1)

In [24]:
```
def math_operation(num1,num2):


    add = num1 + num2
    subs = num1 - num2
    mul = num1 * num2
    div = num1 / num2


    return{

        "Add": add,
        "Subs":subs,
        "Mul":mul,
        "Div":div,
        "Num1":num1,
        "Num2":num2
    }

var = math_operation(40,30)

print("First variable is", var['Num1'],"& second variable is", var['Num2'])
print("Addition:",var['Num1'],'+',var['Num2'],'=',var['Add'])
print("Substraction:",var['Num1'],'-',var['Num2'],'=',var['Subs'])
print("Multiplication:",var['Num1'],'*',var['Num2'],'=',var['Mul'])
print("Division:",var['Num1'],'/',var['Num2'],'=',var['Div'])
```

```
First variable is 40 & second variable is 30
Addition: 40 + 30 = 70
Substraction: 40 - 30 = 10
Multiplication: 40 * 30 = 1200
Division: 40 / 30 = 1.3333333333333333
```

2)

In [ ]:
```
'''
(i) `/` and `//` operators:

1. `/` (Division Operator): The `/` operator performs normal division between two operands and returns
the result as a floating-point number. It does not consider the data types of the operands and always
produces a floating-point result.

2. `//` (Floor Division Operator): The `//` operator performs floor division between two operands
and returns the result as an integer, discarding the fractional part of the division.
It is also known as integer division.

The main difference between `/` and `//` is that `/` always returns a floating-point result,
while `//` returns an integer result, truncating any fractional part.

(ii) `**` and `^` operators:

1. `**` (Exponentiation Operator): The `**` operator is used to raise a number to a power.
It calculates the exponentiation of the left operand with the right operand as the exponent.

2. `^` (Bitwise XOR Operator): The `^` operator is used for bitwise XOR (exclusive OR) operation on two
integers. It performs bitwise XOR on each corresponding bit of the binary representation of the operands.

The key difference between `**` and `^` is that `**` is used for exponentiation, while `^` is used for bitwise XOR operation.
They serve entirely different purposes in Python.
'''
```

In [25]:
```
result = 10 / 3
print(result)
```

```
3.3333333333333335
```

In [26]:
```
result = 10 // 3
print(result)
```

```
3
```

In [27]:
```
result = 2 ** 3
print(result)
```

```
8
```

In [28]:
```
result = 5 ^ 3
print(result)
```

```
6
```

3)

In [29]:
```
# AND

print('1. ',True and False)
print('2. ',True and True)
print('3. ',False and False)
```

```
1.  False
2.  True
3.  False
```

In [30]:
```
# OR

print('1. ', True or False)
print('2. ', True or True)
print('3. ', False or False)
```

```
1.  True
2.  True
3.  False
```

In [31]:
```
# Not

print('1. ', not True)
print('2. ', not False)
```

```
1.  False
2.  True
```

In [32]:
```
# XOR

print('1. ',True ^ True)
print('2. ',True ^ False)
print('3. ',False ^ False)
```

```
1.  False
2.  True
3.  False
```

4)

In [ ]:
```
'''
Right Shift Operator (>>):
The right shift operator shifts the bits of an integer to the right by a specified
number of positions. It fills the vacated bits on the left with the sign bit
(0 for positive numbers and 1 for negative numbers).

Left Shift Operator (<<):
The left shift operator shifts the bits of an integer to the left by a specified
number of positions. It fills the vacated bits on the right with 0.

'''
```

In [33]:
```
num = 10
shifted_num = num >> 2 # Right Shift
print(shifted_num)
```

```
2
```

In [35]:
```
num = 10
shifted_num = num << 2 # Left Shift
print(shifted_num)
```

```
40
```

5)

In [45]:
```
list1 = [1,2,3,4,5,6,7,8,9,10,1,12,13,14,15]
print('Length of the list:',len(list1))

if 10 in list1:
    print('10 is present')
else:
    print('10 is not present')
```

```
Length of the list: 15
10 is present
```