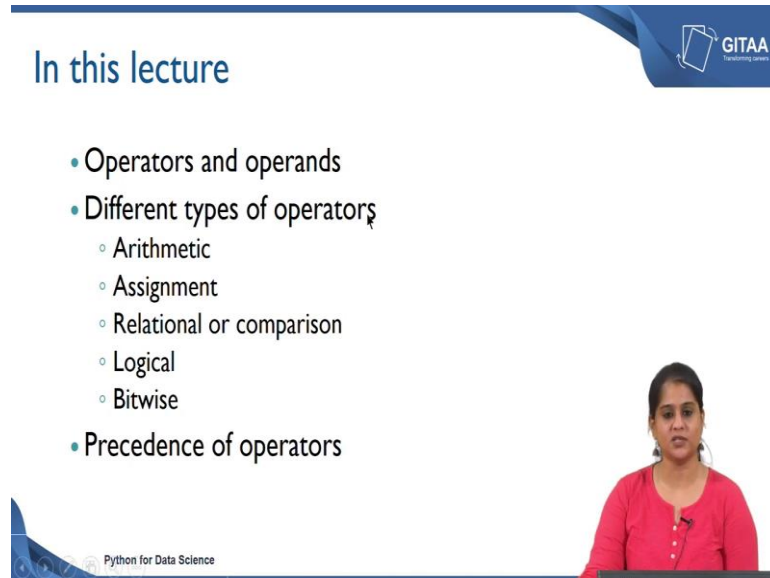


Python for Data Science
Prof. Ragnathan Rengasamy
Department of Computer Science and Engineering
Indian Institute of Technology, Madras

Lecture – 06
Operators

(Refer Slide Time: 00:17)



The slide features a blue header with the text "In this lecture" and a logo for "GITAA Traditional Learning". Below the header is a bulleted list of topics. In the bottom right corner, there is a video inset showing a woman in a red top. The bottom left corner of the slide contains the text "Python for Data Science".

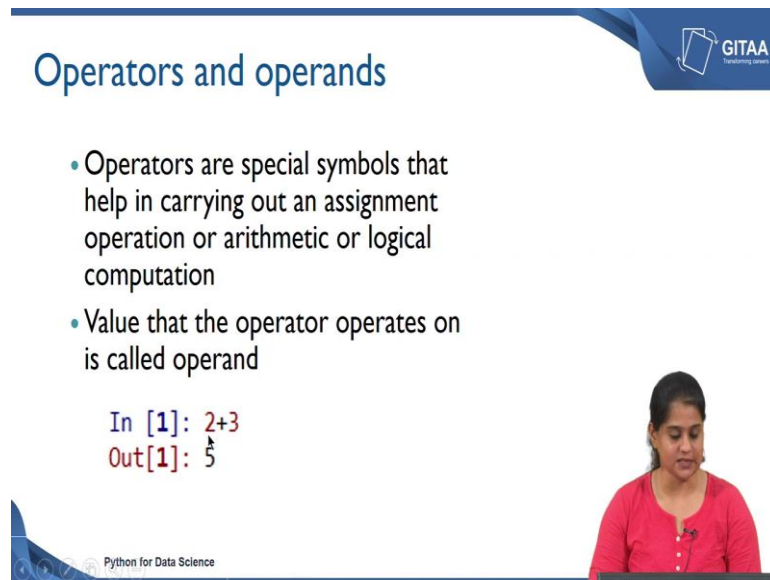
In this lecture

- Operators and operands
- Different types of operators
 - Arithmetic
 - Assignment
 - Relational or comparison
 - Logical
 - Bitwise
- Precedence of operators

Python for Data Science

Welcome, to the lecture on Operators! In this lecture, we are going to see what an operator is and what an operand is. We will also look at the different types of operators that is used in python and these are arithmetic, assignment, relational, logical and bitwise. We will also be looking at the precedence of operators and how to use them in an expression.

