Assignment No. 7

Submitted By: Sachin Dodake

Q.1. Create two int type variables, apply `addition, subtraction, division and multiplications` and store the results in variables. Then print the data in the following format by calling the variables:

```
First variable is __ & second variable is __.
Addition: __ + __ = __
Subtraction: __ - __ = __
Multiplication: __ * __ = __
Division: __ / __ = __
```

Ans-1

- The program for mentioned arithmetic operation is as below:

In [14]:
```
# defining int values to variables
num_1 = 5555
num_2 = 3333

# performating adding
add = num_1 + num_2
# performating subtraction
sub = num_1 - num_2
# performating multiplicatton
mul = num_1 * num_2
# performating division
div = num_1 / num_2

print(f"First variable is `{num_1}` & second variable is `{num_2}`.")
print(f"\tAddition: {num_1} + {num_2} = {add}")
print(f"\tSubtraction: {num_1} - {num_2}= {sub}")
print(f"\tMultiplication: {num_1} * {num_2} = {mul}")
print(f"\tDivision: {num_1} / {num_2}= {div}")
```

```
First variable is `5555` & second variable is `3333`.
        Addition: 5555 + 3333 = 8888
        Subtraction: 5555 - 3333= 2222
        Multiplication: 5555 * 3333 = 18514815
        Division: 5555 / 3333= 1.6666666666666667
```

Q.2. What is the difference between the following `operators`:

```
(i) '/' & '//'
(ii) '**' & '^'
```

Ans-2

`i) '/' & '//' :`

- The difference between `Regular Division '/'` and `Floor Division '//'` is that the regular division returns the exact result in floating-point value. While, the floor division returns the nearest whole integer point value.
- In Python, the `single backslash (/)` is the division operator that performs division between two numbers and returns the result mostly in float.
- In Python, the `double backslash operator (//)` is a unique operator that performs the floor division. It divides two numbers and rounds the result down to its nearest integer.
- The double backslash operator in Python performs floor division between two operands. It rounds down the float value to its nearest whole integer-point value. The single backslash(/) operator or regular division operator performs normal division and returns the result mainly in a floating point.
- The double backslash(//) operator or floor division operator also performs normal division but returns the nearest integer value to the result. Python has a built-in function in the math module called the math.floor() that returns the floor value of any floating number.
- **`Example-1:`** The example which shows the difference between `Regular Division '/'` and `Floor Division '//'` is as below:

In [46]:
```
# defining  variables

number_1 = 14
number_2 = 5
number_3 = 15.3

print(f"Regular division of `{number_1}`  & `{number_2}` is = {number_1/number_2}       ====> It is float va

print(f"\nFloor division of `{number_1}`  & `{number_2}` is =  {number_1//number_2}      ====> It is int  v

print(f"\nFloor division of `{number_3}`  & `{number_2}` is =  {number_3//number_2}      ====> It is float
```

```
Regular division of `14`  & `5` is = 2.8       ====> It is float value.

Floor division of `14`  & `5` is =  2      ====> It is int  value.

Floor division of `15.3`  & `5` is =  3.0      ====> It is float  value.
```

`ii) " & '^' :

- **`Power/Exponential Opearator ():`**

```
    a. The operator that can be used to perform the exponent arithmetic in Python is know
as power/exponential operator `**`.
    b. In Python, we have an exponentiation operator, which is one of the ways to
calculate the exponential value of the given base and exponent values.
    c. We use the ( ** ) double asterisk/exponentiation operator between the base and
exponent values.
```

In [27]:
```
# initializing the values of base and exponent
base = 2
exponent = 16

# Use of exponentiation operator
print("Exponential value is : ", base ** exponent)
```

```
Exponential value is :  65536
```

- **`XOR Opearator (^):`**

```
    a. The operator that can be used to perform the exponent arithmetic in Python is know
as power/exponential operator `**`.
    b. In Python, XOR is a bitwise operator that is also known as Exclusive OR.
    c. It is a logical operator which outputs `1` when either of the operands is `1` (one
is `1` and the other one is `0`), but both are not `1`, and both are not `0`.
```

In [31]:
```
# defining two integers a and b
a = 15
b = 32

# Finding XOR of a and b using ^ operator
c = a ^ b

# Printing the XOR value
print(f"XOR Opearation of `{a}` and `{b}` is :  `{c}`")
```

```
XOR Opearation of `15` and `32` is :  `47`
```

Q.3. List the `Logical Operators`.

