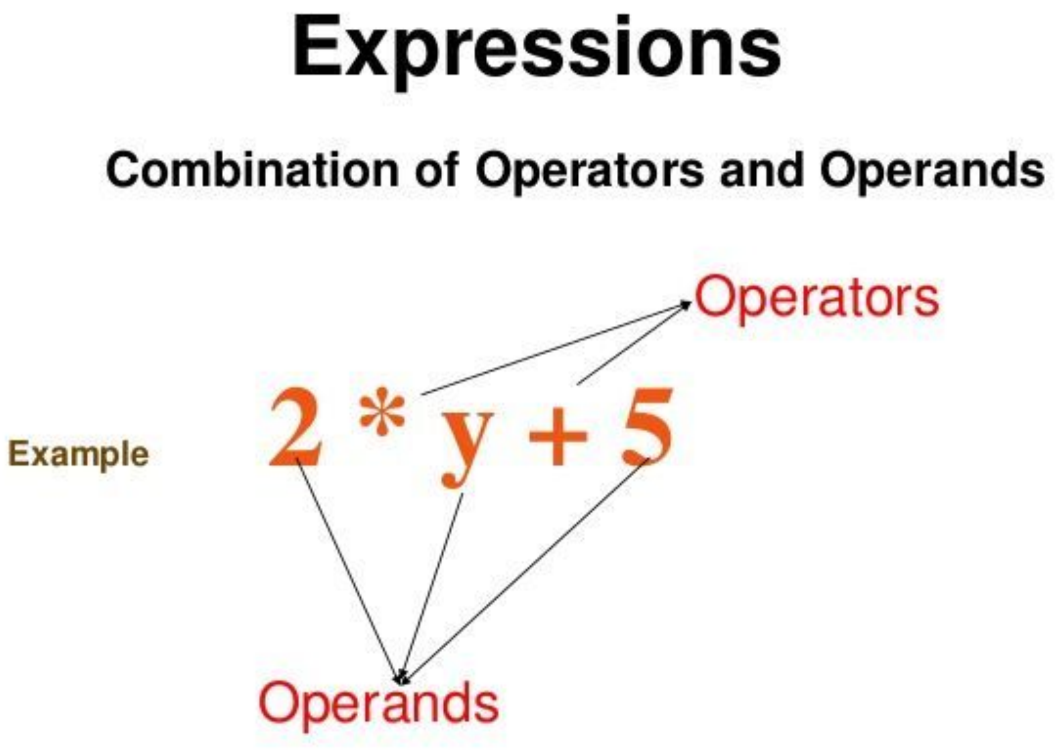
Operators and Expressions



Operators:

Operators are special symbols that represent computations like addition and multiplication. The values the operator is applied to are called *operands*.

Arithmetic Operators

Symbol	Task performed
+	Addition

-	Subtraction
*	Multiplication
/	Division
**	Exponentiation
//	Floored Division

```
>>>a=5+5  #addition
10
>>>b=4-2  #subtraction
2
>>>c=2*2  #multiplication
4
>>> 4/2   #division
2.0       #division always returns a float value
>>> 4/2
2         # floor division always returns an int value
>>> 4**2  #exponentiation
16

>>>18 % 4 #remainder
2 # always return the remainder of the division
```

The last printed value is stored in operator '_'. So, in an interactive mode, it is easier to add the previous value just by writing underscore(_)
For example:

```
>>> 5+7
12
>>> _+5
17
```

Each line in python is treated as a unique line. But there is a unique feature by which we can extend the lines. This can be done by using '\' operator.

```
>>> 5+\
    8+\
    7
```

```
20
```

Other types of operators

Comparison Operators

Symbol	Task performed
>	Greater than
<	Less than
>=	Greater than or equal to
<=	Less than or equal to
==	Equal to
!=	Not equal to

