Logical Operators

1. AND

2. OR

3. NOT

In [3]:
```
a=5
b=6
```

In [5]:
```
a<b and b>a # AND = Returns "true" only if both operands are "true."
```

Out[5]: True

In [7]:
```
a>b and b<a
```

Out[7]: False

In [9]:
```
print(a)
print(b)
```

```
5
6
```

In [13]:
```
a>b or b>a # OR = Returns "true" if at least one of the operands is "true."
```

Out[13]: True

In [15]:
```
a>b or b<a # Here both of them are false. so it gives "False"
```

Out[15]: False

In [23]:
```
x = True
x
```

Out[23]: True

In [25]:
```
not x # NOT = IT negates the operand, meaning it returns the opposite value.
      # Here it is true but it gives as a false.
```

Out[25]: False

In [29]:
```
x = False # Here it is false but it gives as a true
x .
```

Out[29]: False

In [31]:
```
not x
```

Out[31]: True

Number System

1. Binary

2. Octal

3. Hexa Decimal

In [ ]:
```
# Binary() = The base value is (0-1) i.e., "2"
# Octal() = The base value is (0-7) i.e., "8"
# Hexa Decimal() = The base value is (0-9 and a-f) i.e., "16". Here a=10, b=12 e
# To check the Ip address of your device, open CMD and Type "ipconfig"
```

1. Binary()

In [36]:
```
bin(10) # The output is generated with the indication od binary i.e., "0b"
```

Out[36]: '0b1010'

In [40]:
```
bin(20) # Here it converts "20" into Binary format.
        # there are 2 methods to understand this in practical i.e.,
        # 1. LCM
        # 2. Use "64 32 16 8 4 2 1" format
        # this is to convert integer into a binary number.
```

Out[40]: '0b10100'

In [42]:
```
int(0b110) # Use this method to convert binary number into integer number
```

Out[42]: 6

In [44]:
```
int(0b111) # The other method to achieve this is, 1 x 2**0 + 1 x 2**1 + 1 x 2**2
           # This is binary so we take "2" and from left you need to add powers
           # Always prefer "8 4 2 1" It is also called as BCD code. It is very e
```

Out[44]: 7

2. Ocatl()

In [47]:
```
int(0o111) # 1 x 8**0  + 1 x 8**1 + 1 x 8**2
```

Out[47]: 73

In [51]:
```
oct(10)
```

Out[51]: '0o12'

In [55]:
```
0o10
```

Out[55]: 8

3. Hexa Decimal()

In [58]:
```
hex(30) # E = 14
```

Out[58]:
```
'0x1e'
```

Swapping

In [61]:
```
a=5
b=6
```

In [63]:
```
a,b=b,a # 1. Best and easy method. without taking 3rd variable
```

In [65]:
```
print(a)
print(b)
```

```
6
5
```

In [67]:
```
temp=a # 2nd method by taking 3rd variable as "temp"
a=b
b=temp
```

In [69]:
```
print(a)
print(b)
```

```
5
6
```

In [71]:
```
a1=7
b1=8
```

In [73]:
```
a1 = a1 + b1 # 3rd method and also called swapping formulae
b1 = a1 - b1
a1 = a1 - b1
```

In [75]:
```
print(a1)
print(b1)
```

```
8
7
```

In [77]:
```
a2=10
b2=20
```

In [79]:
```
a2 = a2 ^ b2 # 4th method
b2 = a2 ^ b2
a2 = a2 ^ b2
```

In [81]:
```
print(a2)
print(b2)
```

```
20
10
```

Bitwise Operators

1. NOT (~)

2. AND (&)

3. OR (|)

4. XOR (^)

5. left Shift (<<)

6. Right Shift (>>)

1. NOT (~)= The bitwise NOT operator inverts the bits of an integer. It changes all 0s to 1s and all 1s to 0s.

In [3]:
```
~32 # "~" It is called as bitwise NOT operator inverts the bits of an integer. I
    # It reverses the binary number and gives the output.
```

Out[3]: -33

In [5]:
```
~10 # 10 = 1010 and Applies the bitwise NOT (~) gives 0101, so 0101 = 11.
```

Out[5]: -11

In [7]:
```
~4 # "~" It is called as compliment.
```

Out[7]: -5

2. AND (&) = Performs a bit-by-bit AND operation.

In [10]:
```
10 & 12 # 10 = 1010 # It gives "1" only if both corresponding bits are "1"; othe
        # 12 = 1100
       # o/p = 1000 =8
```

Out[10]: 8

In [12]:
```
4 & 6 # 4 = 0100
      # 6 = 0110
    # o/p = 0100 = 4
```

Out[12]: 4

3. OR (|) = Performs a bit-by-bit OR operation.

In [16]:
```
2 | 6 # 2 = 0010 # It gives "1" if at least one of the corresponding bits is "1"
      # 6 = 0110
    # o/p = 0110 = 6
```

Out[16]: 6

In [18]:
```
1 | 0
```

Out[18]: 1

In [20]:
```
10 | 20
```

Out[20]: 30

4. XOR(^) = Performs a bit-by-bit exclusive OR (XOR) operation.

In [25]:
```
4^8 # 4 = 0100 # It gives "1" if the corresponding bits are different; it's 0 if
   #  8 = 1000
  # o/p = 1100 = 12
```

