Python-Operators

Er. Narayan Sapkota
M.Sc. Computational Science and Engineering

Kathmandu University (Visiting Faculty)

December 11, 2024

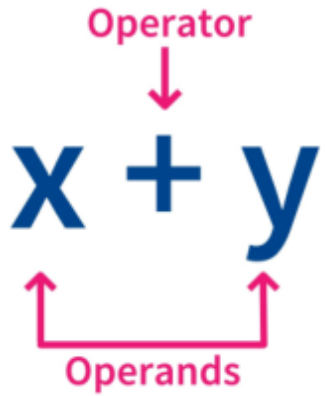
Table of Contents

1. Operators
   - Arithmetic Operators
   - Assignment Operators
   - Comparison Operators
   - Logical Operators
   - Bitwise Operators
2. Operator Precedence and Associativity

Operators

- Operators are special symbols used to perform operations on variables and values.
- They perform certain mathematical or logical operation to manipulate data values and produce a result based on some rules.
- An operator manipulates the data values called operands.

1. Arithmetic Operators
2. Assignment Operators
3. Comparison Operators
4. Logical Operators
5. Bitwise Operators



Arithmetic Operators (1)

List of Arithmetic Operators

Operator	Operator Name	Description	Example
+	Addition operator	Adds two operands, producing their sum.	p + q = 5
–	Subtraction operator	Subtracts the two operands, producing their difference.	p – q = –1
*	Multiplication operator	Produces the product of the operands.	p * q = 6
/	Division operator	Produces the quotient of its operands where the left operand is the dividend and the right operand is the divisor.	q / p = 1.5
%	Modulus operator	Divides left hand operand by right hand operand and returns a remainder.	q % p = 1
**	Exponent operator	Performs exponential (power) calculation on operators.	p**q = 8
//	Floor division operator	Returns the integral part of the quotient.	9//2 = 4 and 9.0//2.0 = 4.0

Note: The value of p is 2 and q is 3.

Arithmetic Operators (2)

Program 1: Python Arithmetic Operators

```
p = 2
q = 3
# Addition
sum = p + q
# Subtraction
diff = p - q
# Multiplication
prod = p * q
# Division
div = p / q
# Floor Division
floor_div = p // q
# Modulus
mod = p % q
# Exponentiation
exp = p ** q
print("Addition:", sum)
```

Arithmetic Operators (3)

```
print("Subtraction:", diff)
print("Multiplication:", prod)
print("Division:", div)
print("Floor Division:", floor_div)
print("Modulus:", mod)
print("Exponentiation:", exp)
```

Output:

```
Addition:  5
Subtraction:  -1
Multiplication:  6
Division:  0.6666666666666666
Floor Division:  0
Modulus:  2
Exponentiation:  8
```

Assignment Operators (1)

List of Assignment Operators

Operator	Operator Name	Description	Example
=	Assignment	Assigns values from right side operands to left side operand.	z = p + q assigns of p + q to z
+=	Addition Assignment	Adds the value of right operand to the left operand and assigns the result to left operand.	z += p is equivalent to z = z + p
−=	Subtraction Assignment	Subtracts the value of right operand from the left operand and assigns the result to left operand.	z −= p is equivalent to z = z − p
*=	Multiplication Assignment	Multiplies the value of right operand with the left operand and assigns the result to left operand.	z *= p is equivalent to z = z * p
/=	Division Assignment	Divides the value of right operand with the left operand and assigns the result to left operand.	z /= p is equivalent to z = z / p
**=	Exponentiation Assignment	Evaluates to the result of raising the first operand to the power of the second operand.	z**= p is equivalent to z = z ** p
//=	Floor Division Assignment	Produces the integral part of the quotient of its operands where the left operand is the dividend and the right operand is the divisor.	z //= p is equivalent to z = z // p
%=	Remainder Assignment	Computes the remainder after division and assigns the value to the left operand.	z %= p is equivalent to z = z % p

Assignment Operators (2)

Program 2: Python Assignment Operators

```
# Initial value of z
z = 10
print("Initial value of z:", z)

# Assignment Operator (=)
z = 20
print("After z = 20:", z)

# Addition Assignment (+=)
z += 5  # Equivalent to z = z + 5
print("After z += 5:", z)

# Subtraction Assignment (-=)
z -= 3  # Equivalent to z = z - 3
print("After z -= 3:", z)

# Multiplication Assignment (*=)
```