Boolean Operator

Symbol	Task performed
and	and logic
or	or logic
not	negation

Bitwise Operators

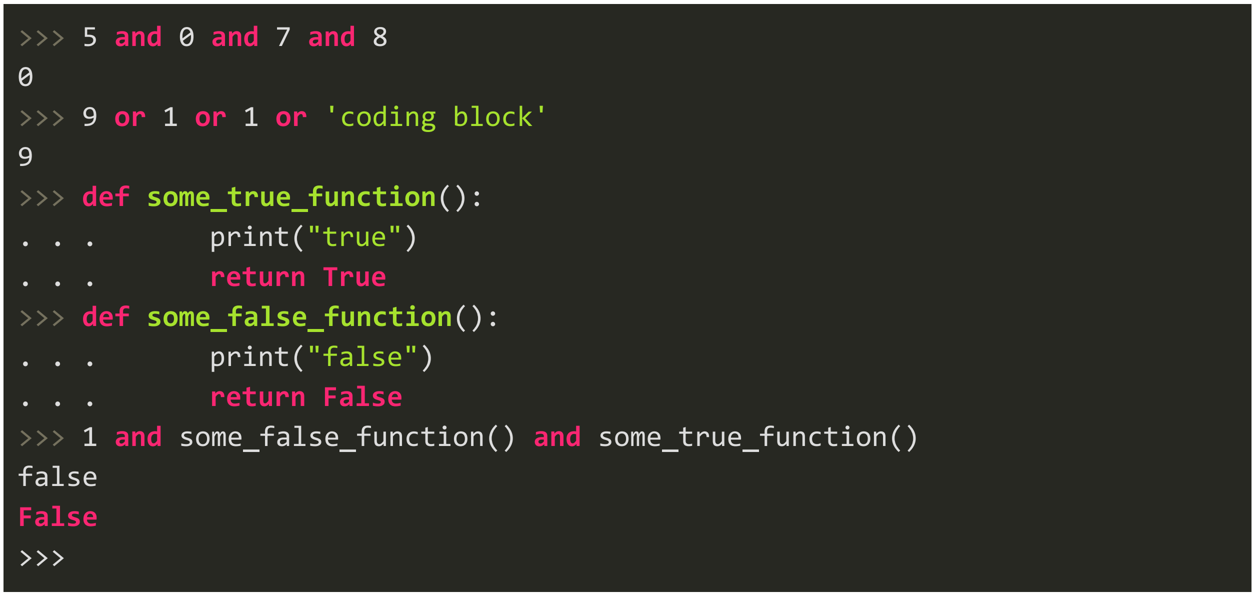
Symbol	Task
|	or
&	and
^	xor
~	negation

When more than one operator appears in an expression, the order of evaluation depends on the **rules of precedence**. For mathematical operators, Python follows mathematical convention. The acronym **PEMDAS** is a useful way to remember the rules:

- **P**arentheses have the highest precedence and can be used to force an expression to evaluate in the order you want. Since expressions in parentheses are evaluated first, 2 * (3-1) is 4, and (1+1)**(5-2) is 8. You can also use parentheses to make an expression easier to read, as in (minute * 100) / 60, even if it doesn't change the result.
- **E**xponentiation has the next highest precedence, so 2**1+1 is 3, not 4, and 3*1**3 is 3, not 27.
- **M**ultiplication and **D**ivision have the same precedence, which is higher than **A**ddition and **S**ubtraction, which also have the same precedence. So 2*3-1 is 5, not 4, and 6+4/2 is 8, not 5.
-
- Operators with the same precedence are evaluated from left to right. So in the expression 5-3-1 is 1, not 3 because the 5-3 happens first and then 1 is subtracted from 2.

Short Circuiting using logical operators.

When Python detects that there is nothing to be gained by evaluating the rest of a logical expression, it stops its evaluation and does not do the computations in the rest of the logical expression. When the evaluation of a logical expression stops because the overall value is already known, it is called **short-circuiting** the evaluation.



```
>>> 5 and 0 and 7 and 8
0
>>> 9 or 1 or 1 or 'coding block'
9
>>> def some_true_function():
. . .       print("true")
. . .       return True
>>> def some_false_function():
. . .       print("false")
. . .       return False
>>> 1 and some_false_function() and some_true_function()
false
False
>>>
```