_All the IPython Notebooks in **Python Introduction** lecture series by **[Dr. Milaan Parmar](https://www.linkedin.com/in/milaanparmar/)** are available @ **[GitHub](https://github.com/milaan9/01_Python_Introduction)**_ </small>

Python Operators

Python can be used like a calculator. Simply type in expressions to get them evaluated.

What are operators in python?

Operators are special **symbols** in Python that carry out **arithmetic** or **logical computation**. The value that the operator operates on is called the **operand**.

For example:

```
>>>6+3

9
```

Here, `+` is the operator that performs addition. `2` and `3` are the operands and `5` is the output of the **operation**.

```
In [1]: 6+3
Out[1]: 9
```

1. Arithmatic Operators

Arithmetic operators are used to perform **mathematical operations** like **addition**, **subtraction**, **multiplication** etc.

Symbol	Task Performed	Meaning	Example
+	Addition	add two operands or unary plus	**x + y** or **+2**
-	Subtraction	substract right operand from the left or unary minus	**x - y** or **-2**
*	Multiplication	Multiply two operands	**x * y**
/	Division	Divide left operand by the right one (always results into float)	**x / y**
%	Modulus (remainder)	remainder of the division of left operand by the right	**x % y** (remainder of **x/y**)
//	Integer/Floor division	division that results into whole number adjusted to the left in the number line	**x // y**
`**`	Exponentiation (power)	left operand raised to the power of right	**x ** y** (**x** to the power **y**)

As expected these operations generally promote to the most general type of any of the numbers involved i.e. int -> float -> complex.

Example : Arithmetic operators in Python

In [2]:

```
print('Addition: ', 1 + 2)
print('Subtraction: ', 2 - 1)
print('Multiplication: ', 2 * 3)
print ('Division: ', 4 / 2)                          # Division in python gives floating numbe
r
print('Division: ', 6 / 2)
print('Division: ', 7 / 2)
print('Division without the remainder: ', 7 // 2)    # gives without the floating number or wi
thout the remaining
print('Modulus: ', 3 % 2)                            # Gives the remainder
print ('Division without the remainder: ',7 // 3)
print('Exponential: ', 3 ** 2)                       # it means 3 * 3
```

```
Addition:  3
Subtraction:  1
Multiplication:  6
Division:  2.0
Division:  3.0
Division:  3.5
Division without the remainder:  3
Modulus:  1
Division without the remainder:  2
Exponential:  9
```

In [3]:

```
x = 16
y = 3

print('x + y =',x+y) # 19
print('x - y =',x-y) # 13
print('x * y =',x*y) # 48
print('x / y =',x/y) # 5.333
print('x // y =',x//y) # 519
```

```
x + y = 19
x - y = 13
x * y = 48
x / y = 5.333333333333333
x // y = 5
```

In [4]:

```
1+2+3
```

Out[4]:

```
6
```

In [5]:

```
7-1
```

Out[5]:

```
6
```

In [6]:

```
6 * (3+0j) * 1.0
```

Out[6]:

```
(18+0j)
```

In [7]:

```
5/6
```

Out[7]:

```
0.8333333333333334
```

In many languages (and older versions of python) $\frac{1}{2}=0$ (truncated division). In Python 3 this behaviour is captured by a separate operator that rounds down: (i.e., **a // b$=\lfloor \frac{a}{b}\rfloor$**)