Assignment Operators (3)

```
z *= 2  # Equivalent to z = z * 2
print("After z *= 2:", z)

# Division Assignment (/=)
z /= 4  # Equivalent to z = z / 4
print("After z /= 4:", z)

# Exponentiation Assignment (**=)
z **= 2  # Equivalent to z = z ** 2
print("After z **= 2:", z)

# Floor Division Assignment (//=)
z //= 3  # Equivalent to z = z // 3
print("After z //= 3:", z)

# Remainder Assignment (%=)
z %= 3  # Equivalent to z = z % 3
print("After z %= 3:", z)
```

Assignment Operators (4)

Output:

```
Initial value of z:  10
After z = 20:  20
After z += 5:  25
After z -= 3:  22
After z *= 2:  44
After z /= 4:  11.0
After z **= 2:  121.0
After z //= 3:  40.0
After z %= 3:  1.0
```

Comparison Operators (1)

List of Comparison Operators

Operator	Operator Name	Description	Example
==	Equal to	If the values of two operands are equal, then the condition becomes True.	(p == q) is not True.
!=	Not Equal to	If values of two operands are not equal, then the condition becomes True.	(p != q) is True
>	Greater than	If the value of left operand is greater than the value of right operand, then condition becomes True.	(p > q) is not True.
<	Lesser than	If the value of left operand is less than the value of right operand, then condition becomes True.	(p < q) is True.
>=	Greater than or equal to	If the value of left operand is greater than or equal to the value of right operand, then condition becomes True.	(p >= q) is not True.
<=	Lesser than or equal to	If the value of left operand is less than or equal to the value of right operand, then condition becomes True.	(p <= q) is True.

Note: The value of p is 10 and q is 20.

Comparison Operators (2)

Program 3: Python Comparison Operators

```
# Initial values
p = 5
q = 3

# Equal to (==)
print("p == q:", p == q)  # False, since p is not equal to q

# Not equal to (!=)
print("p != q:", p != q)  # True, since p is not equal to q

# Greater than (>)
print("p > q:", p > q)  # True, since p is greater than q

# Less than (<)
print("p < q:", p < q)  # False, since p is not less than q

# Greater than or equal to (>=)
```

Comparison Operators (3)

```
print("p >= q:", p >= q)  # True, since p is greater than or
    equal to q

# Less than or equal to (<=)
print("p <= q:", p <= q)  # False, since p is not less than
   or equal to q
```

Output:

```
p == q:  False
p != q:  True
p > q:  True
p < q:  False
p >= q:  True
p <= q:  False
```

Logical Operators (1)

List of Logical Operators

Operator	Operator Name	Description	Example
and	Logical AND	Performs AND operation and the result is True when both operands are True	p and q results in False
or	Logical OR	Performs OR operation and the result is True when any one of both operand is True	p or q results in True
not	Logical NOT	Reverses the operand state	not p results in False

Note: The Boolean value of p is True and q is False.

Logical Operators (2)

Program 4: Python Logical Operators

```
p = True
q = False
# Logical AND (and)
print("p and q:", p and q)  # False, because q is False
# Logical OR (or)
print("p or q:", p or q)  # True, because p is True
# Logical NOT (not)
print("not p:", not p)  # False, because p is True
print("not q:", not q)  # True, because q is False
```

Output:

```
p and q:  False
p or q:  True
not p:  False
not q:  True
```

Bitwise Operators (1)

List of Bitwise Operators

Operator	Operator Name	Description	Example
&	Binary AND	Result is one in each bit position for which the corresponding bits of both operands are 1s.	p & q = 12 (means 0000 1100)
|	Binary OR	Result is one in each bit position for which the corresponding bits of either or both operands are 1s.	p | q = 61 (means 0011 1101)
^	Binary XOR	Result is one in each bit position for which the corresponding bits of either but not both operands are 1s.	(p ^ q) = 49 (means 0011 0001)
~	Binary Ones Complement	Inverts the bits of its operand.	(~p) = −61 (means 1100 0011 in 2's complement form due to a signed binary number.
<<	Binary Left Shift	The left operands value is moved left by the number of bits specified by the right operand.	p << 2 = 240 (means 1111 0000)
>>	Binary Right Shift	The left operands value is moved right by the number of bits specified by the right operand.	p >> 2 = 15 (means 0000 1111)

Note: The value of p is 60 and q is 13.