(Refer Slide Time: 00:37)



The slide features a blue header with the title "Operators and operands" and the GITAA logo. Below the title, there are two bullet points: "Operators are special symbols that help in carrying out an assignment operation or arithmetic or logical computation" and "Value that the operator operates on is called operand". A code example shows "In [1]: 2+3" and "Out[1]: 5". A woman in a red shirt is visible in the bottom right corner of the slide frame. The footer includes the text "Python for Data Science".

Operators and operands

- Operators are special symbols that help in carrying out an assignment operation or arithmetic or logical computation
- Value that the operator operates on is called operand

```
In [1]: 2+3  
Out[1]: 5
```

Python for Data Science

So, let us see what operators and operands are. An operator is a special symbol that will help you carrying out an assignment operation or some kind of a computation the nature of the computation can be arithmetic or logical.

Now, the value that the operator operates on is called an operand. So, let us take a small example here to illustrate what an operator and operand is. In this case, the plus symbol that you see in the plus sign that is an operator. This operator denotes addition. So, you see two numbers before and after the operator. So, 2 and 3 in this case are called operands.


(Refer Slide Time: 01:19)

Arithmetic operators

- Used to perform mathematical operations between two operands
- Create two variable a and b with values 10 and 5 respectively

a=10 b=5

Symbol	Operation	Example
+	Addition	In [3]: a+b Out[3]: 15
-	Subtraction	In [4]: a-b Out[4]: 5



Python for Data Science


So, let us look at arithmetic operators. Now, arithmetic operators are used to perform mathematical operations between any two operands. So, let us take an example to illustrate the use of arithmetic operators. Create two variables a and b with values 10 and 5. In the previous lectures we have already seen how to create a variable. So, we were going to use the same procedure here.

So, the first operator that we are going to look at is the addition operator. It is denoted by a plus symbol and this is how an addition operation is carried out. If I want to add two variables, then I am just going to separate the variables with the plus symbol in this case a is 10, b is 5 so, the result in sum that you get is 15. The next operation is subtraction denoted by a hyphen. I have the corresponding example here. So, in this case I am going to subtract a and b and the output that I get here is 5.

(Refer Slide Time: 02:17)

Arithmetic operators

Symbol	Operation	Example
*	Multiplication	In [5]: a*b Out[5]: 50
/	Division	In [6]: a/b Out[6]: 2.0
%	Remainder	In [8]: a%b Out[8]: 0
**	Exponent	In [7]: a**b Out[7]: 100000



Python for Data Science

So, the next arithmetic operation that we are going to look at is the multiplication operation, it is denoted by an asterisk and if you want to multiply two variables separate the variable and insert an asterisk symbol. The product that you get in this example is 50; a is 10, b is 5 again. Now, 10 into 5 is 50 and that is the output that you get .

The next operation is division that is denoted by a forward slash. So, you separate the variables and insert a forward slash between them, what you get is basically the quotient. So, in this case 10 by 5 gives you a quotient of 2 which is the output here. So, the next operation is getting a remainder and it is denoted by the percentage symbol. So, let us take the same example here. I am trying to get the remainder when I divide a and b and you just separate the variables and insert the percentage symbol and that returns the remainder. In this case a is 10 and b is 5. So, 10 is divisible by 5 and hence I get a remainder of 0.

The next operation is exponent and it is denoted by double asterisk. So, let us say if I want to raise a variable to the power of another variable then I am going to use this operation. So, let us say in this case I want to raise a to the power of b, then I just say a double asterisk b. So, in this case since we have a as 10 and b as 5, I am just going to raise 10 to the power of 5 and the corresponding output that you get here is 1 lakh.


(Refer Slide Time: 03:51)

Hierarchy of arithmetic operators

Decreasing order of precedence	Operation
Parentheses	()
Exponent	**
Division	/
Multiplication	*
Addition and subtraction	+,-

$$A = 7 - 2 \times \frac{27}{3^2} + 4$$

```
In [10]: A=7-2*(27/3**2)+4
In [11]: print(A)
5.0
```



Python for Data Science

So, let us look at the hierarchy of operators. Now, I have ordered the operators in the decreasing order of precedence. So, the first is parentheses. So, parentheses is not really an operator, but anything that is enclosed within parentheses gets the topmost priority. So, therefore, I have included parenthesis also as an operation. Now, this is followed by exponential operations and then division, multiplication and addition and subtraction are given the same precedence. So, let us take an example here in this case.

