Operations on Booleans

Operations on Booleans
1. Arithmetic

You can apply some arithmetic operations to a set. It takes 0 for False, and 1 for True, and then applies the operator to them.

Addition
You can add two or more Booleans. Let's see how that works.

```
>>> True+False #1+0

1

>>> True+True #1+1

2

>>> False+True #0+1

1

>>> False+False #0+0
```

Subtraction and Multiplication
The same strategy is adopted for subtraction and multiplication.

```
>>> False-True

-1
```

Division
Let's try dividing Booleans.

```
>>> False/True

0.0
```

Modulus, Exponentiation, and Floor Division

The same rules apply for modulus, exponentiation, and floor division as well.

```
>>> False%True
0
>>> True**False
1
>> False**False
1
>>> 0//1

0
```

Try your own combinations like the one below.

```
>>> (True+True)*False+True

1
```

Relational

The relational operators we've learnt so far are >, <, >=, <=, !=, and ==. All of these apply to Boolean values. We will show you a few examples, you should try the rest of them.

```
>>> False>True

False

>>> False<=True

True
```

3. Bitwise

Normally, the bitwise operators operate bit-by bit. For example, the following code ORs the bits of 2(010) and 5(101), and produces the result 7(111).

```
>>> 2|5

7
```

But the bitwise operators also apply to Booleans. Let's see how.

Bitwise &
It returns True only if both values are True.

```
>>> True&False

False

>>> True&True

True
```

Since Booleans are single-bit, it's equivalent to applying these operations on 0 and/or 1.

Bitwise |
It returns False only if both values are False.

```
>>> False|True

True
```

Bitwise XOR (^)
This returns True only if one value is True and one is False.

```
>>> False^True

True

>>> False^False

False

>>> True^True

False
```

Binary 1's Complement
This calculates 1's complement for True(1) and False(0).

```
>>> ~True

-2

>>> ~False

-1
```

Left-shift(<<) and Right-shift(>>) Operators
As discussed earlier, these operators shift the value by specified number of bits left and right, respectively.

```
>>> False>>2

0

>>> True<<2

4
```

True is 1. When shifted two places two the left, it results in 100, which is binary for 4. Hence, it returns 4.

4. Identity

The identity operators ‘is’ and ‘is not’ apply to Booleans.

>>> False is False

True

>>> False is 0

False

5. Logical

Finally, even the logical operators apply on Booleans.

>>> False and True

False

This was all about the article on Python set and booleans.

Data structures

```
x=False
print(x)

x=True
print(x+x)

a=[]
x=not a
print(type(a),type(x),x,x+x)
a=""
x=not(a)
print(type(a),type(x),x,x+x)
a=()
x=not a
print(type(a),type(x),x,x+x)
a={}
x=not a
print(type(a),type(x),x,x+x)
```

Short circuit evaluation of logical expressions

```
>>> x = 6
>>> y = 2
>>> x >= 2 and (x/y) > 2
True
>>> x = 1
>>> y = 0
>>> x >= 2 and (x/y) > 2
False
>>> x = 6
>>> y = 0
>>> x >= 2 and (x/y) > 2
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ZeroDivisionError: integer division or modulo by zero
>>>
```

The third calculation
failed because Python was evaluating (x/y) and y
was zero which causes a runtime error.
But the second example did not fail because
the first part of the expression x >= 2
evaluated to False so the (x/y) was not
ever executed due to the short circuit
rule and there was no error.