PYTHON FOR COMPUTATIONAL PROBLEM SOLVING

Operators and Expressions

Prof. Sindhu R Pai
PCPS Theory Anchor - 2024
Department of Computer Science and Engineering

Introduction

- **An operator** is a symbol that represents an operation that may be performed on one or more *operands.*
- An operand is a value that a given operator is applied to.
- An expression is a combination of symbols that evaluates to a value.
- **Expressions,** most commonly, consist of a combination of operators and operands

  4 + (3 * k)
- It can also consist of a single literal or variable. Thus, 4, 3, and k are each expressions
- Expressions that evaluate to a numeric type are called arithmetic expressions

Categories of Operators

1. Based on the type of Operation

- Arithmetic Operators (+ , - , * , / , // , % , **)
- Relational Operators (== , != , < , <= , > , >=)
- Membership Operators (in , not in) (and , or , not)
- Boolean (Logical) Operators
- Bitwise operators (& , | , ^ , >> , << , ~)
- Identity Operators (is , is not)
- Assignment operators
  - shorthand operators (+= , -=, *= , /= , //= , %= , **=)

2. Based on the number of operands

- Unary
- Binary
- Ternary

Operators

Arithmetic Operators

Operator	Expression	Name
-	-x	Negation
+	x + y	Addition
-	x - y	Subtraction
*	x * y	Multiplication
**	x ** y	Exponentiation
/	x / y	Division
//	x // y	Truncation Division
%	x % y	Modulus

Arithmetic Operators – Division

Python provides two forms of division:

- **"True" division** is denoted by a single slash, **/**

  Thus, **25 / 10** evaluates to **2.5**

- **"Truncating" division** is denoted by a double slash, **//**

  Thus, **25 // 10** evaluates to **2**

Arithmetic Operators – Division

Truncating division provides a truncated result based on the type of operands applied to

- When both operands are integer values, the result is a truncated integer referred to as **integer division**.
- When as least one of the operands is a float type, the result is a **truncated floating point**.

Example:

```
>>> 5//2
2
>>> 5//2.0
2.0
```

PES UNIVERSITY

Arithmetic Operators - Division

	Operands	result type	example	result
/ Division operator	int, int	float	`7 / 5`	`1.4`
	int, float	float	`7 / 5.0`	`1.4`
	float, float	float	`7.0 / 5.0`	`1.4`
// Truncating division operator	int, int	truncated int ("integer division")	`7 // 5`	`1`
	int, float	truncated float	`7 // 5.0`	`1.0`
	float, float	truncated float	`7.0 // 5.0`	`1.0`

PES UNIVERSITY

Arithmetic Operators - Modulus Operator

Modulus operator (%) gives the remainder of the division of its operands, resulting in a cycle of values

Modulo 7		Modulo 10		Modulo 100	
0 % 7	**0**	0 % 10	**0**	0 % 100	**0**
1 % 7	**1**	1 % 10	**1**	1 % 100	**1**
2 % 7	**2**	2 % 10	**2**	2 % 100	**2**
3 % 7	**3**	3 % 10	**3**	3 % 100	**3**
4 % 7	**4**	4 % 10	**4**	.	.
5 % 7	**5**	5 % 10	**5**	.	.
6 % 7	**6**	6 % 10	**6**	96 % 100	**96**
7 % 7	0	7 % 10	**7**	97 % 100	**97**
8 % 7	1	8 % 10	**8**	98 % 100	**98**
9 % 7	2	9 % 10	**9**	99 % 100	**99**
10 % 7	3	10 % 10	0	100 % 100	0
11 % 7	4	11 % 10	1	101 % 100	1
12 % 7	5	12 % 10	2	102 % 100	2

Think! - Does % operator works on float values and negative values?

Operators

Relational Operators

Used to compare two values.

Relational expressions are a type of **Boolean expression**, since they evaluate to a Boolean result

Relational Operators	Example	Result
== equal	10 == 10	True
!= not equal	10 != 10	False
< less than	10 < 20	True
> greater than	'Alan' > 'Brenda'	False
<= less than or equal to	10 <= 10	True
>= greater than or equal to	'A' >= 'D'	False

Operators

PES UNIVERSITY

Relational Operators

- **Simple comparison**

```
10 == 10        True
3 > 2           True
```

- **Cascading comparison**

a op1 **b** op2 **c** is the same as (a op1 b) and (b op2 c)

3 > 2 > 1 is the same as **(3>2)** and **(2>1)** **True**

PES UNIVERSITY

Relational Operators

- **String comparison:**
  - Compares the corresponding characters based on the ASCII value.
  - The ord() function returns the number representing the unicode code of a specified character.

