Assignment, Bitwise and Membership Ops

August 11, 2022

1 Operators in Python

Terminology of an Operation 1. Operand(s) 2. Operator

- Operand(s) –> On which the operation is being performed
- Operator –> The one which is performing the operation

Example: 10 + 2

Operand(s): 10, 2
Operator: +

Types of Operators in Python

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

1.1 Assignment Operators

- Used to assign values to variables
- = –> Assign
- += –> Add and assign
- -= –> Subtract and assign
- *= –> Multiply and assign
- /= –> Divide and assign quotient
- //= –> Divide and assign floored quotient
- %= –> Divide and assign remainder
- **= –> Exponent and assign
- In assignment operators the value of the left operand is always affected

[2]:
```
a = 10
a += 15
print(a)
```

```
25
```

[3]:
```
x = 15
x -= 12
print(x)
```

```
3
```

[4]:
```
y = 3
y *= 5
print(y)
```

```
15
```

[6]:
```
p = 25
p %= 3
print(p)
```

```
1
```

[7]:
```
a = 10
b = 3
a += b
print(a, b)
```

```
13 3
```

[8]:
```
a = 10
b = 3
b += a
print(a, b)
```

```
10 13
```

[9]:
```
a = 3
b = 4
c = 5
a += b # a b c = 7 4 5
b += c # a b c = 7 9 5
c += a # a b c = 7 9 12
a *= b # a b c = 63 9 12
b *= c # a b c = 63 108 12
b -= a # a b c = 63 45 12
print(a, b, c) # 63 45 12
```

```
63 45 12
```

[10]:
```
a = 3
b = 4
c = 5
print(a, b, c)
```

```
3 4 5
```

[11]:
```
# Single line assignment
a, b, c = 3, 4, 5
print(a, b, c)
```

```
3 4 5
```

[12]:
```
a = b = c = 0
print(a, b, c)
```

```
0 0 0
```

1.2 Decimal Number System (Base 10)

- 100, 5465456946566 –> Decimal Number System
- Deci –> Ten (10)
- Digits –> 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 –> 10

1.3 Binary Number System (Base 2)

- There are only two digits –> 0, 1
- 2 (and it's powers) a significant role in binary

1.3.1 Decimal to Binary Conversion

Decimal Number: 23
Binary Represe: 10111

- Find out the nearest power of 2 which is <= given number
- Put a 1 below the numbers that you can use to add upto the given number
- Put a 0 below the numbers that you can't use

16 8 4 2 1
1 0 1 1 1 –> This the binary representation of 23.
(10111)

Decimal: 49
Binary: 110001

32 16 8 4 2 1
1 1 0 0 0 1 –> This the binary of 49.
(110001)

Decimal: 75
Binary: 1001011

64 32 16 8 4 2 1
1 0 0 1 0 1 1 –> This is the binary of 75
(1001011)

Halving
Decimal: 23
Binary: ?
23 –> 1
11 –> 1
5 –> 1
2 –> 0
1 –> 1
- The number is even write a 0 - The number is odd write a 1

Write the results bottom up
(10111)

Decimal: 75
Binary: 1001011

75 –> 1
37 –> 1
18 –> 0
9 –> 1
4 –> 0
2 –> 0
1 –> 1

Bottom up (1001011)

1.3.2 Binary to Decimal Conversion

Example 1:

Binary: 10111
Decimal: 23

To convert any binary number into decimal form do the following

- Seperate the bits as follows and
- Write the powers of 2 (starting from 2^0) from right to left under each bit

1	0	1	1	1
16	8	4	2	1

And add all the numbers under 1 to get the decimal representation

$16 + 4 + 2 + 1 = 23$

Some more examples

Example 2:

Binary: 1101101
Decimal: 109

Seperating bits

1	1	0	1	1	0	1
64	32	16	8	4	2	1

Adding all the numbers that are under 1 to get the decimal representation

$64 + 32 + 8 + 4 + 1 = 109$

Example 3:

Binary: 1111011
Decimal:

Seperating bits

1	1	1	1	0	1	1
64	32	16	8	4	2	1

Adding all the numbers that are under 1 to get the decimal representation

$64 + 32 + 16 + 8 + 2 + 1 = 123$

1.4 Bitwise Operators

- Operates on bit level
- Requires knowledge on Binary Number System.
- & –> Bitwise AND
- –> Bitwise OR
- ^ –> Bitwise XOR
- « –> Left shift
- » –> Right shift

[31]:
```
from random import *
l = [randint(1, 500) for i in range(5)]
b = [bin(i)[2:] for i in l]
print(*l, sep = '\n')
```

```
413
202
39
```

```
425
117
```

[32]:
```
for i in b:
    print(i)
```

```
110011101
11001010
100111
110101001
1110101
```

[37]:
```
print(f'{b[0]} --> {l[0]}')
print(f'{b[1]} --> {l[1]}')
print(f'{b[2]} --> {l[2]}')
print(f'{b[3]} --> {l[3]}')
print(f'{b[4]} --> {l[4]}')
```

```
110011101 --> 413
11001010 --> 202
100111 --> 39
110101001 --> 425
1110101 --> 117
```

1.4.1 Bitwise AND (&)

- Compares each bit of first operand to the corresponding bit of second operand and sets the result bit to 1, if and only if both bits are 1, otherwise to 0

[42]:
```
a = 11
b = 12

# 11 -->  1011
# 12 --> 1100

# 1011
# 1100
# ----
# 1000 --> 8
print(a & b)
```

```
8
```

1.4.2 Bitwise OR (|)

- Compares each bit of first operand to the corresponding bit of second operand and sets the result bit to 1, even if one of the bits is 1, otherwise to 0

[43]:
```
a = 11
b = 12

# 1011
# 1100
# ----
# 1111 --> 15
print(a | b)
```

```
15
```

1.4.3 Bitwise XOR

- Compares each bit of first operand to the corresponding bit of
  second operand and sets the result bit to 1, if one bit is 0 and the other is 1 or vice versa (alternate bits) otherwise to 0.

