运算符、优先级

算术运算符

运算符	描述
+	加
-	减
*	乘
/	除
%	取模（返回除法的余数）
**	幂
//	整除（ 相当于 / 再向下取整）

```
a = 3
b = 2

d = a + b
print(d)  # 5

d = a - b
print(d)  # 1

d = a * b
print(d)  # 6

d = a / b
print(d)  # 1.5

# 取模(除法的余数)
```

```
d = a % b
print(d)  # 1

# 相当于 / 再向下取整
d = a // b
print(d)  # 1

# 10**8 还可以用科学计数法写成 1e8
d = a ** b
print(d)  # 9

d = 10 ** 8
print(d)  # int

f = 1e8
print(f)  # float
```

负数取整，取模

```
print(15 // 4)  # 15 / 4等于3点几，然后再向下取整，取更小的最接近的整数3
print(-15 // 4)  # -15 / 4等于负3点几，然后再向下取整，取更小的最接近的整数-4

print(15 % 4)  # 余数 = 15 - (3*4) = 3
print(-15 % 4)  # 余数 = -15 - (-4*4) = 1
# 规律：(a % b) + (-a % b) = b或0 (当a为0时，为0)
```

比较运算符

返回布尔值：True，False

运算符	描述

运算符	描述
==	等于
!=	不等于
>	大于
<	小于
>=	大于或等于
<=	小于或等于

```
a = 20
b = 9

c = a == b
print(c)  # False

c = a != b
print(c)  # True

c = a > b
print(c)  # True

c = a < b
print(c)  # False

c = a >= b
print(c)  # True

c = a <= b
print(c)  # False
```

赋值运算符

运算符	描述

运算符	描述
=	简单的赋值运算符
+=	加法赋值运算符
-=	减法赋值运算符
*=	乘法赋值运算符
/=	除法赋值运算符
%=	取模赋值运算符
**=	幂赋值运算符
//=	取整赋值运算符
:=	海象赋值运算符（Python3.8 版本新增运算符）

```
a = 3

c = a + 2
print(c)  # 5

c += a
print(c)  # 8   结果等于 c = c + a

c -= a
print(c)  # 5   结果等于 c = c - a

c *= a
print(c)  # 15   结果等于 c = c * a

c /= a
print(c)  # 5.0   结果等于 c = c / a

c %= a
print(c)  # 2.0   结果等于 c = c % a

c **= a
print(c)  # 8.0   结果等于 c = c ** a

c //= a
```

```
print(c)  # 2.0    结果等于 c = c // a
```

:= 海象运算符

```
# 写法一：
string = "hello world"
length = len(string)
print(length + 5)
print(f"string的长度为{length}")


# 写法二：
string = "hello world"
print(len(string) + 5)
print(f"string的长度为{len(string)}")


# 写法三：
# 海象运算符，可在表达式内部为变量赋值，记得加括号，不然就是赋值len(string) + 5的返回值了
# 相对写法一：避免了一次赋值给中间变量的步骤
# 相对写法二：避免使用两次len()函数
string = "hello world"
print((length := len(string)) + 5)
print(f"string的长度为{length}")
```

增强赋值

- 增强赋值在条件符合的情况下（例如：操作数是一个可变类型对象）会以追加的方式来进行处理，而普通赋值则会以新建的方式进行处理，此时增强赋值语句比普通赋值语句的效率是更高的，但是也要注意因此产生的问题

```
""" 增强的赋值运算符 +=  -=  *=  /=  %=  **=  //= """

lis1 = [1, 2]
lis2 = lis1
```

```
lis1 = lis1 + [3, 4]
print(lis1)  # [1, 2, 3, 4]
print(lis2)  # [1, 2]


lis1 = [1, 2]
lis2 = lis1
lis1 += [3, 4]
print(lis1)  # [1, 2, 3, 4]
print(lis2)  # [1, 2, 3, 4]


# a = 12
# b = a
# a = a + 1
# print(a)  # 13
# print(b)  # 12

# a += b  # 等效于 a = a + b

# a = 12
# b = a
# a += 1  # 等效于 a = a + 1
# print(a)  # 13
# print(b)  # 12
```

序列赋值

```
a, b = 3, 4  # 等同于 (a,b) = (3,4)
print(a, b)

(a, b) = (3, 4)
print(a, b)

[a, b] = [3, 4]
print(a, b)
```

