Introduction to Programming and Problem Solving

Week 4: Operators and Expressions

Lecturer: Lemi Agrey Oliver

Department of Information Technology

Kumi University

Recap of week 3

- In our previous lecture, we introduced the basics of Python programming. We covered several fundamental concepts, that includes variables, which we explained are references to memory addresses.
- Furthermore, we explored both the **print()** and **input()** functions. The **print()** function is employed to display information on the console, allowing us to output data to the user.

Recap of week 3+

- On the other hand, the **input()** function is used for obtaining user inputs interactively.
- These functions play crucial roles in the interaction between a Python program and its users
- Additionally, we discussed tokens and provided a number of examples including delimiters, variables, and operators. In this week's lecture, our primary topic of discussion will be operators in Python, delving deeper into their usage and significance.

Content

- Introduction to Operators and Expressions
- Operators precedence and Associativity
- Changing operators precedence and Associativity of Arithmetic Operators
- Translating Mathematical Formulae into Equivalent Python Expressions
- Bitwise Operator
- The Compound Assignment Operator

Introduction to Operators

- An operator is a symbol that performs an operations on an operand while and operand is the data or value that is being operated by the operator
- Or Operators are constructs used to modify the values of operands[2]
- 4+6
- Here the values 4 and 6 are the operands which are acted upon by the plus(+) operator to produce 10 as a final result.
- Note also that the (+) operator can be used to add or concatenate two strings in which case it becomes a concatenation operator
- In our daily interactions with our computers and the environment, we interfaced with various types of operators to carry out different types of operations

Types of Operators

Python has a set of operators as categorized in the table here under

Table showing the different operators used in python

S/no	Type of Operator	Operators symbol
01	Arithmetic Operators	+, -, *, /, %, **, //
02	Comparison Operators	==, !=, <, >, <=, >=
03	Logical Operators	And, or, not
04	Assignment Operators	=, +=, -=,*=, %=, **=, //=
05	Membership Operators:	In, not in
06	Identity operator	is, is not
07	Bitwise Operators	&, |, ^, ~, <<, >>

Figure 1.0 Python Operators

OPERATORS AND EXPRESSIONS

- An expression is block of code that produces results or value upon evaluation[1]
- 7+8 is an expression that consist of both an operator and operands
- **Arithmetic operators**: These operators are used to perform basic mathematical operations, such as addition, subtraction, multiplication, and division. They are sub divided into, unary operators and binary operators. The unary operator's operates on one operand. For example x=-1, y=+4
- while its binary counter part operates on at least two operands as can be seen below

1. x = 7
2. y = 5

OPERATORS AND EXPRESSIONS +

3. print(x + y) # the + operator can also be used to concatenated strings
4. print(x - y) # 2
5. print(x * y) # 35
6. print(x / y) # 1.4
7. Print(x**y)#This means y raised to power x which when computed will give us 16807
8. print(x//y)#this is a flour division which disregards the decimal place, giving us an output of 1
9. Print(x%y)#This is a modulo operator which only returns remainder 2

OPERATORS AND EXPRESSIONS

- **The modulo operator** is an arithmetic operator which is used for finding remainder when we divide one integer by an other
- It is represented by the percentage(%) sign
- The modulo operator helps us in so many things including extracting even or odd numbers

```
1. a=19
2. b=4
3. c=a%b
4. print(c)
```

- Output:

```
3
```

The program to show remaining fruits after distribution

- **Problem:** You have 8 oranges and 9 bananas to distribute to 6 student colleagues with each student receiving at least a banana and an orange
- Write a program to show the remaining oranges and bananas after distribution

```
1. oranges=8
2. banana=9
3. roran=oranges%6
4. rban=bananas%6
5. print("Remaining banana after distribution ", rban)
6. print("Remaining oranges after distribution ", roran)
```

Output:

Remaining banana after distribution 3

Remaining orange after distribution 2

A Program to calculate the profits of an item in Uganda shillings

- Let us start to work on the above project to deepen our understanding of the arithmetic operators
- But first we need an algorithm
- Algorithm steps:
- Step 1: Take the item's buying price from the user
- Step 2: Take the selling price of the item from the user
- Step 3: subtract the buying price from the selling price
- Step 4: Output the difference to the console

A Program to calculate the profits an item in Uganda shillings+

```
1. bprice=eval(input("Please enter the buying price"))
2. sprice=eval(input("Please enter the selling price"))
3. profits=sprice-bprice
4. print("Buying price :  ",bprice,"Uganda Shillings")
5. print("Selling price: ", sprice,"Uganda Shillings")
6. print("Profits      : ", profits,"Uganda Shillings")
```

A Program to calculate the profits an item in Uganda shillings++

- Please enter the buying price
- 30000
- Please enter the selling price
- 45000
- Buying price : 30000 Uganda Shillings
- Selling price: 45000 Uganda Shillings
- Profits : 15000 Uganda Shillings

OPERATORS

➢ **Comparison operators**: These operators are used to compare two values and return a Boolean value (True or False) depending on the result of the comparison.

```
1. x = 1
2. y = 2
3. print(x == y)  # False
4. print(x != y)  # True
5. print(x < y)  # True
6. print(x > y)  # False
7. print(x <= y)  # True
8. print(x >= y)  # False
```

OPERATORS+

- **Logical operators**: These operators are used to combine two or more Boolean values and return a single Boolean value.

```
1. x = 1
2. y = 2
3. print(x > 0 and y > 0)
```

```
Output: True
```

```
1. print(x > 0 or y > 0)
```

```
Output: True
```

```
1. print(not x > 0)
```

```
Output: False
```

OPERATORS++

- How Logical 'and' and 'or' Operators Work

- Logical 'and': Let's consider the scenario where Odeke is going to the market, and I want him to buy some fruits for me. I specialize in making cocktail juice, which requires at least two different types of fruits. So, I instruct Odeke that if he goes to the market and finds both oranges and banana, he should buy them for me; otherwise, he should simply return with my money. In this case, Odeke won't make a purchase unless both of the specified fruits are available in the market. This analogy can be represented in code as follows:

OPERATORS+++

```
1. if banana == True and orange == True:
2.     buy both
3. else:
4.     Bring back my money
```

- In this code, we use the logical 'and' operator (and) to ensure that both conditions (banana being true and orange being true) must be met for the 'buy both' action to be taken. Otherwise, Odeke will return with the money if either condition is not satisfied

OPERATORS ++++

- Now, let's take a look at the logical 'or': I have been invited to South Korea for a conference that begins in two days, and I am supposed to travel as soon as possible. Usually, I prefer to travel in business class because of the comfort it offers. However, due to the limited time available, I might have to settle for an economy class ticket. So, I called my travel agent to book a business class ticket if available but also to book an economy class ticket if there's no business class option. In this situation, the priority is for a business class ticket, but if it's not available, then an economy class ticket will suffice.

OPERATORS ++++

- This program can be written as:
- if ticket == 'business class' or ticket == 'economy class':
- book()
- Here, we use the logical 'or' (or) to check if the 'ticket' variable is either 'business class' or 'economy class' and then take appropriate action

OPERATORS ++++

Operator	Description
and	The condition will also be true if the expression is true. If the two expressions a and b are the same, then a and b must both be true.
or	The condition will be true if one of the phrases is true. If a and b are the two expressions, then an or b must be true if and is true and b is false.
not	If an expression a is true, then not (a) will be false and vice versa.

Figure 2: Logical Operators summary [4]

OPERATORS AND EXPRESSIONS

- **Assignment operators:** These operators are used to assign values to variables.

```
1. x = 4
2. y = 2
3. x += y  # x is now equal to 6
4. x -= y  # x is now equal to 2
```

- **Identity operators**: These operators are used to compare two objects and return a Boolean value (True or False) depending on whether they are the same object. The ‘is’ key is used as an identity operator

```
1. x = 2
2. y = 3
3. print(x is y)  # False
4. print(x is not y)  # True
```

OPERATORS AND EXPRESSIONS

- **Membership operators**: These operators are used to check whether a value is contained in a sequence or set.

```
1.  list1 = [1, 2, 3, 4, 5]
2.  print(1 in list1)  # True
3.  print(6 in list1)  # False
```

➢ **Bitwise operators**: These operators are used to perform operations on bits, which are the smallest units of data in a computer.

```
1. x = 1
2. y = 2
3. print(x | y)  # 3
4. print(x & y)  # 0
5. print(x ^ y)  # 3
6. print(~x)  # -2
```

A program to calculate the body mass index (BMI)

➢ Algorithm

Step 1: Take the weight of the user

Step 2: Take the user height

Step 3: Calculate the BMI

Set 4: Display the results

1. w=eval(input("Please enter your weight"))
2. h=eval(input("Please enter your height"))

A program to calculate the body mass index (BMI)+

3. bmi=w/(h*h)
4. print("Your body mass index is ", bmi)

Output:

Please enter your weight 84

Please enter your height 2

Your body mass index is 21.0

Explanation: we took in user inputs and then compute the BMI by assigning the BMI to results of weight divide by product of height multiplied by height.

Operator Precedence and Associativity

- Operator Precedence refers to the order in which the interpreter evaluates an operator in an expression.
- Consider the example 45+2*3 this expression evaluates to 51. the question is, how did we arrived at that?
- As you will see later in the operators precedence table, the * operator have a higher priority than the + operator and therefore the statement first evaluated 2*3 to get 6 and later added it to 45 +6
- In python each of the operators have their own priority levels. This means that operations are performed according to the operators priority
- To easily remember the priority order, we used the acronym PEMDAS(Parenthesis, Exponentiation, Multiplication, Division, Addition and Subtraction)

Operator Precedence and Associativity

- Left to right: in this evaluation it is done from left to right
- 3+4+8-1=14
- Right to left: here the evaluation starts from the right hand side and ends at the left hand
- 3**3**2. The evaluation of these expressions begins with 3**2, which results in 9. This result is then used in the remaining part of the statement, which now becomes 3**9, resulting in 19683 as the final result.

Operator Precedence and Associativity+

- Now let us look at the operator precedence table to understand the order of priority of each operator.

Precedence	Operators	Description	Associativity
1	()	Parentheses	Left to right
2	**	Exponentiation	Right to left
3	+x, -x, ~x	Positive, negative, bitwise NOT	Right to left
4	*, @, /, //, %	Multiplication, matrix, division, floor division, remainder/modulo	Left to right
5	+, –	Addition and subtraction	Left to right
6	<<, >>	Shifts	Left to right
7	&	Bitwise AND	Left to right
8	^	Bitwise XOR	Left to right
9	|	Bitwise OR	Left to right
10	in, not in, is, is not, <, <=, >, >=, !=, ==	Comparisons, membership tests, identity tests	Left to Right
11	not x	Boolean NOT	Right to left
12	and	Boolean AND	Left to right
13	or	Boolean OR	Left to right

Operator Precedence and Associativity++

- According to the table, the operators at the top are the operators with the highest precedence and the priority keeps dropping as you move towards the bottom of the table
- When operations are at same priority level, then associativity comes to play.
- Associativity is that rule that determines how operators with the same precedence are evaluated within an expression and there are left and right associativity.

Operator Precedence and Associativity+++

- The associativity dictates that priority is given to the left most operator as can be seen in the expression below
- a=34*3/2%6
- print(a) #
- Consider this example: 25+4-2, to evaluate this we start by adding 25 to 4 which comes to 29 and later subtract 2 to get a final result of 27.

TRANSLATING MATHEMATICAL FORMULAE INTO EQUIVALENT PYTHON EXPRESSIONS

❖ To translate a mathematical formula/equation into an equivalent Python expression, you can follow these steps:

- ➢ Identify all of the mathematical operators and functions in the formula.
- ➢ Replace each mathematical operator with its Python equivalent.
- ➢ Replace each mathematical function with its Python equivalent.

TRANSLATING MATHEMATICAL FORMULAE INTO EQUIVALENT PYTHON EXPRESSIONS

- Add parentheses to the Python expression as needed to ensure that the order of operations is correct.

- Here are some common mathematical operators and functions, along with their Python equivalents:

- | Mathematical operator | Python equivalent | ^ | ** | | sin(x) | np.sin(x) | | cos(x) | np.cos(x) | | tan(x) | np.tan(x) | | log(x) | np.log(x) | | exp(x) | np.exp(x) |

- For example, to translate the mathematical formula/equation sin(x) + cos(x) into an equivalent Python expression, we would replace the sin() and cos() functions with their Python equivalents, and we would add parentheses to ensure that the order of operations is correct:

- Python
- np.sin(x) + np.cos(x)

TRANSLATING MATHEMATICAL FORMULAE INTO EQUIVALENT PYTHON EXPRESSIONS

- This Python expression will evaluate to the same value as the mathematical formula/equation sin(x) + cos(x) for all values of x.

- Here is another example:

- Mathematical formula/equation: sqrt(x^2 + y^2)
- Python equivalent: np.sqrt(x**2 + y**2)

- This Python expression will evaluate to the same value as the mathematical formula/equation sqrt(x^2 + y^2) for all values of x and y.

THE COMPOUND ASSIGNMENT OPERATOR

- **Compound assignment** operators are operators that combines the assignment operator and an other operator
- Among the computer assignment operators we have
- **Addition assignment**. Assigns the sum of the left operand and the right operand to the left operand
- **Example**

```
1. a=5
2. a+=5 #This line can also be write as a=a+5 which is a=5+5 and therefore a=10
3. print(a)
```

- Output: 10

THE COMPOUND ASSIGNMENT OPERATOR

Subtraction assignment. Assigns the difference of the left operand and the right operand to the left operand.

b=2

b-=1 #b=b-1 which is b=2-1, b=1

print(b)

Output: 1

Multiplication assignment. Assigns the product of the left operand and the right operand to the left operand

c=3

c*=c #c=c*c which is c=3*3 where c=9

print(c)

Output: 9

THE COMPOUND ASSIGNMENT OPERATORS

Division assignment. Assigns the quotient of the left operand and the right operand to the left operand

1. e=7
2. e/=2
3. print(e)

Output: 3.5

Floor division assignment. Assigns the result of dividing the left operand by the right operand, rounded down to the nearest integer, to the left operand.

1. f=9
2. f//=2
3. Print(f)

Output: 4

THE COMPOUND ASSIGNMENT OPERATOR

- **Modulus assignment**. Assigns the remainder of the left operand divided by the right operand to the left operand.

```
1. g=11
2. g%=5
3. print(g)
```

Output: 1

- **Exponentiation assignment**. Assigns the left operand raised to the power of the right operand to the left operand.

```
1. h=9
2. h**=3 #this line is the same wih h=h**3 which is 9**3
3. print(h)
```

Program for squaring any number

Algorithm

- Get an integer input from the user
- Compute the square root
- Display the results on the console

```
1. sr=eval(input("Please enter a number\n"))
2. sr**=2 #compute the square of the given number
3. print("The Square root of your number is ", sr)
```

Output:

Please enter a number

8

The Square root of your number is 64

CHANGING PRECEDENCE AND ASSOCIATIVITY OF ARITHMETIC OPERATORS

- The order of precedence and associativity can be changed by the use of parenthesis (), the parenthesis is an operator which has the highest priority.
- **Example1: Unaltered order of precedence**

```
1. a=1+4+6**2*2
2. print(a)
```

- Output: 77

Explanation

CHANGING PRECEDENCE AND ASSOCIATIVITY OF ARITHMETIC OPERATORS

- You may recall that in our operators' table, exponentiation has the second-highest priority, followed by the positive/negative bitwise operators and the arithmetic operators, in that order. Therefore, the computation in the given expression will begin with:
- 6 ** 2 = 36

CHANGING PRECEDENCE AND ASSOCIATIVITY OF ARITHMETIC OPERATORS

Our new expression will now be 1 + 4 + 36 * 2, again following the priority order, let us now procced by adding the sum of 1 + 4 to the products of 36 * 2. This will give us 77, which becomes our final result.

Example 2: Altered Precedence

1. b=1+4+6**(2*2)
2. print(b)

➢ Output: 1301

CHANGING PRECEDENCE AND ASSOCIATIVITY OF ARITHMETIC OPERATORS

➢ Explanation

- In our first example, we began with exponentiation; however, with the introduction of brackets into our expression, the priority changed.

➢ We had to start by dealing with the brackets, followed by exponentiation, and then addition. So, after addressing the brackets, we end up with 1 + 4 + 6 ** 4, which then becomes 1 + 4 + 1296. Finally, we sum it up to equal 1301, which is our final output

Bitwise Operators

- Bitwise operators are operators that perform bit-level operations on integers
- **Bitwise AND (&)**: The bitwise AND operator takes two integers and performs a bitwise AND operation on each pair of corresponding bits. It returns 1 if both bits are 1, otherwise 0.
- Example: a = 5 # Binary: 101 b = 3 # Binary: 011 result = a & b # Binary: 001 (Decimal: 1)
- **Bitwise OR (|)**: The bitwise OR operator takes two integers and performs a bitwise OR operation on each pair of corresponding bits. It returns 1 if at least one of the bits is 1.
- Example: a = 5 # Binary: 101 b = 3 # Binary: 011 result = a | b # Binary: 111 (Decimal: 7)

Bitwise Operators+

- **Bitwise XOR (^)**: The bitwise XOR (exclusive OR) operator takes two integers and performs a bitwise XOR operation on each pair of corresponding bits. It returns 1 if the bits are different, otherwise 0.
- Example: a = 5 # Binary: 101 b = 3 # Binary: 011 result = a ^ b # Binary: 110 (Decimal: 6)
- **Bitwise NOT (~)**: The bitwise NOT operator (unary operator) takes a single integer and inverts all the bits. It changes 1s to 0s and 0s to 1s.
- Example: a = 5 # Binary: 101 results = ~a # Binary: 11111010 (Decimal: -6)

Bitwise Operators++

- **Bitwise Left Shift (<<)**: The bitwise left shift operator shifts the bits of an integer to the left by a specified number of positions. It effectively multiplies the number by 2 to the power of the shift amount.
- Example: a = 5 # Binary: 101
- results = a << 2 # Binary: 10100 (Decimal: 20)
- **Bitwise Right Shift (>>)**: The bitwise right shift operator shifts the bits of an integer to the right by a specified number of positions. It effectively divides the number by 2 to the power of the shift amount.
- Example: a = 20 # Binary: 10100
- result = a >> 2 # Binary: 00101 (Decimal: 5)

Summary

- In this lecture, we explored the fundamental concepts of operators and expressions in the Python programming language. Operators and expressions are at the core of any programming language, enabling us to manipulate data and perform computations.
- **Operators Overview**
- Operators are special symbols in Python used to perform various operations on data.

- Arithmetic operators are used for basic mathematical calculations, such as addition (+), subtraction (-), multiplication (*), division (/), and modulus (%).

Summary+

- Python follows the standard order of operations (PEMDAS) for evaluating arithmetic expressions.
- Comparison Operators, these operators are used to compare values and return Boolean results, such as equality (==), inequality (!=), greater than (>), less than (<), etc.
- Logical operators are used to perform logical operations on Boolean values, including AND (and), OR (or), and NOT (not).

Summary++

- Assignment operators are used to assign values to variables. Common ones include =, +=, -=, *=, /=, and %=.
- Bitwise Operators: Bitwise operators manipulate individual bits of integer values. Examples include bitwise AND (&), bitwise OR (|), bitwise XOR (^), left shift (<<), and right shift (>>).
- Expressions, these are combinations of operators and operands that produce a result. They can be as simple as a single value or involve complex computations.

Summary+++

- Operator Precedence: Operator precedence determines the order in which operators are evaluated in an expression. Python follows a specific precedence hierarchy, but parentheses can be used to override it.
- Operator Associativity: Operator associativity defines the order of evaluation when operators have the same precedence. Most operators in Python are left-associative.
- Operands and Data Types: Operands are the values that operators act upon. Python supports various data types as operands, including integers, floats, strings, and more.

Summary++++

- String Concatenation, the + operator can also be used for string concatenation, allowing you to combine strings together.
- Type Conversion (Typecasting): You can convert data from one type to another using typecasting functions like int(), float(), and str().
- Expression Evaluation: Python evaluates expressions based on operator precedence and associativity, producing a final result that can be assigned to variables or used in program logic.

Summary++++

- By understanding and mastering operators and expressions in Python, you have gained a solid foundation for building more complex programs and solving a wide range of computational problems. These concepts are essential for any Python programmer, and they will continue to be integral as you progress in your coding journey.

Reference

[1]Introduction Programming and problem solving with python, Ashok N. Kamthane, Amit A. Kamthane, McGraw Hill Education (India) Private Limited,2018, page 56

[2]Introduction to computing and problem-solving using python, E Balagurusamy, McGraw Hill Education (India) Private Limited, 2016, page 51

[3] GeeksforGeeks. (2023). Precedence and associativity of operators in Python. *GeeksforGeeks*. https://www.geeksforgeeks.org/precedence-and-associativity-of-operators-in-python/

[4] (n.d.). *Python operators - javatpoint*. Www.Javatpoint.com. Retrieved September 30, 2023, from https://www.javatpoint.com/python-operators