ (as in i++) operator.
If you want to increment a variable:
x += 1

Assignment Operator (Problem)

What is the output of this program?

```
x = 5
y = 10
x = y
print(x)
print(y)
```

Assignment Operator (Solution)

What is the output of this program?

```
x = 5    # x is the variable name. Explicit value being passed is 5.
y = 10  # y is the variable name. Explicit value being passed is 10.
x = y    # Variable which is taking the value is on the left. X is taking the value. And value being passed is 10 through y.
print(x)
print(y)
```

Comparison Operators

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

Logical Operators

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)

“and” in C++ is: &&
“or” in C++ is: ||

“and” checks both the conditions. It returns true only if both of them are true.

“or” checks both the conditions. It returns true if either of the two conditions are true.

```
x = 6
```

```
y = x < 7
```

```
y
```

```
True
```

```
not(y)
```

```
False
```

Truth Table For ‘and’

If both expressions are True, then it will return True.

Left Side Exp.	Right Side Exp.	Overall Result
True	True	True
False	True	False
True	False	False
False	False	False

Problem

What will be the output of:

```
x = 0
print(x<5 and x<10)
x = 6
print(x<5 and x<10)

x = 0
print(x<5 or x<4)
x = 6
print(x<5 or x<4)
```

Solution

What will be the output of:

```
x = 0
print(x<5 and x<10) # True and True
x = 6
print(x<5 and x<10) # False and True

X = 0
print(x<5 or x<4) # True
X = 6
print(x<5 or x<4) # False
```

Identity Operators

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 is not y

```
s1 = 'mystring'
s2 = 'mystring'
```

```
s1 is s2
```

```
True
```

```
id(s1)
```

```
140082311787504
```

```
id(s2)
```

```
140082311787504
```

```
l1 = ['apple', 'orange']
l2 = ['apple', 'orange']
```

```
l1 is l2
```

```
False
```

Strings are immutable, while lists are mutable

```
>>> b = 'mystring'
>>> c = 'mystring'
>>> id(b)
139777910977712
>>> id(c)
139777910977712
>>>
>>> d = ['second', 'string']
>>> e = ['second', 'string']
>>> id(d)
139777910977984
>>> id(e)
139777910977920
>>>
```

```
>>> b[0]
'm'
>>> b[1]
'y'
>>> b[2]
's'
>>> d[0]
'second'
>>> d[1]
'string'
>>>
>>> d[0] = 'new'
>>> d
['new', 'string']
>>>
>>> b[0] = 'a'
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'str' object does not support item assignment
```

Note: You cannot change a string in it’s place.

Membership Operators

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	x not in y

```
# Checking an item exists in a list.
# Use of 'in' and 'not in' operators

list1 = ["apple", "banana", "cherry"]
if "apple" in list1:
  print("Yes, 'apple' is in the fruits list")

if "guava" not in list1:
  print("No, 'guava' is not in the fruits list")
```

```
Yes, 'apple' is in the fruits list
No, 'guava' is not in the fruits list
```

```
print('a' in "apple") # True
print('o' in "apple") # False
```

```
True
False
```

```
>>> l
['rakesh', 'vikash', 'ashish']
>>> 'rakesh' in l
True
>>> 'sonia' in l
False
>>>
```

```
>>> l = ['one', 'two', 'three']
>>> s = {'one', 'two', 'three'}
>>> t = ('one', 'two', 'three')
>>> d = {'one': 1, 'two': 2, 'three': 3}
>>> print(type(l))
<class 'list'>
>>> print(type(s))
<class 'set'>
>>> print(type(t))
<class 'tuple'>
>>> print(type(d))
<class 'dict'>
>>>
>>> print('two' in l)
True
>>> print('three' in s)
True
>>> print('four' in t)
False
>>> print('one' in d)
True
>>>
```

Bitwise Operators

Operator	Name	Description	Example
&	AND	Sets each bit to 1 if both bits are 1	x & y
|	OR	Sets each bit to 1 if one of two bits is 1	x | y
^	XOR	Sets each bit to 1 if only one of two bits is 1	x ^ y
~	NOT	Inverts all the bits	~x
<<	Zero fill left shift	Shift left by pushing zeros in from the right and let the leftmost bits fall off	x << 2
>>	Signed right shift	Shift right by pushing copies of the leftmost bit in from the left, and let the rightmost bits fall off	x >> 2

Bitwise &

2	5	
2	2	1
	1	0

5 ->101

2	7	
2	3	1
	1	1

7 -> 111

In: 5	In: 7	Op: &
1	1	1
0	1	0
1	1	1

```
>>> 5 & 7
5

>>> bin(5)
'0b101'

>>> bin(7)
'0b111'
>>>
```

Bitwise |

2	5	
2	2	1
	1	0

5 ->101

2	7	
2	3	1
	1	1

7 -> 111

```
>>> bin(5)
'0b101'
>>> bin(7)
'0b111'
>>>
>>> 5 | 7
7
```

In: 5	In: 7	Op: |
1	1	1
0	1	1
1	1	1

Problem

You have two switches. One is MCB and second one is in the light board.
State of a switch is: 1 if it is on.
State of a switch is: 0 if it is off.

If either of the switch is off, the fan won’t run.

Whici operator is acting between MCB and light board.

MCB	Light Board	Fan State
0	0	0
0	1	0
1	0	0
1	1	1

```
>>> 0 & 1
0
>>> 0 | 1
1
```

Bitwise ~

When x is a natural number:
~x = -(x+1)
Below we are assuming register size is of 3 bits.