Bitwise Operators (2)

Bitwise Truth Table

P	Q	P & Q	P | Q	P ^ Q	~ P
0	0	0	0	0	1
0	1	0	1	1	
1	0	0	1	1	0
1	1	1	1	0	

Bitwise Operators (3)

Bitwise and (&)				**Bitwise or (|)**			
a	=	0011 1100	→ (60)	a	=	0011 1100	→ (60)
b	=	0000 1101	→ (13)	b	=	0000 1101	→ (13)
a & b	=	0000 1100	→ (12)	a | b	=	0011 1101	→ (61)
Bitwise exclusive or (^)				**One's Complement (~)**			
a	=	0011 1100	→ (60)	a	=	0011 1100	→ (60)
b	=	0000 1101	→ (13)	~ a	=	1100 0011	→ (-61)
a ^ b	=	0011 0001	→ (49)				
Binary left shift (<<)				**Binary right shift (>>)**			
a	=	0011 1100	→ (60)	a	=	0011 1100	→ (60)
a << 2	=	1111 0000	→ (240)	a >> 2	=	0000 1111	→ (15)
left shift of 2 bits				right shift of 2 bits			

Figure 1: Examples for bitwise logical operations. The value of operand a is 60 and value of operand b is 13

Bitwise Operators (4)

Program 5: Python Bitwise Operators

```
p = 60   # 60 = 0011 1100 in binary
q = 13   # 13 = 0000 1101 in binary

# Binary AND (&)
print("p & q:", p & q)   # 12 = 0000 1100 in binary

# Binary OR (|)
print("p | q:", p | q)   # 61 = 0011 1101 in binary

# Binary XOR (^)
print("p ^ q:", p ^ q)   # 49 = 0011 0001 in binary

# Binary Ones Complement (~)
print("~p:", ~p)   # -61 = 1100 0011 in binary (2's
    complement)

# Left Shift (<<)
```

Bitwise Operators (5)

```
print("p << 2:", p << 2)   # 240 = 1111 0000 in binary

# Right Shift (>>)
print("p >> 2:", p >> 2)   # 15 = 0000 1111 in binary
```

Output:

```
p & q:  12
p | q:  61
p q̂:  49
p:  -61
p << 2:  240
p >> 2:  15
```

Operator Precedence and Associativity (1)

Operator precedence determines the way in which operators are parsed with respect to each other. Operators with higher precedence become the operands of operators with lower precedence. Associativity determines the way in which operators of the same precedence are parsed. Almost all the operators have left-to-right associativity apart from exponent (right-to-left associativity).

Operator Precedence and Associativity (2)

Operator Precedence in Python

Operators	Meaning
()	Parentheses
**	Exponent
+x, −x, ~x	Unary plus, Unary minus, Bitwise NOT
*, /, //, %	Multiplication, Division, Floor division, Modulus
+, −	Addition, Subtraction
<<, >>	Bitwise shift operators
&	Bitwise AND
^	Bitwise XOR
|	Bitwise OR
==, !=, >, >=, <, <=, is, is not, in, not in	Comparisons, Identity, Membership operators
not	Logical NOT
and	Logical AND
or	Logical OR

Operator Precedence and Associativity (3)

Program 6: Operator Precedence and Associativity

```
a = 5
b = 3
c = 2

# Using exponentiation (right-to-left associativity)
result1 = a ** b ** c
# Evaluates as a ** (b ** c) -> 5 ** (3 ** 2) -> 5 ** 9 =
    1953125
print("a ** b ** c =", result1)

# Using multiplication and addition (left-to-right
    associativity)
result2 = a + b * c
# Evaluates as a + (b * c) -> 5 + (3 * 2) -> 5 + 6 = 11
print("a + b * c =", result2)

# Using parentheses to change precedence
```

Operator Precedence and Associativity (4)

```
result3 = (a + b) * c
# Evaluates as (a + b) * c -> (5 + 3) * 2 = 8 * 2 = 16
print("(a + b) * c =", result3)

# Using logical AND and OR (left-to-right associativity)
result4 = a > b and b < c or a == b
# Evaluates as (a > b and b < c) or a == b -> (5 > 3 and 3 <
    2) or 5 == 3
# -> (True and False) or False -> False or False -> False
print("a > b and b < c or a == b =", result4)
```

Output:

```
a ** b ** c = 1953125
a + b * c = 11
(a + b) * c = 16
a > b and b < c or a == b = False
```