Operators & Comments in Python

- Notes By Jagdish Chavan
- For more code and Projects follow Github ('https://github.com/JagdishChavan081')|Linkedin ('https://www.linkedin.com/in/jagdishchavan')

Python Operators

Operators are used to perform operations on variables and values.

Python divides the operators in the following groups:

- Arithmetic operators
- Assignment operators
- Comparison operators
- Logical operators

- Identity operators
- Membership operators

- Bitwise operators

PEMDAS = BODMAS

Precedence	Associativity	Operator	Description
18	Left-to-right	()	Parentheses (grouping)
17	Left-to-right	*f*(args...)	Function call
16	Left-to-right	*x*[index:index]	Slicing
15	Left-to-right	*x*[index]	Array Subscription
14	Right-to-left	**	Exponentiation
13	Left-to-right	~*x*	Bitwise not
12	Left-to-right	+*x* -*x*	Positive, Negative
11	Left-to-right	* / %	Multiplication Division Modulo
10	Left-to-right	+ -	Addition Subtraction
9	Left-to-right	<< >>	Bitwise left shift Bitwise right shift
8	Left-to-right	&	Bitwise AND
7	Left-to-right	^	Bitwise XOR
6	Left-to-right	|	Bitwise OR
5	Left-to-right	in, not in, is, is not, <, <=, >, >=, <>, == !=	Membership Relational Equality Inequality
4	Left-to-right	not *x*	Boolean NOT
3	Left-to-right	and	Boolean AND
2	Left-to-right	or	Boolean OR
1	Left-to-right	lambda	Lambda expression

Python Arithmetic Operators

In [ ]:
```
#Addition
x = 5
y = 3

print(x + y)
```

```
8
```

In [ ]:
```
#Subtraction

x = 5
y = 3

print(x - y)
```

```
2
```

In [ ]:
```
#Multiplication

x = 5
y = 3

print(x * y)
```

```
15
```

In [ ]:
```
#Division
x = 12
y = 3

print(x / y)
```

```
4.0
```

In [ ]:
```
#Modulus        (x % y)
x = 5
y = 2

print(x % y)
```

```
1
```

In [ ]:
```
#Exponentiation(x ** y)
x = 2
y = 5

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

print(pow(3,2))
```

```
32
9
```

In [ ]:
```
#Floor division(x // y)

x = 15
y = 2

print(x // y)

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

```
7
```

Python Assignment Operators

Assignment operators are used to assign values to variables:

In [ ]:
```
# '=',  x = 5
x = 5

print(x)
```

```
5
```

In [ ]:
```
# '+=', x += 3, x = x + 3
x = 5

x = x+3

x += 3

print(x)
```

```
11
```

In [ ]:
```
# '-=', x -= 3, x = x - 3

x = 5

x -= 3

print(x)
```

```
2
```

In [ ]:
```
# '*=', x *= 3, x = x * 3

x = 5

x *= 3

print(x)
```

```
15
```

In [ ]:
```
# '/=', x /= 3, x = x / 3

x = 5

x /= 3

print(x)
```

```
1.6666666666666667
```

In [ ]:
```
# '%=', x %= 3, x = x % 3

x = 5

x%=3

print(x)
```

```
2
```

In [ ]:
```
# '//=',        x //= 3,        x = x // 3

x = 5

x//=3

print(x)
```

```
1
```

In [ ]:
```
# '**=',        x **= 3,        x = x ** 3

x = 5

x **= 3

print(x)
```

```
125
```

In [ ]:
```
# '&='  x &= 3  x = x & 3

x = 5

x &= 3

print(x)
```

```
1
```

In [ ]:
```
# '|=', x |= 3, x = x | 3

x = 5

x |= 3

print(x)
```

```
7
```

In [ ]:
```
# '^=', x ^= 3, x = x ^ 3

x = 5

x ^= 3

print(x)
```

```
6
```

In [ ]:
```
# '>>=',        x >>= 3,        x = x >> 3

x = 5

x >>= 3

print(x)
```

```
0
```

In [ ]:
```
# '<<=',        x <<= 3,        x = x << 3

x = 5

x <<= 3

print(x)
```

```
40
```

Python Comparison Operators

Comparison operators are used to compare two values:

In [ ]:
```
#Equal(==)
x = 5
y = 3