“ cat ” > “car”	# True : “ t ” > “ r ”
“ cat ” > “ cattle"	# False : Second string is longer
“ cat ” == “Cat”	# False : “ C ” < “ c ”
“ apple ” > “z”	# False :Comparison not based on the length
“ zebra ” > “abcedefgh”	# True “ z ” > a ”

Relational Operators

- **List comparison:**

Rules are same as that of string - compare the corresponding elements until a mismatch or one or both ends

```
[10, 20, 30] > [10, 25]                    # False  20 > 25 is false
[(10, 20), "abcd" ] >[(10, 20), "abcc" ]   # True  d of abcd > last c of abcc
```

Membership Operators

- These operators can be used to determine if a particular value occurs within a specified collection of values.

Membership Operators	Examples	Result
in	10 in (10, 20, 30)	True
	red in ('red','green','blue')	True
not in	10 not in (10, 20, 30)	False

- The membership operators can also be used to check if a given string occurs within another string

```
>>> 'E' in 'PES'
True
```

Boolean (Logical) Operators

- Boolean algebra contains a set of **Boolean (logical) operators**
- Denoted by **and**, **or**, and **not**.
- These logical operators can be used to construct arbitrarily complex Boolean expressions

x	y		x and y	x or y	not x
False	False		False	False	True
True	False		False	True	False
False	True		False	True	
True	True		True	True	

Boolean (Logical) Operators

- **False Values**: 0 , ‘’ (Empty String), [] , {} , () (Empty Collections), None, False
- **True Values:** non – Zero numbers , Non Empty String, Non Empty Collections, True

PES UNIVERSITY

Boolean (Logical) Operators

Short Circuit Evaluation

- logical and, if the first operand evaluates to false, then regardless of the value of the second operand, the expression is false
- logical or, if the first operand evaluates to true, regardless of the value of the second operand, the expression is true.
- Python interpreter does not evaluate the second operand when the result is known by the first operand alone
- This is called **short-circuit (lazy) evaluation**

PES UNIVERSITY

Bitwise Operators

Operations are performed at the bit level

Operator	Name	Result
&	AND	result is 1 if the corresponding bits are one
|	OR	result is 1 if even at least one of the bits is one
^	Exclusive OR	result is 1 if and only if one of the bits is 1
<<	LEFT SHIFT	multiply by 2 for each left shift
>>	RIGHT SHIFT	divide by 2 for each right shift
~	ONE'S COMPLIMENT	change 0 to 1 and 1 to 0

Operators

Bitwise Operators

& AND

```
a = 5          # 0101
b = 6          # 0110
c = a & b      # 0100        =4
```

| OR

```
a = 5          # 0101
b = 6          # 0110
c = a | b      # 0111        =7
```

^ XOR

```
a = 5          # 0101
b = 6          # 0110
c = a ^ b      # 0011        =3
```

Bitwise Operators

>> (Right shift operator)

```
a=250
a>>3     #Right shift a by three bits
31       #Answer is 31
```

Working:

1	1	1	1	1	0	1	0			**=250**
			1	1	1	1	1	~~0~~	~~1~~	~~0~~ **=31**

These bits are lost

Bitwise Operators

<< (Left Shift)

```
a=25
a<<2    #Left shift a by 2 bits
100
```

Working:

		1	1	0	0	1	=25
1	1	0	0	1	0	0	=100

Left shift by 2 places and insert zeros in the emptied places

Identity Operators

- **is**
- **is not**

Checks if the operands on either side of the operator point to the same object or not

```
>>> a=10;b=10
>>> a is b
True
>>> a=10; b=10.0
>>> a is b
False
>>> a is not b
True
```

Assignment / Shorthand Operators

Combines arithmetic and assignment operators

Operator	Expression	Short Hand
+= (Addition)	a = a + b	a += b
-= (Subtraction)	a = a - b	a -= b
***=** (Multiplication)	a = a * b	a *= b
/= (Division)	a = a / b	a /= b
//= (Truncation Division)	a = a // b	a //= b
%= (Modulus)	a = a % b	a %= b
****=** (Exponentiation)	a = a ** b	a **= b

THANK YOU

Department of Computer Science and Engineering

Dr. Shylaja S S, Director, CCBD & CDSAML, PESU
Prof. Sindhu R Pai – sindhurpai@pes.edu
Prof. Chitra G M
Prof. Gayatri S