**Python Operators**

Operators are used to perform operations on variables and values.

There are seven groups that operators are coming under:

1. **Arithmetic operators** :used with numeric values to perform common mathematical operations.

|  |  |  |
| --- | --- | --- |
| **Operators** | **Name** | **Example** |
| + | Addition | x,y=5,2 print(x+y)*#7* |
| - | Subtraction | x,y=5,2 print(x-y)*#3* |
| \* | Multiplication | x,y=5,2  print(x\*y)*#10* |
| / | Division | x,y=5,2  print(x/y)*#2.5* |
| % | Modulus | x,y=5,2  print(x%y)*#1* |
| \*\* | Exponentiation | x,y=5,2  print(x\*\*y)*#25* |
| // | Floor Division returns the largest possible integer | x,y=5,2  print(x//y)*#2* |

1. **Assignment Operators**: used to assign values to variables

|  |  |  |
| --- | --- | --- |
| **Operators** | **Name** | **Example** |
| = | Assign right value to the left value | x=5 print(x)*#5* |
| += | Add and Assign: add right value with left value and assign the result to left value | x,y=5,1 y+=x *# y=y+x* print(y)*#6* |
| -= | Subtract and Assign: subtract right value from left value and assign the result to left value | x,y=5,1 y-=x *# y=y-x* print(y)*#-4* |
| \*= | Multiply and Assign: multiply right value with left value and assign result to left value | x,y=5,1 y\*=x *# y=y\*x* print(y)*#5* |
| /= | Divide and Assign: divide left value by right value and assign result to left value | x,y=5,1 y/=x *# y=y/x* print(y)*#0.2* |
| %= | Modulus and Assign: take the modulus using left and right values and assign result to left value | x,y=5,1 y%=x *# y=y%x* print(y)*#1* |
| //= | Floor Division and Assign: divide left value by right value and assign result to left value | x,y=5,1 y//=x *# y=y//x* print(y)*#0* |
| \*\*= | Exponent and Assign: calculate exponent value using left and right values and assign result to left value | x,y=5,1 y\*\*=x *# y=y\*\*x* print(y)*#1* |
| &= | Bitwise AND and Assign: perform bitwise AND on both values and assign result to left value | x,y=5,1 y&=x *# y=y&x* print(y)*#1* |
| |= | Bitwise OR and Assign: perform bitwise OR on both values and assign result to left value | x,y=5,1 y|=x *# y=y|x* print(y)*#5* |
| ^= | Bitwise XOR and Assign: perform bitwise XOR on both values and assign result to left value | x,y=5,1 y^=x *# y=y^x* print(y)*#4* |
| >>= | Bitwise Right Shift and Assign: perform bitwise Right Shift on values and assign result to left value | x=5 x>>=1 *# x=x>>1* print(x)*#2* |
| <<= | Bitwise Left Shift and Assign: perform bitwise Left Shift on values and assign result to left value | x=5 x<<=1 *# x=x<<1* print(x)*#10* |

1. **Comparison operators:** used to compare two values

|  |  |  |
| --- | --- | --- |
| **Operators** | **Name** | **Example** |
| == | Equal | x,y=5,2 print(x==y)*#False* |
| != | Not Equal | x,y=5,2 print(x!=y)*#True* |
| > | Greater Than | x,y=5,2 print(x>y)*#True* |
| > | Less Than | x,y=5,2 print(x<y)*#False* |
| >= | Greater than or equal to | x,y=5,2 print(x>=y)*#True* |
| <= | Less than or equal to | x,y=5,2 print(x<=y)*#False* |

1. Logical operators: used to combine conditional statements

|  |  |  |
| --- | --- | --- |
| **Operators** | **Name** | **Example** |
| and | Returns True if both statements are true | x,y=5,6 print(x<y and x==5)*#True* |
| or | Returns True if one of the statements is true | x,y=5,6 print(x>y and x==5)*#False* |
| not | Reverse the result, returns False if the result is true | x,y=5,6 print(not(x>y))*#True* |

1. Identity Operators : used to compare the objects, not if they are equal, but if they are actually the same object, with the same memory location:

|  |  |  |
| --- | --- | --- |
| **Operators** | **Name** | **Example** |
| is | Returns True if both variables are the same object | x,y=["apple","orange","grapes"],  ["dogs","cats","cows"] z=x print(x is z)*#True* print(y is z)*#False* |
| Is not | Returns True if both variables are not the same object | x,y=["apple","orange","grapes"],  ["dogs","cats","cows"] print (x is not y)*#True* |

1. Membership Operators: used to test if a sequence is presented in an object

|  |  |  |
| --- | --- | --- |
| **Operators** | **Name** | **Example** |
| in | Returns True if a sequence with the specified value is present in the object | x=["apple","orange","grapes"] print("apple" in x)*#True* |
| not in | Returns True if a sequence with the specified value is not present in the object | x=["apple","orange","grapes"] print ("cherry" is not x)*#True* |

1. Bitwise operators: used to perform bitwise calculations on integers. The integers are first converted into binary and then operations are performed on bit by bit, hence the name bitwise operators. Then the result is returned in decimal format.

**BITWISE AND (&) – if both bits are 1 return 1 else 0**

a = 10 = 1010 (Binary)

b = 4 = 0100 (Binary)

a & b = 1010

0100

----

= 0000

= 0 (Decimal)

**BITWISE OR (|) – if both bits are 0 return 0 else 1**

a = 10 = 1010 (Binary)

b = 4 = 0100 (Binary)

a | b = 1010

0100

----

= 1110

= 14 (Decimal)

**BITWISE XOR (^) – Returns 1 if one of the bits is 1 and the other is 0 else returns false**

a = 10 = 1010 (Binary)

b = 4 = 0100 (Binary)

a ^ b = 1010

0100

----

= 1110

= 14 (Decimal)

**BITWISE COMPLEMENT/NOT (~) – returns one's complement of the number. If it 0 , flip as 1 and if it 1, flip as 0. [~x=-x-1]**

a = 10 = 1010 (Binary)

b = 4 = 0100 (Binary)

~a = ~1010+1 = -1011 = -11(Decimal)

~b = ~0100+1 = -0101 = -5(Decimal)

**BITWISE RIGHT SHIFT (>>) – Shifts the bits of the number to the right and fills 0 on voids left. If calculate for negative value, then you need to add 1 for the left side otherwise 0.**

**[x>>n] x=value & n=number of bits to shift**

a = 10 = 00001010 (Binary)

b = 4 = 00000100 (Binary)

a << 2 = 00001010 ->00000010= 2(Decimal)

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 0 | 0 | 0 | 1 | 0 | 1 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 |

b << 2 = 00000100 ->00010000 = 1 (Decimal)

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |

**BITWISE LEFT SHIFT (<<) – Shifts the bits towards left and add 0 bits at the right. For both negative and positive values, then you need to add 1 for the left side otherwise 0.**

**[x<<n] x=value & n=number of bits to shift**

a = 10 = 00001010 (Binary)

b = 4 = 00000100 (Binary)

a << 2 = 00001010 ->00101000 = **25 +23**= 40(Decimal)

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 0 | 0 | 0 | 1 | 0 | 1 | 0 |
| 0 | 0 | 1 | 0 | 1 | 0 | 0 | 0 |
| 27 | 26 | **25** | 24 | **23** | 22 | 21 | 20 |

b << 2 = 00000100 ->00010000 = **24** = 16(Decimal)

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 |
| 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 |
| 27 | 26 | 25 | **24** | 23 | 22 | 21 | 20 |