Ans-3

- Logical Operators are used to perform certain logical operations on values and variables. These are the special reserved keywords that carry out some logical computations. The value the operator operates on is known as Operand. In Python, they are used on conditional statements (either True or False), and as a result, they return boolean only (True or False). They are used to combine conditional statements
- There are following logical operators supported by Python language:

```
i) Logical AND
ii) Logical OR
iii) Logical NOT
```

`i) Logical AND`

- Logical operator AND returns True only if both the operands are True else it returns False. It is a binary operator, which means to return some value, it has to be operated between two operators (i.e, two operators are required)

In [35]:
```
a = 12
b = 26
c = 4

if a > b and a > c:
    print("Number `a` is larger")

if b > a and b > c:
    print("Number `b` is larger")

if c > a and c > b:
    print("Number `c` is larger")
```

```
Number `b` is larger
```

`ii) Logical OR`

- The logical operator OR returns False only if both the operands are False else it returns True. It is a binary operator, which means to return some value, it has to be operated between two operators (i.e, two operators are required)

In [37]:
```
a = 10
b = -5

if a < 0 or b < 0:
    print("Their product will be `Negative`.")
else:
    print("Their product will be `Positive`.")
```

```
Their product will be `Negative`.
```

`iii) Logical NOT`

- Logical NOT operator works with the single boolean value and returns the value as True if the boolean value is False and vice-versa (that is the opposite of it). It is a unary operator, which means to return some value, it has to be operated on one operator only. (i.e, only operator is required)

In [38]:
```
a = 10

if not a == 10:
    print(" a` not equals `10`")
else:
    print("`a` equals `10`")
```

```
`a` equals `10`
```

Q.4. Explain `right shift operator` and `left shift operator` with examples.

Ans-4

`i) Left Shift(<<) :`

- Bitwise Left shift operator is used to shift the binary sequence to the left side by specified position.
- The bitwise left shift operator (<<) moves the bits of its first operand to the left by the number of places specified in its second operand. It also takes care of inserting enough zero bits to fill the gap that arises on the right edge of the new bit pattern.
- Shifting a single bit to the left by one place doubles its value. For example, instead of a two, the bit will indicate a four after the shift. Moving it two places to the left will quadruple the resulting value. When you add up all the bits in a given number, you'll notice that it also gets doubled with every place shifted.
- **`Example-1`** The example which explain the `Right-Shift` operation is as below,

In [47]:
```
num1 = 14
num2 = 2
shift_left = num1 << num2
print("Operand 1 is:", num1)
print("Operand 2 is:", num2)
print(f"Result of the let shift operation on `{num1}` by `{num2}` bits is : {shift_left}.")
```

```
Operand 1 is: 14
operand 2 is: 2
Result of the let shift operation on `14` by `2` bits is : 56.
```

`ii) Right Shift(>>) :`

- Bitwise Right shift operator >> is used to shift the binary sequence to right side by specified position.
- The bitwise right shift operator (>>) is analogous to the left one, but instead of moving bits to the left, it pushes them to the right by the specified number of places. The rightmost bits always get dropped.
- Every time you shift a bit to the right by one position, you halve its underlying value. Moving the same bit by two places to the right produces a quarter of the original value, and so on.When you add up all the individual bits, you'll see that the same rule applies to the number they represent:
- **`Example-2`** The example which explain the `Left-Shift` operation is as below,

In [45]:
```
num1 = 14
num2 = 2
shift_right = num1 >> num2
print("Operand 1 is:", num1)
print("operand 2 is:", num2)
print(f"Result of the right shift operation on `{num1}` by `{num2}` bits is : {shift_right}.")
```

```
Operand 1 is: 14
operand 2 is: 2
Result of the right shift operation on `14` by `2` bits is : 3.
```

Q.5. Create a list containing int type data of length 15. Then write a code to check if 10 is present in the list or not

Ans-5

In [60]:
```
# creating list with 15 elements
num_list = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15 ]


# checking lenght of list
print(f"The length of given list is :  {len(num_list)}")

# taking element from user element
check_num = int(input("\nEnter the number: "))

if check_num in num_list:
    print(f"\tThe number {check_num} is AVAILABLE in given list.")
else:
    print(f"\tThe entered number `{check_num}` is NOT AVAILABLE in given list.")
```

```
The length of given list is :  15

Enter the number: 8
        The number 8 is AVAILABLE in given list.
```

In [ ]: