PYTHON FOR COMPUTATIONAL PROBLEM SOLVING

Prof. Gayathri R S

Computer Science and Engineering

Operator Precedence

- Determines the order of evaluation.
- Each programming language has its own rules for the order that operators are applied, called operator precedence
- Consider the following expression:

4+3*5

There are two possible ways in which it can be evaluated

4 + 3 * 5 → 4 + 15 → **19**

~~4 + 3 * 5 → 7 * 5 → **35**~~

Operator Precedence

- Operator precedence guarantees a consistent interpretation of expressions

  * has higher precedence than +. Therefore the expression will be evaluated as follows.

  4 + 3 * 5 → 4 + 15 → **19**

- If the addition is to be performed first, parentheses would be needed,

  (4 + 3) * 5 → 7 * 5 → **35**

- It is good programming practice to use parentheses even when not needed

Operator Associativity

Operator associativity defines the order that it and other operators with the same level of precedence are evaluated.

For example,

the associativity of exponentiation operator is right to left.

Therefore, 23**2** will be evaluated as follows

2 ** (3 ** 2) → **512**

(2 ** 3) ** 2 → **64**

Operator Precedence and Associativity

- The following table summarizes the operator precedence in Python, from highest precedence to lowest precedence. Operators in the same box have the same precedence.

- **Operators in the same box group left to right (except for exponentiation, which groups from right to left**).

- Note that comparisons, membership tests, and identity tests, all have the same precedence and have a left-to-right chaining feature.

- To see the complete documentation of operator precedence execute the below statement in Python.

print(help('>'))

PYTHON FOR COMPUTATIONAL PROBLEM SOLVING

Operator Precedence

PES UNIVERSITY

Operator	Description
(expressions...)	Parenthesized expression
**	Exponentiation
"+x", "-x", "~x"	Positive, negative, bitwise NOT
"*", "/", "//", "%"	Multiplication, division, floor division, remainder
"+", "-"	Addition and subtraction
"<<", ">>"	Shifts
"&"	Bitwise AND
"^"	Bitwise XOR
"|"	Bitwise OR
"in", "not in", "is", "is not", "<", "<=", ">", ">=", "!=", "=="	Comparisons, including membership tests and identity tests
"not x"	Boolean NOT
"and"	Boolean AND
"or"	Boolean OR

THANK YOU

Prof. Gayathri R S
gayathrirs@pes.edu
Department of Computer Science and Engineering