**4. Operators**

* 1. Arithmetic Operators Page 1 x = 10
  2. Comparison (Relational) Operators Page 2
  3. Assignment Operators Page 3
  4. Logical Operators Page 4
  5. Bitwise Operators Page 4
  6. Membership Operators Page 6
  7. Identity Operators Page 6

Operator Precedence Page 7

**4.1 Arithmetic Operators :**

|  |  |  |
| --- | --- | --- |
| **Operator** | **Description** | **Example** |
| **+ Addition** | Adds values on either side of the operator. | x = 10  y = 20  result = x + y |
| **-** Subtraction | Subtracts right hand operand from left hand operand. | result = x - y |
| **\***Multiplication | Multiplies values on either side of the operator | result = x \* y |
| **/**Division | Divides left hand operand by right hand operand | result = x / y |
| **%**Modulus | Divides left hand operand by right hand operand and returns remainder | result = x % y |
| **\*\***Exponent | Performs exponential (power) calculation on operators | a\*\*b =10 to the power 20 |
| **//**  Floor Division | - The division of operands where the result is the quotient in which the digits after the decimal point are removed. But if one of the operands is negative, the result is floored, i.e., rounded away from zero (towards negative infinity) − | 9//2 = 4 and 9.0//2.0 = 4.0,  -11//3 = -4, -11.0//3 = -4.0 |

**4.2 Relational(Comparison) Operators:**

These operators compare the values on either sides of them and decide the relation among them. They are also called Relational operators.

Assume variable **a** holds **10** and variable **b** holds **20**, then –

|  |  |  |
| --- | --- | --- |
| **Operator** | **Description** | **Example** |
| **==** | If the values of two operands are equal, then the condition becomes true.  It will compare **contents** of two values | (a == b) is not true. |
| **!=** | If values of two operands are not equal, then condition becomes true. | (a != b) is true. |
| <> | If values of two operands are not equal, then condition becomes true. | (a <> b) is true. This is similar to != operator. |
| **>** | If the value of left operand is greater than the value of right operand, then condition becomes true. | (a > b) is not true. |
| **<** | If the value of left operand is less than the value of right operand, then condition becomes true. | (a < b) is true. |
| **>=** | If the value of left operand is **greater than OR equal** to the value of right operand, then condition becomes true. | (a >= b) is not true.  20**>=** 20  **OR** |
| **<=** | If the value of left operand is **less than OR equal to** the value of right operand, then condition becomes true. | (a <= b) is true. |

**4.3 Assignment Operators :**

Assume variable **a** holds **10** and variable **b** holds **20**, then –

|  |  |  |
| --- | --- | --- |
| **Operator** | **Description** | **Example** |
| **=** | Assigns values from right side operands to left side operand  **c = 20**  12345  **c = 30**    45678  1000001 | Age = 20  LHS = RHS  a=10  b=20  c = a+b  print(c)  **a=10**  **c=20**  **c = c+a**  **print(c)**  **c = 40** |
| **+=** | It adds right operand to the left operand and assign the result to left operand | c += a is equivalent to **c = c + a** |
| **-=**  Subtract AND | It subtracts right operand from the left operand and assign the result to left operand | c -= a is equivalent to c = c – a |
| **\*=**  Multiply AND | It multiplies right operand with the left operand and assign the result to left operand | c \*= a is equivalent to c = c \* a |
| **/=**  Divide AND | It divides left operand with the right operand and assign the result to left operand | c /= a is equivalent to c = c / a  c /= a is equivalent to c = c / a |
| **%=**  Modulus AND | It takes modulus using two operands and assign the result to left operand | c %= a is equivalent to c = c % a |
| **\*\*=**  Exponent AND | Performs exponential (power) calculation on operators and assign value to the left operand | c \*\*= a is equivalent to c = c \*\* a |
| **//=**Floor Division | It performs floor division on operators and assign value to the left operand | c //= a is equivalent to c = c // a |

**4.4Logical Operators :**

Assume variable a holds 10 and variable b holds 20 then –

|  |  |  |
| --- | --- | --- |
| **Operator** | **Description** | **Example** |
| **And**  Logical AND | If both the operands are true then condition becomes true. **Deciding based on**  True **AND** True - True 2nd cond.  True **AND** False - False 2nd cond.  False **AND** True – False 1st cond.  False **AND** False – False 1st cond. | (a and b) is true. |
| **Or**  Logical OR | If any of the two operands are non-zero then condition becomes true. **Deciding based** on  **True** **OR** True - True 🡪 1st cond.  **True** **OR** False - True 🡪 1st cond.  False **OR True** - True 🡪 2nd cond.  False  **OR False** – False 🡪 2nd cond. | (a or b) is true. |
| **Not**  Logical NOT | Used to reverse the logical state of its operand. | not(a and b) is False.  not(False) => True  not(True) => False |

**Logical Gates:  
==================  
1 represents True  
0 represents False**

**AND gate :  
-------------  
1 and 1 => 1 True and True\* => True  
1 and 0 => 0 True and False\* => False  
0 and 1 => 0 False\* and True => False  
0 and 0 => 0 False\* and False => False  
  