In [8]:

```
5//6.0
```

Out[8]:

```
0.0
```

In [9]:

```
15%10
```

Out[9]:

```
5
```

```
In [10]: 3 ** 2      # it means 3 * 3

Out[10]: 9
```

Python natively allows (nearly) infinite length integers while floating point numbers are double precision numbers:

```
In [11]: 22**600

Out[11]: 2841898045385262226038838564947097313112181583622932247004455439407966437769117881295085772463
         0992925806959846789600828759605386308210974680194772228078372003264981819846445237535834861912
         0021135797251857019557032565638659667745421115194468559021541269027438746788486392649581405327
         5166587183170114538584652187798809721406125515041497010821641793674841985707342300035283558
         2654564422379477991172391711584521373992132782040900546070854794244523490649888058841127675512
         0002108963510955276534630522965966177951626038063994937181228938844252668098515385045718428663
         5477881221737987047175109793197498340834190043552220120437158551236152496820395878444312462
         6901761272829171071882138884062727740718986350978992933910240665511578689969682990203228266698
         342689711175020883915283057368851283198460208536057248586137 6
```

```
In [12]: 22.0**600

---------------------------------------------------------------------------
OverflowError                             Traceback (most recent call last)
<ipython-input-12-bfc5aa62a0ff> in <module>
----> 1 22.0**600

OverflowError: (34, 'Result too large')
```

2. Comparison/Relational operators

Comparison operators are used to **compare values**. It either returns **True** or **False** according to the **condition**.

Symbol	Task Performed	Meaning	Example
>	greater than	True if left operand is greater than the right	**x > y**
<	less than	True if left operand is less than the right	**x < y**
==	equal to	True if both operands are equal	**x == y**
!=	not equal to	True if both operands are not equal	**x != y**
>=	greater than or equal to	True if left operand is greater than or equal to the right	**x >= y**
<=	less than or equal to	True if left operand is less than or equal to the right	**x <= y**

Note the difference between `==` (equality test) and `=` (assignment)

Example : Comparison operators in Python

In [13]:

```
print(6 > 3)                          # True, because 3 is greater than 2
print(6 >= 3)                         # True, because 3 is greater than 2
print(6 < 3)                          # False,  because 3 is greater than 2
print(3 < 6)                          # True, because 2 is less than 3
print(3 <= 6)                         # True, because 2 is less than 3
print(6 == 3)                         # False, because 3 is not equal to 2
print(6 != 3)                         # True, because 3 is not equal to 2
print(len("apple") == len("avocado"))  # False
print(len("apple") != len("avocado"))  # True
print(len("apple") < len("avocado"))   # True
print(len("banana") != len("orange"))  # False
print(len("banana") == len("orange"))  # True
print(len("tomato") == len("potato"))  # True
print(len("python") > len("coding"))   # False
```

```
True
True
False
True
True
False
True
False
True
True
False
True
True
False
```

In [14]:

```
x = 30
y = 22

print('x > y is',x>y)   # False
print('x < y is',x<y)   # True
print('x >= y is',x>=y) # False
print('x <= y is',x<=y) # True
```

```
x > y is True
x < y is False
x >= y is True
x <= y is False
```

In [15]:

```
z = 3  # 3 is assign to variable z
z == 3 # 3 is equal to z
```

Out[15]:

```
True
```

In [16]:

```
z > 3
```

Out[16]:

```
False
```

Comparisons can also be chained in the mathematically obvious way. The following will work as expected in Python (but not in other languages like C/C++):

In [17]:

```
0.5 < z <= 1   # z == 3
```

Out[17]:

```
False
```

3. Logical/Boolean operators

Logical operators are the `and` , `or` , `not` operators.

Symbol	Meaning	Example
and	True if both the operands are true	**x and y**
or	True if either of the operand is true	**x or y**
not	True if operand are false (complements the operand)	**not x**

Example : Logical operators in Python

In [18]:

```
print('True == True: ', True == True)
print('True == False: ', True == False)
print('False == False:', False == False)
print('True and True: ', True and True)
print('True or False:', True or False)
```

```
True == True:  True
True == False:  False
False == False: True
True and True:  True
True or False: True
```