print(x == y)
```

```
False
```

In [ ]:
```
#Not equal(!=)
x = 5
y = 3

print(x != y)
```

```
True
```

In [ ]:
```
#Greater than(>)
x = 5
y = 3

print(x > y)
```

```
True
```

In [ ]:
```
#Less than(<)
x = 5
y = 3

print(x < y)
```

```
False
```

In [ ]:
```
#Greater than or equal to(>=)
x = 5
y = 3

print(x >= y)
```

```
True
```

In [ ]:
```
#Less than or equal to(<=)
x = 5
y = 3

print(x <= y)
```

```
False
```

Python Logical Operators

Logical operators are used to combine conditional statements:

In [ ]:
```
#and
#Returns True if both statements are true
x = 5

print(x > 3 and x < 10)
```

```
True
```

In [ ]:
```
#or
#Returns True if one of the statements is true
x = 5

print(x > 3 or x < 4)
```

```
True
```

In [ ]:
```
#not
#Reverse the result, returns False if the result is true
x = 5

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

```
False
```

Python Identity Operators

Identity operators are used to compare the objects, not if they are equal, but if they are actually the same object, with the same memory location:

In [ ]:
```
#is
#Returns True if both variables are the same object

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 s
ame content

print(x == y)

# to demonstrate the difference betweeen "is" and "==": this comparison return
s True because x is equal to y
```

```
True
False
True
```

In [ ]:
```
#is not
#Returns True if both variables are not the same object
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 sa
me content

print(x != y)

# to demonstrate the difference betweeen "is not" and "!=": this comparison re
turns False because x is equal to y
```

```
False
True
False
```

Python Membership Operators

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

In [ ]:
```
#in
#Returns True if a sequence with the specified value is present in the object
x = ["apple", "banana"]

print("banana" in x)

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

```
True
```

In [ ]:
```
#not in
#Returns True if a sequence with the specified value is not present in the obj
ect
x = ["apple", "banana"]

print("pineapple" not in x)

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

```
True
```

Python Bitwise Operators

Bitwise operators are used to compare (binary) numbers:

- & 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 leftmost bits fall off
-
  > Signed right shift Shift right by pushing copies of the leftmost bit in from the left, and let the rightmost bits fall off

Mathematical Function

- We can use built-in functions given in Python to perform various advanced operations.
- all The advance operations like square root we can develop our own logic or use sqrt() function available in **math** module.

In [ ]:
```
import math
```

In [ ]:
```
from math import  sqrt

x = math.sqrt(16)
x
```

Out[ ]:
```
4.0
```

In [ ]:
```
# celin(x) raises x value to nect higher integer
x = 4.5
y = math.ceil(x)
y
```

Out[ ]:
```
5
```

In [ ]:
```
#floor(x):- decreases x value to previous integer
x = 4.5
y = math.floor(x)
y
```

Out[ ]:
```
4
```

In [ ]:
```
# degrees(x): angle radian -> degree
x = 3.14159
y = math.degrees(x)
y
```

Out[ ]: 179.9998479605043

In [ ]:
```
#radians(x): degree -> radian
x = 179.9
y = math.radians(x)
y
```

Out[ ]: 3.139847324337799

In [ ]:
```
# sin(x), radians
x = 0.5
y = math.sin(x)
y
```

Out[ ]: 0.479425538604203

In [ ]:
```
#cos(x), radian
x = 0.5
y = math.cos(x)
y
```

Out[ ]: 0.8775825618903728

- tan(x)
- exponent->exp(x)
- absolute value ->fabs(x)
- factorial->factorial(x)
- fmod(x,y)
- fsum(x,y)
- modf(x)
- log10(x)
- log[x,[,base]
- sqrt(x)
- pow(x,y)
- gcd(x,y)
- trunc(x)
- isinf(x)
- isnan(x)

In [ ]:
```
import math
x = math.pi
x
```

Out[ ]: 3.141592653589793

In [ ]:
```
import math
x = math.e
x
```

Out[ ]:
```
2.718281828459045
```

In [ ]:
```
import math
x = math.inf
print(x)
```

```
inf
```

In [ ]:
```
import math
x = -math.inf
print(x)
```

```
-inf
```

In [ ]:
```
import math
x = math.nan
print(x)
```

```
nan
```

In [ ]:
```
1095, 1055, 1065

1005, 1015, 1025
```