**Experiment Number – 2**

**Title - Program to swap values of two variables using bitwise operator**

Operators are special symbols that are useful for doing computations like addition, subtraction, multiplication, division, and exponentiation etc. The operators are always applied to some values which are called operands.

Python has so many built-in operators to perform different arithmetic and logical operations. There are mainly 7 types of operators in Python.

1. Arithmetic Operators
2. Relational Operators
3. Logical Operators
4. Bitwise Operators
5. Assignment operators
6. Identity operators
7. Membership operators

**Arithmetic Operators**

Arithmetic operations between two operands are carried out using arithmetic operators. It includes the exponent (\*\*) operator as well as the + (addition), - (subtraction), \* (multiplication), / (divide), % (reminder), and // (floor division) operators.

Consider the following table for a detailed explanation of arithmetic operators.

|  |  |
| --- | --- |
| **Operator** | **Description** |
| **+ (Addition)** | It is used to add two operands. For example, if a = 10, b = 10 => a+b = 20 |
| **- (Subtraction)** | It is used to subtract the second operand from the first operand. If the first operand is less than the second operand, the value results negative. For example, if a = 20, b = 5 => a - b = 15 |
| **/ (divide)** | It returns the quotient after dividing the first operand by the second operand. For example, if a = 20, b = 10 => a/b = 2.0 |
| **\* (Multiplication)** | It is used to multiply one operand with the other. For example, if a = 20, b = 4 => a \* b = 80 |
| **% (reminder)** | It returns the reminder after dividing the first operand by the second operand. For example, if a = 20, b = 10 => a%b = 0 |
| **\*\* (Exponent)** | As it calculates the first operand's power to the second operand, it is an exponent operator. |
| **// (Floor division)** | It provides the quotient's floor value, which is obtained by dividing the two operands. For example, if a = 25, b = 10 => a//b = 2 |

**Relational Operators**

Relational operators compare two values/variables and return a boolean result: True or False. For example,

|  |  |  |
| --- | --- | --- |
| **Operator** | **Meaning** | **Example** |
| == | Is Equal To | 3 == 5 gives us **False** |
| != | Not Equal To | 3 != 5 gives us **True** |
| > | Greater Than | 3 > 5 gives us **False** |
| < | Less Than | 3 < 5 gives us **True** |
| >= | Greater Than or Equal To | 3 >= 5 give us **False** |
| <= | Less Than or Equal To | 3 <= 5 gives us **True** |

**Logical Operators**

Logical operators are used to check whether an expression is True or False. They are used in decision-making. For example,

|  |  |
| --- | --- |
| **Operator** | **Description** |
| and | The condition will also be true if the expression is true. If the two expressions a and b are the same, then a and b must both be true. |
| or | The condition will be true if one of the phrases is true. If a and b are the two expressions, then an or b must be true if and is true and b is false. |
| not | If an expression **a** is true, then not (a) will be false and vice versa. |

**Bitwise operators**

Bitwise operators act on operands as if they were strings of binary digits. They operate bit by bit, hence the name.

**In the table below:** Let x = 10 (0000 1010 in binary) and y = 4 (0000 0100 in binary)

|  |  |  |
| --- | --- | --- |
| **Operator** | **Meaning** | **Example** |
| & | Bitwise AND | x & y = 0 (0000 0000) |
| | | Bitwise OR | x | y = 14 (0000 1110) |
| ~ | Bitwise NOT | ~x = -11 (1111 0101) |
| ^ | Bitwise XOR | x ^ y = 14 (0000 1110) |
| >> | Bitwise right shift | x >> 2 = 2 (0000 0010) |
| << | Bitwise left shift | x << 2 = 40 (0010 1000) |

**Assignment Operators**

Assignment operators are used to assign values to variables.

|  |  |  |
| --- | --- | --- |
| **Operator** | **Name** | **Example** |
| = | Assignment Operator | a = 7 |
| += | Addition Assignment | a += 1 # a = a + 1 |
| -= | Subtraction Assignment | a -= 3 # a = a - 3 |
| \*= | Multiplication Assignment | a \*= 4 # a = a \* 4 |
| /= | Division Assignment | a /= 3 # a = a / 3 |
| %= | Remainder Assignment | a %= 10 # a = a % 10 |
| \*\*= | Exponent Assignment | a \*\*= 10 # a = a \*\* 10 |

### Identity operators

In Python, is and is not are used to check if two values are located on the same part of the memory. Two variables that are equal does not imply that they are identical.

|  |  |  |
| --- | --- | --- |
| **Operator** | **Meaning** | **Example** |
| is | True if the operands are identical (refer to the same object) | x is True |
| is not | True if the operands are not identical (do not refer to the same object) | x is not True |

x1 = 5

y1 = 5

x2 = 'Hello'

y2 = 'Hello'

x3 = [1,2,3]

y3 = [1,2,3]

print(x1 is not y1) # prints False

print(x2 is y2) # prints True

print(x3 is y3) # prints False

Here, we see that x1 and y1 are integers of the same values, so they are equal as well as identical. Same is the case with x2 and y2 (strings). But x3 and y3 are lists. They are equal but not identical. It is because the interpreter locates them separately in memory although they are equal.

### Membership operators

In Python, in and not in are the membership operators. They are used to test whether a value or variable is found in a sequence ([string](https://www.programiz.com/python-programming/string), [list](https://www.programiz.com/python-programming/list), [tuple](https://www.programiz.com/python-programming/tuple), [set](https://www.programiz.com/python-programming/set) and [dictionary](https://www.programiz.com/python-programming/dictionary)).

In a dictionary we can only test for presence of key, not the value.

|  |  |  |
| --- | --- | --- |
| **Operator** | **Meaning** | **Example** |
| in | True if value/variable is **found** in the sequence | 5 in x |
| not in | True if value/variable is **not found** in the sequence | 5 not in x |

x = 'Hello world'

y = {1:'a', 2:'b'}

# check if 'H' is present in x string

print('H' in x) # prints True

# check if 'hello' is present in x string

print('hello' not in x) # prints True

# check if '1' key is present in y

print(1 in y) # prints True

# check if 'a' key is present in y

print('a' in y) # prints False

#### Expressions:

An expression is a combination of values, variables, and operators. A value all by itself is considered an expression, and so is a variable, so the following are all legal expressions:

#### Order of operations:

If more than one operator appears in an expression, the order of evaluation depends on the rules of precedence. For mathematical operators, Python follows mathematical convention. The acronym **PEMDAS** is a useful way to remember the rules:

1. Parentheses have the highest precedence. It can be used to force an expression to evaluate in the order you want. Since expressions in parentheses are evaluated first, 2 \* (3-1) is 4, and (1+1)\*\*(5-2) is 8. You can also use parentheses to make an expression easier to read, as in (minute \* 100) / 60, even if it doesn’t change the result.
2. Exponentiation has the next highest precedence, so 2\*\*1+1 is 3, not 4, and 3\*1\*\*3 is 3, not 27.
3. Multiplication and Division have the same precedence, which is higher than Addition and Subtraction, which also have the same precedence. So 2\*3-1 is 5, not 4, and 6+4//2 is 8, not 5.
4. Operators with the same precedence are evaluated from left to right. So the expression 5-3-1 is 1, not 3, because the 5-3 happens first and then 1 is subtracted from 2.