I have the following expression. I have

$$A = 7 - 2 * \frac{27}{3^2} + 4$$

So, to avoid confusion I am going to add bracket for this 27 by 3 square term. So, 27 is the numerator, the denominator is 3 square and to denote 3 square I add a double asterisk here and the entire term I am enclosing it within parentheses. So, once you execute the command in your console, if you print the value of A, it should return 5. So, this is how you will use the operators in an expression. You can also try out this example.

(Refer Slide Time: 05:03)

Assignment operators

- Used to assign values to variables

Symbol	Operation	Example
=	Assign values from right side operands to left side operand	<code>a=10</code> <code>b=5</code>
+=	Adds right operand to left operand and stores result on left side operand (<code>a=a+b</code>)	In [55]: <code>a+=b</code> ...: <code>print(a)</code> 15
-=	Subtracts right operand from left operand and stores result on left side operand (<code>a=a-b</code>)	In [57]: <code>a-=b</code> ...: <code>print(a)</code> 5

Python for Data Science

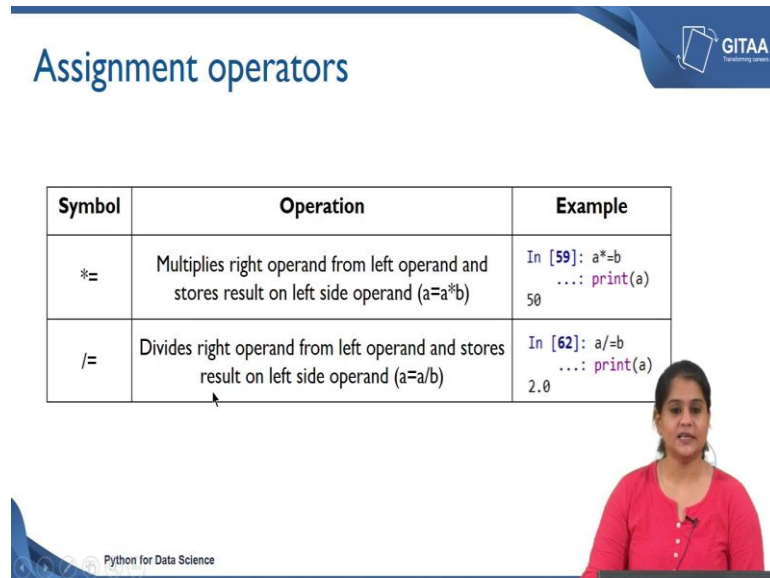
So, now let us look at assignment operators. So, an assignment operator is used to assign a value to a variable. So, the first assignment operator that we are going to look into is the equal to symbol. Now, this is the most commonly used assignment operation and that is because whenever you want to create a variable you want to assign a value to it. So, we have learnt how to use this operator in our earlier lectures.

So, what it basically does is that it will assign values from the right side operand to the left side operand. Now, the left side operand is a variable name and the right side operand is the value that is given to the variable. So, let us take an example in this case. I am retaining the same variable names a and b, I am assigning a value of 10 to a. In this case 10 is my right side operand and a is my left side operand. So, the same definition also holds good for b equal to 5.

So, the next operator that we are going to look into is the += operator. So, what it basically does is that, it first adds the right operand to the left operand and then it will store the result on the left side operand. Now, let us take the same variables a and b. Now, if I were to denote `a += b` then this would translate to `a = a + b`. I have indicated that within parentheses here. So, whenever I give `a += b`, then I am saying `a = a + b`, so, the first operation that happens is the addition operation which is `a + b`. So, the value of `a + b` gets stored in a and hence you can see that the value of a gets updated to 15 a was earlier 10, and now the value gets updated to 15.

The next operator is the minus equal to operator. So, this is also similar to the addition operator that we earlier saw. It basically subtracts the right operand from the left and it will store the result on the left side operand. Now, whenever I give $a -= b$ then it translates to $a = a - b$. Now, whenever I compute the difference in this case I am getting a difference of 5 and that is what I am printing.