[44]:
```
a = 11
b = 12
# 1011
# 1100
# ----
# 0111 --> 7
print(a ^ b)
```

```
7
```

A = 9 B = 14

A	B	A&B	A|B	A^B
1	1	1	1	0
0	1	0	1	1
0	1	0	1	1
1	0	0	1	1
		8	15	7

[49]:
```
lst = [(13, 10), (21, 24), (36, 41), (67, 73)]
for i in lst:
    print(*i)
```

```
13 10
21 24
36 41
67 73
```

[59]:
```
def get_bitwise_ops(x):
    return (x[0]&x[1], x[0]|x[1], x[0]^x[1])
def result(z):
    print('{} --> {}'.format(z, get_bitwise_ops(z)))
```

[63]:
```
result(lst[0])
result(lst[1])
result(lst[2])
result(lst[3])
```

```
(13, 10) --> (8, 15, 7)
(21, 24) --> (16, 29, 13)
(36, 41) --> (32, 45, 13)
(67, 73) --> (65, 75, 10)
```

1.4.4 Bit Shortage:

- In case of a bit shortage, we are allowed to add bits (0s) on the LEFT SIDE (Preceding bits) of the operand.

1.4.5 Left shift («)

- Left shift <–> multiplying
- a « b <–> a * 2 ** b

[64]:
```
print(3 << 2)
# 12
```

```
12
```

[65]:
```
print(7 << 1)
# 7 * 2 = 14
```

```
14
```

[66]:
```
print(11 << 4)
# 11 * 16
```

```
176
```

1.4.6 Right shift (»)

- Right shift <–> dividing
- a » b <–> a // 2 ** b

[67]:
```
print(11 >> 2)
# 11 // 2 ** 2 --> 11 // 4 --> 2
```

```
2
```

[68]:
```
print(25 >> 1)
# 25 // 2 ** 1 --> 25 // 2 --> 12
```

```
12
```

[69]:
```
print(76 >> 4)
# 76 // 2 ** 4 --> 76 // 16 --> 4
```

```
4
```

1.5 Membership Operators

- in
- not in
- Tells if the given element is a member of an iterable
- Iterables: Everything that can be looped over is an iterable in python
- Examples of iterable objects: list, string, tuple, set, range
- Result of a membership operator is always a boolean value (True or False)

[70]:
```
print('h' in 'hello')
```

```
True
```

[71]:
```
print('H' in 'hello')
```

```
False
```

[72]:
```
print('hell' in 'hello')
```

```
True
```

[73]:
```
print('helo' in 'hello')
```

```
False
```

[74]:
```
print('llo' in 'hello')
```

```
True
```

[75]:
```
print(' ' in 'hello world')
```

```
True
```

[76]:
```
my_list = [10, 20, 30]
print(10 in my_list)
```

```
True
```

[77]:
```
my_list = [10, 20, 30]
print(100 in my_list)
```

```
False
```

[78]:
```
print(1 in 100)
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Input In [78], in <cell line: 1>()
----> 1 print(1 in 100)

TypeError: argument of type 'int' is not iterable
```

[79]:
```
print('1' in '100')
```

```
True
```

[80]:
```
print(2 in 12.2)
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Input In [80], in <cell line: 1>()
----> 1 print(2 in 12.2)

TypeError: argument of type 'float' is not iterable
```

[81]:
```
print('h' not in 'hello')
```

```
False
```

[82]:
```
print(100 not in [10, 20, 30])
```

```
True
```

[85]:
```
# Vowel or Consonant
ch = input("Enter a character")
if ch == 'a' or ch == 'e' or ch == 'i' or ch == 'o' or ch == 'u':
    print('Vowel')
else:
    print('Consonant')
```

```
Enter a characterA
Consonant
```

[89]:
```
ch = input("Enter a character: ") # Z
vowels = ['a', 'e', 'i', 'o', 'u', 'A', 'E', 'I', 'O', 'U']
if ch in vowels: # if False
    print('Vowel')
else:
    print('Consonant')
```

```
Enter a character: Z
Consonant
```

[92]:
```python
# Using Membership Operator
ch = input("Enter a character: ")
vowels = 'aeiouAEIOU'
if ch in vowels:
    print('Vowel')
else:
    print('Consonant')
```

```
Enter a character: O
Vowel
```

[99]:
```python
# Program to find if the given character is
# - digit (0 - 9)
# - alphabet (a - z or A - Z)
# - special character (&*%^$...)
ch = input('Enter a character: ')
digits = '0123456789'
alpha = 'abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ'
if ch in digits:
    print('Digit')
elif ch in alpha:
    print('Alphabet')
else:
    print('Special Character')
```

```
Enter a character: A
Alphabet
```