OR gate :  
-----------  
1 or 1 => 1 True\* or True => True  
1 or 0 => 1 True\* or False => True  
0 or 1 => 1 False or True\* => True  
0 or 0 => 0 False or False\* => False  
  
NOT gate:  
------------  
not 1 => 0 not True => False  
not 0 => 1 not False => True**

>>> x = 10

>>> y = 20

>>>

>>> x > y and x == y

**False and False 🡺 False**

## 4.5 Bitwise Operators :

Bitwise operator works on bits and performs bit by bit operation. Assume if

a = 60; and b = 13; Now in binary format they will be as follows –

1 bit

**8 bits – 1 byte**

1024 bytes – 1 kb

1024 kbytes – 1mb

1024 mb – 1GB

1024 GB – 1TB

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Binary 0 | 0 | 1 | 1 | 1 | 1 | 0 | 0 |
| pos7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

**2\*\*7 2\*\*6 2\*\*5 2\*\*4 2 \*\*3 2 \*\*2 2 \*\*1 2 \*\* 0**

**32 16 8 4**

**1234567**

**1 byte**

**2 bytes**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |  |

**Unicode format/ 2 bytes** 0 - 255

a = 60, b = 13

a = 0 0 1 1 1 1 0 0 60

b = 0 0 0 0 **1 1 0 1 13**

--------------------

a&b = 0000 1100 10&20

a|b = 0011 1101

~a  = 1100 0011

a^b = 0011 0001

There are following Bitwise operators supported by Python language

|  |  |  |
| --- | --- | --- |
| **Operator** | **Description** | **Example** |
| **&**  **Binary AND** | **Operator copies a bit to the result if it exists in both operands** | **(a & b)**  **0000 1100** |
| **|**  **Binary OR** | **It copies a bit if it exists in either operand.** | **(a | b) = 61**  **0011 1101** |
| **^**  Binary XOR | It copies the bit if it is set in one operand but not both. | (a ^ b) = 49  0011 0001 |
| **~**  Binary Ones Complement | It is unary and has the effect of 'flipping' bits. | (~a ) = -61  1100 0011  in 2's complement form due to a signed binary number. |
| **<<**Binary Left Shift | The left operands value is moved left by the number of bits specified by the right operand. | a << 2 = 240  1111 0000 |
| **>>**  Binary Right Shift | The left operands value is moved right by the number of bits specified by the right operand. | a >> 2 = 15  0000 1111 |

## 4.6 Membership Operators :

Python’s membership operators test for membership in a sequence, such as strings, lists, or tuples. Set1 = {1, 2, 3, 4}

There are two membership operators

1 **in** {1, 2, 3, 4} True

1. **in** {1,2,3,4} False
2. **not in** {1,2,3,4} True

1 not in {1,2,3,4} False

|  |  |  |
| --- | --- | --- |
| **Operator** | **Description** | **Example** |
| **in** | Evaluates to true if it finds a variable in the specified sequence and false otherwise. | x in y,  here **in** results in a 1 if x is a member of sequence y. |
| **not in** | Evaluates to true if it does not finds a variable in the specified sequence and false otherwise. | x not in y,  here **not in** results in a 1 if x is not a member of sequence y. |

## 4.7 Identity Operators :

Identity operators compare the **memory locations** of two objects.

There are two Identity operators explained below –

>>> x = 10

>>> y = 10

>>> x == y

True

>>> x is y

True

>>>

>>> x != y

False

>>> x is not y

False

>>>

|  |  |  |
| --- | --- | --- |
| **Operator** | **Description** | **Example** |
| **Is** | Evaluates to true if the variables on either side of the operator point to the same object and false otherwise. | x is y True x == y,  here **is** results in 1 if id(x) equals id(y). |
| **is not** | Evaluates to false if the variables on either side of the operator point to the same object and true otherwise. | x is not y,  here **is not** results in 1 if id(x) is not equal to id(y). |

## 

## 

## Operators Precedence :

## The following table lists all operators from highest precedence to lowest.

## (1+4)%(2+3)-1+5%3 BODMAS principle

## 5%5-1+5%3

## 0-1+2

## 0+1

## 1

|  |  |
| --- | --- |
| **Sr.No.** | **Operator & Description** |
| 1 | **\*\***Exponentiation (raise to the power) |
| 2 | **~ + -**Complement, unary plus and minus  (method names for the last two are +@ and -@) |
| 3 | **\* / % //** Multiply, divide, modulo and floor division |
| 4 | **+ -**Addition and subtraction |
| 5 | **>><<**Right and left bitwise shift |
| 6 | **&**Bitwise 'AND' |
| 7 | **^ |**Bitwise exclusive `OR' and regular `OR' |
| 8 | **<= <>>=**Comparison operators |
| 9 | **<> == !=**Equality operators |
| 10 | **= A**ssignment operators |
| 11 | **is is not I**dentity operators **AIML** |
| 12 | **in not in M**embership operators |
| 13 | **not or and L**ogical operators |

## BODMAS

## Brackets of Division Multiplication

## Addition Subtraction

## 5+(10-20)\*30/5