"OR" OPERATOR

In [ ]:
```
"""If the two are more relational expressions connected logical operator "or"
if the initial realation expression is True then PVM will not evaluation is
called short circuit evaluation"""
```

In [3]:
```
True or False
```

Out[3]: True

In [5]:
```
False or True
```

Out[5]: True

In [7]:
```
False or False
```

Out[7]: False

In [9]:
```
True or True
```

Out[9]: True

In [ ]:
```
"""Short-Circuit Evaluation (for or)
When multiple relational expressions are connected using the or operator, Python ev
If the first expression is True, Python Virtual Machine (PVM) does not evaluate the
This is because the final result is already guaranteed to be True.
This behavior is called Short-Circuit Evaluation."""
```

In [11]:
```
10>2 or 20>30
# Short circuit Rule: In an OR expression, if the first condition is True,
# Python does not evaluate the second one.
```

Out[11]: True

In [13]:
```
10>20 or 30>40
```

Out[13]: False

In [15]:
```
10>20 or 40>20 or 150>100 #SCR
```

Out[15]: True

In [17]:
```
10>20 or 40>30 or 50>60 #SCR
```

Out[17]: True

In [22]:
```
10 or 20
```

Out[22]: 10

In [24]:
```
0 or 20
```

Out[24]: 20

In [26]:
```
20 or 0
```

Out[26]: 20

In [28]:
```
10 or 20 or 30
```

Out[28]: 10

In [ ]:
```
"""Short-Circuit Evaluation (for or)
When multiple relational expressions are connected using the or operator,
Python evaluates them from left to right.
If the first expression is True,
Python Virtual Machine (PVM) does not evaluate the remaining expressions.
This is because the final result is already guaranteed to be True.
This behavior is called Short-Circuit Evaluation."""
```

In [30]:
```
"Python" or "Java" or "c"
```

Out[30]: 'Python'

In [32]:
```
"Python" or True or False
```

Out[32]: 'Python'

In [34]:
```
10 or 20 and 40
```

Out[34]: 10

In [36]:
```
20 and 40 or 30
```

Out[36]: 40

Special points about 'and ' 'or' operators

In [2]:
```
10 or 20 and 40 # We much first evaluate 'and' later 'or'
```

Out[2]: 10

In [4]:
```
20 and 40 or 30
```

Out[4]: 40

In [ ]:
```
# 15:09): We must first evaulate "and" later evaulate "or"
```

In [4]:
```
10 and 30 and 20 or 50 and 60 or 70
```

Out[4]: 20

'not' operator:

```
In [11]: not True
Out[11]: False

In [13]: not False
Out[13]: True

In [29]: not 10
Out[29]: False

In [17]: not "python"
Out[17]: False

In [19]: not 0
Out[19]: True

In [21]: not""
Out[21]: True

In [23]: 10>20
Out[23]: False

In [25]: not 10>20
Out[25]: True

In [27]: not 20>10
Out[27]: False
```

5. Bitwise Operators — (#37:00)

➤ Purpose of Bitwise Operators: To perform operations on integer data at the bit level, i.e., bit by bit.

➤ Applicability: Bitwise operators work only on integer data, not on floating-point values. This is because integer values provide certainty, whereas floating-point values may not.

➤ Execution Process of Bitwise Operators:

1. Conversion: Integer data is first converted into binary format.

2. Operation: Bitwise operations are then applied on binary data, and the result is also in binary format.
3. Display: By default, Python displays the result of bitwise operations in the decimal number system, since Python is a high-level language.

SLNO SYMBOL MEANING 1 << Bitwise Left Shift Operator 2 >> Bitwise Right Shift Operator 3 / Bitwise OR Operator 4 & Bitwise AND Operator 5 ~ Bitwise Complement Operator 6 ^ Bitwise XOR Operator

1. << Bitwise Left Shift Operator (<<)

"""The execution process of the Bitwise Left Shift Operator (<<) is:

👉 It moves all bits towards the left by the specified number of positions. 👉 The rightmost positions are filled with zeros, depending on how many bits are shifted. 👉 In decimal, this is equivalent to multiplying the number by 2^n."""

In [4]:
```
a=10
b=a<<3
print(b)
```

```
80
```

In [6]:
```
print(4<<3)
```

```
32
```

In [11]:
```
print(10.3 << 2)
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[11], line 1
----> 1 print(10.3 << 2)

TypeError: unsupported operand type(s) for <<: 'float' and 'int'
```

In [13]:
```
print(4 << -1)
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
Cell In[13], line 1
----> 1 print(4 << -1)

ValueError: negative shift count
```

2. >> Bitwise Right Shift Operator (>>):

In [ ]:
```
"""Bitwise Right Shift Operator (>>)
The right shift operator moves all the bits of a number to the right side by a
specified number of positions.
The leftmost bits (most significant bits) depend on the type of number:
For positive integers → they are filled with 0s.
```

```
For negative integers → they are filled with 1s (sign extension).
The rightmost bits that are shifted out are discarded"""
```

In [18]:
```
print(10>>3)
```

```
1
```

In [20]:
```
print(20>>2)
```

```
5
```

In [22]:
```
print(20>>4)
```

```
1
```

In [24]:
```
print(80>>4)
```

```
5
```

In [26]:
```
print(80.4>>4.5)
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[26], line 1
----> 1 print(80.4>>4.5)

TypeError: unsupported operand type(s) for >>: 'float' and 'float'
```

In [28]:
```
#"""THE CLASS WILL CONTINUE IN THE NEXT SESSION"""
```

Out[28]:
```
'THE CLASS WILL CONTINUE IN THE NEXT SESSION'
```