Out[25]: 12

In [27]:
```
10^15 # 10 = 1010 # It is purely opposite to the OR operation.
     #  15 = 1111
     # o/p = 0101 = 5
```

Out[27]: 5

5. Leftshift (<<) = Shifts the bits of an integer to the left by a specified number of positions.

In [38]:
```
10<<2 # 10 = 1010. Now, we have given "2". so it adds two zeros at the last.
      # then it becomes "101000" = 40
```

Out[38]: 40

In [42]:
```
5<<2 # It depends on how many values you want to add.
```

Out[42]: 20

6. Rightshift (>>) = Shifts the bits of an integer to the right by a specified number of positions.

In [49]:
```
8>>2 # 4 = "1000" Now, we have given "2". so it removes two zeros at the last.
    # now it has only "10" . the answer is "2"
```

Out[49]: 2

In [53]:
```
20>>3 # 20 = 10100
     #  it removes "100" then the output is "2"
```

Out[53]: 2

Math Module

In [ ]:
```
# You need to use "import math" otherwise python won't recognize the math functi
```

In [1]:
```
import math # Math is a module
```

In [9]:
```
x = sqrt(25) # we didn't give "math" function. so it won't recognize.
x
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Cell In[9], line 1
----> 1 x = sqrt(25)
      2 x

NameError: name 'sqrt' is not defined
```

In [11]:
```
x = math.sqrt(25) # By default it gives float value.
x
```

Out[11]: 5.0

In [13]:
```
x1 = math.sqrt(9)
x1
```

Out[13]: 3.0

In [43]:
```
import math as m # you can simply import math function and continue with "m"
m.sqrt(81)
```

Out[43]: 9.0

In [49]:
```
from math import pow # you can also write like this using "from" function.
pow(2,3)
```

Out[49]: 8.0

In [ ]:
```
from math import * # instead writing sqrt, pow etc function. we can simply assig
```

In [51]:
```
round(pow(2,3)) # To return a floating-point number that is a rounded version of
                # Simply it gives int values of the output.
```

Out[51]: 8

In [55]:
```
pow(4,3)
```

Out[55]: 64.0

In [57]:
```
round(pow(4,3))
```

Out[57]: 64

Floor() = it rounds value down to the nearest whole number.

In [18]:
```
math.floor(2.9) # imagine floor as a building floor where the floor is always do
```

Out[18]: 2

In [20]:
```
math.floor(2.1)
```

Out[20]: 2

Ceil() = It rounds value up to the nearest whole number.

In [23]:
```
math.ceil(2.9) # imagine it as a ceiling of the building where it always lies on
```

Out[23]: 3

In [25]:
```
math.ceil(2.1)
```

Out[25]: 3

pow() = pow(x, y) returns x raised to the power of y.

In [28]:
```
math.pow(3,2)
```

Out[28]: 9.0

In [30]:
```
math.pow(2,4)
```

Out[30]: 16.0

In [36]:
```
math.pow(4,4)
```

Out[36]: 256.0

In [38]:
```
math.pi # constant value
```

Out[38]: 3.141592653589793

In [41]:
```
math.e # It's the base of the natural logarithm and appears in many mathematical
```

Out[41]: 2.718281828459045

Input Function [v.important concept]

In [60]:
```
x = input() # console is waiting for user to enter input.
y = input() # Observe closely it's not adding the numbers because, they are in s
z = x+y     # 1st way of writing input
print(z)
```

```
55
```

In [62]:
```
type(x)
```

Out[62]: str

In [64]:
```
a1 = input("Enter the 1st number :") # string format so it's conactenating the n
b1 = input("Enter the 2nd number :")
```

```
c1 = a1+b1
print(c1)
```

```
1010
```

In [70]:
```
x1 = input('Enter the 1st number') #whenever you works in input function it alwa
a1 = int(x1)
y1 = input('Enter the 2nd number') # it wont understand as arithmetic operator
b1 = int(y1) # It takes too much memory. so we need to reduce it.
z1 = a1 + b1
print(z1)
```

```
22
```

In [66]:
```
a1 = int(input("Enter the 1st number :")) # we have declared at first as "int"
b1 = int(input("Enter the 2nd number :"))
c1 = a1+b1
print(c1)
```

```
20
```

In [68]:
```
type(a1)
```

Out[68]:
```
int
```

char()

In [75]:
```
ch = input("Enter a char")
ch
```

Out[75]:
```
'ravi'
```

In [77]:
```
print(ch[0])
```

```
r
```

In [79]:
```
ch = input("Enter a char")[2] # we can also write like this and also you can app
ch
```

Out[79]:
```
'v'
```

Eval()

In [84]:
```
result = eval(input("Enter an expr")) # This function takes a string as input an
result
```

Out[84]:
```
13
```

In [86]:
```
result = (input("Enter an expr")) # It returns the input as output.
result # so eval functioon evaluates the expression and gives it as output.
```

Out[86]:
```
'2*3+4'
```

In [ ]:
```
# You can do this expressions using command prompt. And the notes is present in
```

In [ ]:
```

```