In [19]:

```
# Another way comparison

print('1 is 1', 1 is 1)                 # True  - because the data values are the same
print('1 is not 2', 1 is not 2)         # True  - because 1 is not 2
print('A in Milaan', 'A' in 'Milaan')   # True  - A found in the string
print('B in Milaan', 'B' in 'Milaan')   # False - there is no uppercase B
print('python' in 'python is fun')      # True  - because coding for all has the word coding
print('a in an:', 'a' in 'an')          # True
print('27 is 3 ** 3:', 27 is 3**3)      # True
```

```
1 is 1 True
1 is not 2 True
A in Milaan False
B in Milaan False
True
a in an: True
27 is 3 ** 3: True

<>:3: SyntaxWarning: "is" with a literal. Did you mean "=="?
<>:4: SyntaxWarning: "is not" with a literal. Did you mean "!="?
<>:9: SyntaxWarning: "is" with a literal. Did you mean "=="?
<>:3: SyntaxWarning: "is" with a literal. Did you mean "=="?
<>:4: SyntaxWarning: "is not" with a literal. Did you mean "!="?
<>:9: SyntaxWarning: "is" with a literal. Did you mean "=="?
<ipython-input-19-7c9145eb11e9>:3: SyntaxWarning: "is" with a literal. Did you mean "=="?
  print('1 is 1', 1 is 1)                 # True  - because the data values are the same
<ipython-input-19-7c9145eb11e9>:4: SyntaxWarning: "is not" with a literal. Did you mean "!="?
  print('1 is not 2', 1 is not 2)         # True  - because 1 is not 2
<ipython-input-19-7c9145eb11e9>:9: SyntaxWarning: "is" with a literal. Did you mean "=="?
  print('27 is 3 ** 3:', 27 is 3**3)      # True
```

In [20]:

```
print(6 > 3 and 5 > 3) # True  - because both statements are true
print(6 > 3 and 5 < 3) # False - because the second statement is false
print(6 < 3 and 5 < 3) # False - because both statements are false
print(6 > 3 or 5 > 3)  # True  - because both statements are true
print(6 > 3 or 5 < 3)  # True  - because one of the statement is true
print(6 < 3 or 5 < 3)  # False - because both statements are false
print(not 6 > 3)       # False - because 6 > 3 is true, then not True gives False
print(not True)        # False - Negation, the not operator turns true to false
print(not False)       # True
print(not not True)    # True
print(not not False)   # False
```

```
True
False
False
True
True
False
False
False
True
True
False
```

In [21]:

```
x = True
y = False

print('x and y is',x and y) # False
print('x or y is',x or y) # True
print('not x is',not x) # False
```

```
x and y is False
x or y is True
not x is False
```

In [22]:

```
True and (not(not False)) or (True and (not True))  # What will be output?

# True and (not(True)) or (True and (False))
# True and False or (False)
# False or False
# False
```

Out[22]:

```
False
```

Here is the **truth table (@ and, or, not) (https://github.com/milaan9/01_Python_Introduction/blob/main/Python_Keywords_List.ipynb)** for these operators.

4. Bitwise operators

Bitwise operators act on operands as if they were string of binary digits. It operates **bit by bit**, hence the name.

For example: 2 is `10` in binary and 7 is `111` .

In the table below: Let `x` = 10 (`0000 1010` in binary) and `y` = 4 (`0000 0100` in binary)

Operator	Meaning	Symbol	Task Performed	Example
`and`	Logical and	**&**	Bitwise And	**x & y** = 0 (`0000 0000`)
`or`	Logical or	**$\mid$**	Bitwise OR	**x | y** = 14 (`0000 1110`)
`not`	Not	**~**	Bitwise NOT	**~x** = -11 (`1111 0101`)
		^	Bitwise XOR	**x ^ y** = 14 (`0000 1110`)
		>>	Bitwise right shift	**x >> 2** = 2 (`0000 0010`)
		<<	Bitwise left shift	**x << 2** = 40 (`0010 1000`)

In [23]:

```
a = 2 #binary: 0010
b = 3 #binary: 0011
print('a & b =',a & b,"=",bin(a&b))
```

```
a & b = 2 = 0b10
```

In [24]:

```
5 >> 1

# 0 ➞ 0000 0101
#      0000 0010
# 0010 is 2 in decimal
```

