Python - Operadores bit a bit

Operadores bit a bit Python

Os operadores bit a bit do Python são normalmente usados para realizar operações bit a bit em objetos do tipo inteiro. Porém, em vez de tratar o objeto como um todo, ele é tratado como uma sequência de bits. Diferentes operações são realizadas em cada bit da string .

Python tem seis operadores bit a bit - &, |, ^, ~, << e >>. Todos esses operadores (exceto ~) são de natureza binária, no sentido de que operam em dois operandos. Cada operando é um dígito binário (bit) 1 ou 0.

A seguir estão os operadores bit a bit em Python -

- Operador AND bit a bit
- Operador OR bit a bit
- Operador XOR bit a bit
- Operador NOT bit a bit
- Operador de deslocamento esquerdo bit a bit
- Operador de deslocamento à direita biwtise

Operador AND bit a bit do Python (&)

O operador AND bit a bit é um pouco semelhante ao operador lógico e. Ele retorna True somente se ambos os operandos de bit forem 1 (ou seja, True). Todas as combinações são -

```
0 e 0 é 0
1 e 0 é 0
0 e 1 é 0
1 e 1 é 1
```

Quando você usa números inteiros como operandos, ambos são convertidos em binário equivalente, a operação & é feita no bit correspondente de cada número, começando pelo bit menos significativo e indo em direção ao bit mais significativo.

Exemplo de operador AND bit a bit em Python

Vamos pegar dois inteiros 60 e 13 e atribuí-los às variáveis a e b, respectivamente.

```
a=60
b=13
print ("a:",a, "b:",b, "a&b:",a&b)
```

Ele produzirá a seguinte **saída** -

```
a: 60 b: 13 a&b: 12
```

Para entender como o Python executa a operação, obtenha o equivalente binário de cada variável.

```
print ("a:", bin(a))
print ("b:", bin(b))
```

Ele produzirá a seguinte **saída** -

```
a: 0b111100
b: 0b1101
```

Por uma questão de conveniência, use o formato padrão de 8 bits para cada número, de modo que "a" seja 00111100 e "b" seja 00001101. Vamos realizar manualmente uma operação em cada bit correspondente desses dois números.

```
0011 1100
&
0000 1101
-------------
0000 1100
```

Converta o binário resultante de volta para inteiro. Você obterá 12, que foi o resultado obtido anteriormente.

```
>>> int('00001100',2)
12
```

Operador Python bit a bit OR (|)

O "|" símbolo (chamado **pipe**) é o operador OR bit a bit. Se qualquer operando de bit for 1, o resultado será 1, caso contrário será 0.

```
0 | 0 é 0
0 | 1 é 1
1 | 0 é 1
1 | 1 é 1
```

Exemplo de operador OR bit a bit em Python

Tome os mesmos valores de a=60, b=13. O "|" a operação resulta em 61. Obtenha seus equivalentes binários.

```
a=60
b=13
print ("a:",a, "b:",b, "a|b:",a|b)
print ("a:", bin(a))
print ("b:", bin(b))
```

Ele produzirá a seguinte **saída** -

```
a: 60 b: 13 a|b: 61
a: 0b111100
b: 0b1101
```

Para realizar o "|" operação manualmente, use o formato de 8 bits.

```
0011 1100
   |
0000 1101
-------------
0011 1101
```

Converta o número binário de volta em inteiro para calcular o resultado -

```
>>> int('00111101',2)
61
```

Operador Python XOR bit a bit (^)

O termo XOR significa OR exclusivo. Isso significa que o resultado da operação OR em dois bits será 1 se apenas um dos bits for 1.

```
0 ^ 0 is 0
0 ^ 1 is 1
1 ^ 0 is 1
1 ^ 1 is 0
```

Exemplo de operador XOR bit a bit em Python

Vamos realizar a operação XOR em a=60 e b=13.

```
a=60
b=13
print ("a:",a, "b:",b, "a^b:",a^b)
```

Ele produzirá a seguinte **saída** -

```
a: 60 b: 13 a^b: 49
```

Agora executamos o XOR bit a bit manualmente.

```
0011 1100
  ^
0000 1101
------------
0011 0001
```

A função int() mostra que 00110001 é 49.

```
>>> int('00110001',2)
49
```

Operador NOT bit a bit Python (~)

Este operador é o equivalente binário do operador lógico NOT. Ele inverte cada bit para que 1 seja substituído por 0 e 0 por 1 e retorne o complemento do número original. Python usa o método complemento de 2. Para números inteiros positivos, é obtido simplesmente invertendo os bits. Para um número negativo, -x, ele é escrito usando o padrão de bits para (x-1) com todos os bits complementados (alterados de 1 para 0 ou de 0 para 1). Portanto: (para representação de 8 bits)

```
-1 is complement(1 - 1) = complement(0) = "11111111"
-10 is complement(10 - 1) = complement(9) = complement("00001001") = "1111011
```

Exemplo de operador NOT bit a bit em Python

Para a=60, seu complemento é -

```
a=60
print ("a:",a, "~a:", ~a)
```

Ele produzirá a seguinte **saída** -

```
a: 60 ~a: -61
```


Operador de deslocamento esquerdo bit a bit em Python (<<)

O operador de deslocamento para a esquerda desloca os bits mais significativos para a direita pelo número no lado direito do símbolo "<<". Conseqüentemente, "x << 2" causa dois bits da representação binária para a direita.

Exemplo de operador de deslocamento esquerdo bit a bit em Python

Vamos realizar o deslocamento para a esquerda em 60.

```
a=60
print ("a:",a, "a<<2:", a<<2)
```

Ele produzirá a seguinte **saída** -

```
a: 60 a<<2: 240
```

Como isso acontece? Vamos usar o equivalente binário de 60 e realizar o deslocamento para a esquerda em 2.

```
0011 1100
<<
    2
-------------
1111 0000
```

Converta o binário em inteiro. São 240.

```
>>> int('11110000',2)
240
```


Operador Python bit a bit para a direita (>>)

O operador de deslocamento para a direita desloca os bits menos significativos para a esquerda pelo número no lado direito do símbolo ">>". Conseqüentemente, "x >> 2" causa dois bits da representação binária para a esquerda.

Exemplo de operador bit a direita Shift em Python

Vamos realizar o deslocamento para a direita em 60.

```
a=60
print ("a:",a, "a>>2:", a>>2)
```

Ele produzirá a seguinte **saída** -

```
a: 60 a>>2: 15
```

A operação manual de mudança para a direita em 60 é mostrada abaixo -

```
   0011 1100
   >>
   2
   -------------
   0000 1111
```

Use a função int() para converter o número binário acima em inteiro. São 15.

```
>>> int('00001111',2)
15
```