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

a, b, c = "你好吗"
print(a, b, c)
```

多目标赋值

```
a = b = c = 999
d = 999

print(id(a))
print(id(b))
print(id(c))
print(id(d))

b = d
print(id(b))



a = b = c = [1, 2, 3]
b.append(4)
print(a, b, c)
```

逻辑运算符

运算符	描述
and	布尔"与"（左边为假，返回左边；否则返回右边）
or	布尔"或"（左边为真，返回左边；否则返回右边）
not	布尔"非"（假，返回 True；真，返回 False）

```

```

```
# None, False, 0, 空列表/空元组/空字典/空集合都会被认定为
False
a = 2
b = "hello world"
c = []
d = 0

# 左边为假，返回左边；否则返回右边
print(c and a)  # []
print(c and d)  # []
print(d and c)  # 0

print(a and c)  # []
print(a and b)  # "hello world"
print(b and a)  # 2


# 左边为真，返回左边；否则返回右边
print(a or c)  # 2
print(a or b)  # 2
print(b or a)  # "hello world"

print(c or a)  # 2
print(c or d)  # 0
print(d or c)  # []


# 假，返回真；真，返回假
print(not a)  # False
print(not b)  # False
print(not c)  # True

print(not (a and c))  # True
# 优先级：not > and > or
print(not a or c)  # []
```

规律：

> *and 只要有其中一边为False，最后结果进行bool 判断一定为False*
> *只有两边都为 True 的情况下，最后结果进行bool 判断才为True*

> *or 只要有其中一边为True，最后结果进行bool 判断就为True*
> *只有两边都为False 的情况下，最后结果进行bool 判断才为False*

短路原则：如果前面的部分已经计算出整个表达式的结果，则后面的部分不再计算

```
a = 0
b = 1
print(a == 0 or b / a > a)
print(a > 0 and b / a > a)
print(a > 0 or b / a > a)
```

成员运算符

返回布尔值：True，False（判断对象中是否存在某个元素）

运算符	描述
in	在其中
not in	不在其中

```
str1 = "hello world"
list1 = [1, 2, 3, 4, 5]
tuple1 = (1, 2, 3, 4, 5)
set1 = {1, 2, 3, 4, 5}

a = "hel" in str1
b = 4 in list1
c = 4 in tuple1
d = 4 in set1
print(a,b,c,d)  # True True True True
```

```
a = "hel" not in str1
b = 4 not in list1
c = 4 not in tuple1
d = 4 not in set1
print(a,b,c,d)  # False False False False
```

身份运算符

返回布尔值：True，False

运算符	描述
is	判断两个标识符是不是引用自一个对象，类似判断 id(a) == id(b)
is not	判断两个标识符是不是引用自不同对象，类似判断 id(a) != id(b)

```
a = 257
b = 257
print(a == b)
print(a is b)
print(id(a))
print(id(b))


a = 256
b = 256
print(a == b)
print(a is b)
print(id(a))
print(id(b))
```

is 和 == 的区别

- is 是身份运算符，判断对象的地址是否相同；而 == 是比较运算符，判断对象的值是否相等

位运算符

位运算符是把数字看作二进制来进行计算的

运算符	描述
&	按位与运算符，如果两个相应位都为1，则该位的结果为1，否则为0
|	按位或运算符，只要两个相应位有一个为1时，结果位就为1
^	按位异或运算符，只要两个相应位不同，结果就为1
~	按位取反运算符，对数据的每个二进制位取反,即把1变为0,把0变为1。~x 值等于 -（x+1）
<<	左移动运算符：运算数的各二进位全部左移若干位，右边补0
>>	右移动运算符：运算数的各二进位全部右移若干位，左边补对应的符号位值

进制转换函数

bin(x)

- 将一个整数转变为一个前缀为“0b”的二进制字符串。

```
print(bin(3))  # 0b11
print(bin(-100))  # -0b1100100
```

hex(x)

- 将一个整数转变为一个前缀为“0x”的十六进制字符串。

```
print(hex(255))  # 0xff
print(hex(-42))  # -0x2a
```

oct(x)

- 将一个整数转变为一个前缀为“0o”的八进制字符串。