Out[24]:

```
2
```

Explanation:

0000 0101 -> 5 (5 is 0101 in binary)

Shifting the digits by 1 to the right and zero padding that will be: 0 ➞ 0000 0101 = 0000 0010

0000 0010 -> 2

In [25]:

```
5 << 1

# 0000 0101  ⟵ 0
# 0000 1010
# 1010 is 10 in decimal
```

Out[25]:

```
10
```

Explanation:

0000 0101 -> 5

Shifting the digits by 1 to the left and zero padding will be: 0000 0101 ⟵ 0 = 0000 1010

0000 1010 -> 10

In [26]:

```
6 >> 2  # What will be output???
```

Out[26]:

```
1
```

In [27]:

```
print(not (True and False), "==", not True or not False)
#       TRUE                       ==      True
```

```
True == True
```

In [28]:

```
print (False and (not False) or (False and True), "==",not (True and (not False) or (not Tru
e)))
```

```
False == False
```

5. Assignment operators

Assignment operators are used in Python to **assign values** to **variables**.

`a = 5` is a simple assignment operator that assigns the value 5 on the right to the variable `a` on the left.

There are various compound operators in Python like a `+= 5` that adds to the variable and later assigns the same. It is equivalent to `a = a + 5`.

Symbol	Example	Equivalent to
=	x = 5	x = 5
+=	x += 5	x = x + 5
-=	x -= 5	x = x - 5
*=	x *= 5	x = x * 5
/=	x /= 5	x = x / 5
%=	x %= 5	x = x % 5
//=	x //= 5	x = x // 5
`**=`	x **= 5	x = x ** 5
&=	x &= 5	x = x & 5
|=	x |= 5	x = x | 5
^=	x ^= 5	x = x ^ 5
>>=	x >>= 5	x = x >> 5
<<=	x <<= 5	x = x << 5

The binary operators can be combined with assignment to modify a variable value. For example:

In [29]:

```
x = 1
x += 2 # add 2 to x
print("x is",x)
x <<= 2 # left shift by 2 (equivalent to x *= 4)
print('x is',x)
x **= 2 # x := x^2
print('x is',x)
```

```
x is 3
x is 12
x is 144
```

6. Special operators

Python language offers some special types of operators like the identity operator or the membership operator. They are described below with examples.

1. Identity operators

`is` and `is not` are the identity operators in Python. They are used to check if two values (or variables) are located on the same part of the **memory**. Two variables that are equal does not imply that they are **identical**.

Symbol	Meaning	Example
`is`	True if the operands are identical (refer to the same object)	**x is True**
`is not`	True if the operands are not identical (do not refer to the same object)	**x is not True**

Example : Identity operators in Python

In [30]:

```python
x1 = 6
y1 = 6
x2 = 'Hello'
y2 = 'Hello'
x3 = [1,2,3] # list
y3 = [1,2,3] # list

# Output: False
print(x1 is not y1)

# Output: True
print(x2 is y2)

# Output: False because two list [] can never be equal
print(x3 is y3)
```

```
False
True
False
```

Explanation:

Here, we see that **`x1`** and **`y1`** are integers of same values, so they are equal as well as identical. Same is the case with **`x2`** and **`y2`** (strings).

But **`x3`** and **`y3`** are list. They are equal but not identical. It is because interpreter locates them **separately in memory** although they are equal.

2. Membership operators

`in` and `not in` are the membership operators in Python. They are used to test whether a value or variable is found in a **sequence** (string (https://github.com/milaan9/02_Python_Datatypes/blob/main/002_Python_String.ipynb), list (https://github.com/milaan9/02_Python_Datatypes/blob/main/003_Python_List.ipynb), tuple (https://github.com/milaan9/02_Python_Datatypes/blob/main/004_Python_Tuple.ipynb), set (https://github.com/milaan9/02_Python_Datatypes/blob/main/006_Python_Sets.ipynb) and dictionary (https://github.com/milaan9/02_Python_Datatypes/blob/main/005_Python_Dictionary.ipynb)).

