

Introduction to Programming and Data Structures

Python – Operators and Expressions

Malay Bhattacharyya

Associate Professor

MIU, CAIML, TIH
Indian Statistical Institute, Kolkata

August, 2023



1 Operators and Expressions

Arithmetic operators

Summation (+)

Subtraction (−)

Multiplication (*)

Power ()**

Float Division (/)

Floor Division (//)

Modulo division (%)

Arithmetic operators

Summation (+)

Subtraction (−)

Multiplication (*)

Power ()**

Float Division (/)

Floor Division (//)

Modulo division (%)

Note: Modulo division operator works on any type of numbers (including floating point values and negatives!!!) and returns the sign of the denominator (with a different kind of computation).

Modular division on negative values

What will be the output of the following program?

```
print(4 / 3, 4 // 3, 4 % 3)
print(-4 / 3, -4 // 3, -4 % 3)
print(4 / -3, 4 // -3, 4 % -3)
print(-4 / -3, -4 // -3, -4 % -3)
```

Output:

```
1.3333333333333333 1 1
-1.3333333333333333 -2 2
-1.3333333333333333 -2 -2
1.3333333333333333 1 -1
```

Modulo division on floating point values

What will be the output of the following program?

```
print(4.3 % 3)
print(4 % 3.5)
print(4.3 % 3.5)
print(4 % 3)
```

Modulo division on floating point values

What will be the output of the following program?

```
print(4.3 % 3)
print(4 % 3.5)
print(4.3 % 3.5)
print(4 % 3)
```

Output:

```
1.2999999999999998
0.5
0.7999999999999998
1
```

Note: $x\%y$ returns r (following IEEE 754, the IEEE Standard for floating-point arithmetic) if and only if $x = n * y + r$, where n is an integer, r has the same sign as y , and $|r| < |y|$.

Modulo division on Boolean values

What will be the output of the following program?

```
var1 = True
var2 = False
print(var1 % var1)
print(var1 % var2)
print(var2 % var1)
print(var2 % var2)
```

Modulo division on Boolean values

What will be the output of the following program?

```
var1 = True
var2 = False
print(var1 % var1)
print(var1 % var2)
print(var2 % var1)
print(var2 % var2)
```

Output:

0

Error

0

Error

Relational operators

Less than ($<$)

Less than equals to ($<=$)

Greater than ($>$)

Greater than equals to ($>=$)

Equals to ($==$)

Not equals to ($!=$)

Logical operators

Logical and (and)

Logical or (or)

Logical not (not)

Assignment operators

Assignment (=)

Addition and assignment (+ =)

Subtraction and assignment (- =)

Multiplication and assignment (* =)

Power and assignment (** =)

Float Division and assignment (/ =)

Floor Division and assignment (// =)

Modulo division and assignment (% =)

Assignment operators

What will be the output of the following program?

```
a = 10
b = 20
print(a, b)
print(hex(id(a)), hex(id(b)))
a, b = b, a # Swapping values
print(a, b)
print(hex(id(a)), hex(id(b)))
```

Assignment operators

What will be the output of the following program?

```
a = 10
b = 20
print(a, b)
print(hex(id(a)), hex(id(b)))
a, b = b, a # Swapping values
print(a, b)
print(hex(id(a)), hex(id(b)))
```

Output:

```
10 20
0x72fca540 0x72fca8e0
20 10
0x72fca8e0 0x72fca540 # Only the pointers will change!!!
```

Bitwise operators

Bitwise and (&)

Bitwise or (|)

Bitwise not (~)

Bitwise xor (^)

Left shift (<<)

Right shift (>>)

Bitwise operators

```
i = 11
print(i>>1) # 1 place right shift (i unchanged)
print(i<<2) # 2 places left shift (i unchanged)
```

Bitwise operators

```
i = 11
print(i>>1) # 1 place right shift (i unchanged)
print(i<<2) # 2 places left shift (i unchanged)
```

Output:

```
5
44
```

Bitwise operators

```
i = 11
print(i>>1) # 1 place right shift (i unchanged)
print(i<<2) # 2 places left shift (i unchanged)
```

Output:

5

44

Decimal	Binary							
11	0	0	0	0	1	0	1	1
5	0	0	0	0	0	1	0	1
44	0	0	1	0	1	1	0	0

Efficiency of bitwise operators

Verifying whether a number is a power of 2 with bitwise AND:

```
n = 32
if n & n-1:
    print(n, "is not a power of 2")
else:
    print(n, "is a power of 2")
```

Efficiency of bitwise operators

Verifying whether a number is a power of 2 with bitwise AND:

```
n = 32
if n & n-1:
    print(n, "is not a power of 2")
else:
    print(n, "is a power of 2")
```

Output:

```
32 is a power of 2
```

Efficiency of bitwise operators

Verifying similarity of signs with bitwise XOR:

```
m = 10
n = -20
if m ^ n < 0:
    print(m, " and ", n, " have different signs");
else:
    print(m, " and ", n, " have the same signs");
```

Efficiency of bitwise operators

Verifying similarity of signs with bitwise XOR:

```
m = 10
n = -20
if m ^ n < 0:
    print(m, " and ", n, " have different signs");
else:
    print(m, " and ", n, " have the same signs");
```

Output:

```
10  and -20  have different signs
```

Identity operators

Identical (is)

Not identical (is not)

Identity operators

Identical (`is`)

Not identical (`is not`)

Note: Two variables that are equal does not imply that they are identical. For being identical, they must be located on the same part of the memory.

Identity operators

What will be the output of the following program?

```
var1 = 123
var2 = 123
print(var1 is var2)
var1 = 'Python'
var2 = "Python"
print(var1 is not var2)
var1 = [1, 2, 3]
var2 = [1, 2, 3]
print(var1 is var2)
```

Identity operators

What will be the output of the following program?

```
var1 = 123
var2 = 123
print(var1 is var2)
var1 = 'Python'
var2 = "Python"
print(var1 is not var2)
var1 = [1, 2, 3]
var2 = [1, 2, 3]
print(var1 is var2)
```

Output:

True

False

False

Identity operators

What will be the output of the following program?

```
var1 = 123
var2 = 123
print(hex(id(var1)), hex(id(var2)))
var1 = 'Python'
var2 = "Python"
print(hex(id(var1)), hex(id(var2)))
var1 = [1, 2, 3]
var2 = [1, 2, 3]
print(hex(id(var1)), hex(id(var2)))
```

Identity operators

What will be the output of the following program?

```
var1 = 123
var2 = 123
print(hex(id(var1)), hex(id(var2)))
var1 = 'Python'
var2 = "Python"
print(hex(id(var1)), hex(id(var2)))
var1 = [1, 2, 3]
var2 = [1, 2, 3]
print(hex(id(var1)), hex(id(var2)))
```

Output:

```
0x812ee0 0x812ee0
0x9227e8 0x9227e8
0x27f9430 0x31b8740
```

Some comments

The following popular operators are **not** available in Python:

- **Increment** (`++`)
- **Decrement** (`--`)
- **Comma** (`,`)

Use of comma

What will be the output of the following program?

```
var = 20, 30, 40
print(var)
var = (50, 60, 70)
print(var)
```

Use of comma

What will be the output of the following program?

```
var = 20, 30, 40
print(var)
var = (50, 60, 70)
print(var)
```

Output:

```
(20, 30, 40)
(50, 60, 70)
```

Operator precedence (highest to lowest)

Associativity	Operator	Description
Left-to-right	()	Parentheses (grouping)
Left-to-right	f(args...)	Function call
Left-to-right	x[index:index]	Slicing
Left-to-right	x[index]	Array Subscription
Right-to-left	**	Exponentiation
Left-to-right	~x	Bitwise not
Left-to-right	+x -x	Positive, Negative
Left-to-right	* / %	Multiplication Division Modulo
Left-to-right	+ -	Addition Subtraction
Left-to-right	<< >>	Bitwise left shift Bitwise right shift
Left-to-right	&	Bitwise AND
Left-to-right	^	Bitwise XOR
Left-to-right		Bitwise OR
Left-to-right	in, not in, is, is not, <, <=, >, >=, <>, == !=	Membership Relational Equality Inequality
Left-to-right	not x	Boolean NOT
Left-to-right	and	Boolean AND
Left-to-right	or	Boolean OR
Left-to-right	lambda	Lambda expression