```
print(oct(255))  # 0o377
print(oct(-42))  # -0o52
```

```
a = 60    # 二进制 60 = 0011 1100
b = 13    # 二进制 13 = 0000 1101

c = a & b  # 12 = 0000 1100    同都为真(1和1), 才为真
print(c)

c = a | b  # 61 = 0011 1101    一个真(至少一个1), 就为真
print(c)

c = a ^ b  # 49 = 0011 0001    不同(01或者10), 就为真
print(c)

c = a ^ b ^ a  # 一个数异或于同一个数两次，结果还是它本身
print(c)  # 13
```

- 在计算机中，负数以原码的补码形式表达
- 负数的反码：对该数的原码除符号位外各位取反
- 负数的补码：对该数的原码除符号位外各位取反，然后在最后一位加1 （即反码+1）
- 正数的反码和补码都与原码相同

```
# 23的原码：0,0010111 右移两位舍弃右边的两位，最高位使用符号位填充：000,00101 还原成十进制就是5
# 右移三位舍弃右边的三位：0000,0010 还原成十进制就是2
print(23 >> 2, 23 >> 3)

# 23的原码：0,0010111 左移两位后面补两个0：0,001011100 换算成十进制就是92
```

```
# 左移三位后面补三个0：0,0010111000 换算成十进制就是184
print(23 << 2, 23 << 3)

# 因为在计算机中，负数以原码的补码形式表达，所以先算出补码，再来移位(符号位1代表负数)
# -23原码：1,0010111    -23反码：1,1101000    -23补码：1,1101001
# 补码向右移两位得到：111,11010  这个还是补码形式，需要先转成原码再换算成十进制
# 补码转成原码就是逆过程：111,11010减一再取反码就是原码了，得到：111,00110 换算成十进制就是-6

# 补码向右移三位得到：1111,1101  这个还是补码形式，需要先转成原码再换算成十进制
# 补码转成原码就是逆过程：1111,1101减一再取反码就是原码了，得到：1111,0011 换算成十进制就是-3

# 补码向左移两位得到：1,110100100  这个还是补码形式，需要先转成原码再换算成十进制
# 补码转成原码就是逆过程：1,110100100减一再取反码就是原码了，得到：1,001011100 换算成十进制就是-92

# 补码向左移三位得到：1,1101001000  这个还是补码形式，需要先转成原码再换算成十进制
# 补码转成原码就是逆过程：1,1101001000减一再取反码就是原码了，得到：1,0010111000 换算成十进制就是-184
print(-23 >> 2, -23 >> 3)
print(-23 << 2, -23 << 3)
```

- 按位取反运算符，对数据的每个二进制位取反（包括符号位）

```
# 23的原码：0,0010111    按位取反：1,1101000
# 按位取反之后变成了负数，负数在计算机中以原码的补码形式表达,所以需要先转成原码再换算成十进制
# 补码转成原码就是逆过程：1,1101000减一再取反码就是原码了，得到：1,0011000 换算成十进制就是-24
print(~23)

# 负数在计算机中以原码的补码形式表达,所以先算出补码，再来按位取反
# -23原码：1,0010111    -23反码：1,1101000    -23补码：1,1101001
# 再来按位取反得到：0,0010110   按位取反之后变成了正数，正数直接换算成十进制就是22
print(~-23)
```

运算符优先级

以下表格列出了从高到低优先级的常用运算符：

运算符	描述
**	指数 (最高优先级)
~	按位取反
* / % //	乘，除，求余数和取整除
+ -	加法、减法
>> <<	右移，左移
&	按位与
^ |	按位异或、或
<= < > >=	比较运算符
== !=	等于运算符
= %= /= //= -= += *= **=	赋值运算符
is is not	身份运算符
in not in	成员运算符

运算符	描述
not and or	逻辑运算符

+、+=、* 的连接操作

+、+=、* 还可以对字符串、列表、元组进行连接操作（要保证连接对象的类型一致）

```
a = "1"
b = "2"
c = a + b
print(c)  # '12'

a = ["1"]
b = [2, 3]
c = a + b
print(c)  # ['1', 2, 3]

a = ("1",)
b = (2, 3)
c = a + b
print(c)  # ('1', 2, 3)


a = "1"
b = "2"
a += b  # a = a + b
print(a)  # '12'

a = ["1"]
b = [2, 3]
a += b
print(a)  # ['1', 2, 3]
```

```python
a = ("1",)
b = (2, 3)
a += b
print(a)  # ('1', 2, 3)


a = "1"
b = a * 3
print(b)  # "111"

a = ["1"]
b = a * 3
print(b)  # ['1', '1', '1']

a = ("1",)
b = a * 3
print(b)  # ('1', '1', '1')
```