In a dictionary we can only test for presence of **key, not the value**.

Symbol	Meaning	Example
`in`	True if value/variable is found in sequence	**5 in x**
`not in`	True if value/variable is not found in sequence	**5 not in x**

Example : Membership operators in Python

In [31]:

```
x = 'Hello world'
y = {1:'a',2:'b'} # dictionary 1 is key and 'a' is element. So we access element without its key.

# Output: True
print('H' in x)  # Do we have 'H' in 'Hello World' ?

# Output: True
print('hello' not in x)  # Do we have 'hello' in 'Hello World' ?

# Output: True
print(1 in y)

# Output: False because we cannot identify 'a' without its key hence it is Flase.
print('a' in y)
```

```
True
True
True
False
```

Explanation:

Here, `' H '` is in `x` but `' hello '` is not present in `x` (remember, Python is case sensitive). Similary, `1` is key and `' a '` is the value in dictionary y. Hence, `'a'in y` returns `False` .

💻 Exercises ➞ Operators

1. Declare your age as integer variable
2. Declare your height as a float variable
3. Declare a variable that store a complex number
4. Write a code that prompts the user to enter base and height of the triangle and calculate an area of this triangle (area = 0.5 x b x h).

```
Enter base: 20
    Enter height: 10
    The area of the triangle is 100
```

1. Write a code that prompts the user to enter side a, side b, and side c of the triangle. Calculate the perimeter of the triangle (perimeter = a + b + c).

```
Enter side a: 5
Enter side b: 4
Enter side c: 3
The perimeter of the triangle is 12
```

1. Get length and width of a rectangle using prompt. Calculate its area (**area = length x width**) and perimeter (**perimeter = 2 x (length + width)**)
2. Get radius of a circle using prompt. Calculate the area (**area = pi x r x r**) and circumference (**c = 2 x pi x r**) where pi = 3.14.
3. Calculate the slope, x-intercept and y-intercept of $y = 2x -2$
4. Slope is ($m = (y2-y1)/(x2-x1)$). Find the slope and [Euclidean distance (https://en.wikipedia.org/wiki/Euclidean_distance#:~:text=In%20mathematics%2C%20the%20Euclidean%20distance,beir](https://en.wikipedia.org/wiki/Euclidean_distance#:~:text=In%20mathematics%2C%20the%20Euclidean%20distance,beir) between point (2, 2) and point (6,10)
5. Compare the slopes in tasks 8 and 9.
6. Calculate the value of y ($y = x^2 + 6x + 9$). Try to use different x values and figure out at what x value y is going to be 0.
7. Find the length of `'python'` and `'datascience'` and make a falsy comparison statement.
8. Use `and` operator to check if `on` is found in both `python` and `cannon`
9. `I hope this course is not full of jargon`. Use `in` operator to check if `jargon` is in the sentence.
10. There is no `on` in both `python` and `cannon`
11. Find the length of the text `python` and convert the value to float and convert it to string
12. Even numbers are divisible by 2 and the remainder is zero. How do you check if a number is even or not using python?
13. Check if the floor division of 7 by 3 is equal to the int converted value of 2.7.
14. Check if type of **"10"** is equal to type of 10
15. Check if int(**"9.6"**) is equal to 10
16. Write a code that prompts the user to enter hours and rate per hour. Calculate pay of the person?

```
Enter hours: 40
Enter rate per hour: 30
Your weekly earning is 1200
```

1. Write a script that prompts the user to enter number of years. Calculate the number of seconds a person can live. Assume a person can live hundred years

```
Enter number of years you have lived: 100
You have lived for 3153600000 seconds.
```

1. Write a Python code that displays the following table

```
1 2 3 4 5
2 4 6 8 10
3 6 9 12 15
4 8 12 16 20
5 10 15 20 25
```

In [ ]:

```
```