(Refer Slide Time: 07:19)



The slide features a blue header with the title "Assignment operators" and the GITAA logo. Below the header is a table with three columns: Symbol, Operation, and Example. The first row describes the asterisk assignment operator (*=), and the second row describes the forward slash assignment operator (/=). A presenter in a red shirt is visible in the bottom right corner of the slide.

Symbol	Operation	Example
<code>*=</code>	Multiplies right operand from left operand and stores result on left side operand ($a=a*b$)	In [59]: <code>a*=b</code> ...: <code>print(a)</code> 50
<code>/=</code>	Divides right operand from left operand and stores result on left side operand ($a=a/b$)	In [62]: <code>a/=b</code> ...: <code>print(a)</code> 2.0

So, asterisk operator will multiply the right operand from the left and will store the result on the left operand. Now, in this case again I am going to retain the same values of a and b ; a is 10, b is 5 which means I am multiplying a and b first. So, $a * b$ will give me a product of 50. Now, whenever I print the value of a it will give me the updated value which is 50. Forward slash equal to means division. So, whenever I use this operator, I am going to divide the right operand from the left and store the value on the left operand.

Now, in this case $a /= b$ translates to $a = a / b$. Now, if you print the value of a you can see that the value of two has been updated 2 to from 10.

(Refer Slide Time: 08:09)

Relational or comparison operators

- Tests numerical equalities and inequalities between two operands and returns a boolean value
- All operators have same precedence
- Create two variables x and y with values 5 and 7 respectively

```
In [1]: x = 5
In [2]: y = 7
```

Symbol	Operation	Example
<	Strictly less than	In [3]: print(x<y) True
<=	Less than equal to	In [4]: print(x<=y) True

Python for Data Science 17

So, now let us see what relational or comparison operators are. A relational operator will test for a numerical equality or an inequality between two operands. The value that a relational operator returns is Boolean in nature which means it will basically return true or false. Now, all the relational operators that we are going to look into have the same precedence which means they all have the same priority.


So, let us create two variables x and y. I am assigning a value of 5 to x and 7 to y. The first relational operation is the strictly less than operation. It is denoted by the angle operator with its tip towards the left. So, let us take an example and see how this operator works. Now, we already have the values for x and y. Now, I am just giving the relation $x < y$. So, now, what will happen is that it will check whether x is strictly less than y? In our case, yes, x is 5, y is 7. So, yes, 5 is strictly less than 7 and hence the output that you will see is true.

The next operation is the less than equal to operation. It is denoted again with the angled operator with its tip towards the left followed by an equal to symbol. Now, let us take an example. Now, in this case I am trying to print the output for $x \leq y$. So, in this case we are checking if $x < y$ or is $x = y$. So, these are the two conditions that we are going to check. So, x is of course, not equal to y, but x is less than y. So, that condition is satisfied and hence the output that you get is true.

(Refer Slide Time: 09:49)

Relational or comparison operators

Symbol	Operation	Example
>	Strictly greater than	In [5]: <code>print(x>y)</code> False
>=	Greater than equal to	In [6]: <code>print(x>=y)</code> False
==	Equal to equal to	In [7]: <code>print(x==y)</code> False
!=	Not equal to	In [8]: <code>print(x!=y)</code> True



Python for Data Science

So, similar to the strictly less than and less than equal to operator, you have the strictly greater than and greater than equal to operator. So, contrary to what a strictly less than operator does, in this case you are strictly checking whether x is greater than y . Of course, it is not and hence the output is false. For greater than equal to you basically are checking whether $x > y$ or is $x = y$. So, both the conditions are false anyways and hence the output that you get is also false. So, a strictly greater than operator is denoted by an angled operator with its tip to the right and a greater than equal to operator is denoted by an angled operator with its tip to the right followed by an equal to symbol.

The next operation is the equal to equal to operation. It is denoted by a double equal to symbol and what we really check when we give double equal to? We check if the left hand side operand is it exactly equal to the right hand side operand. In this case the value of x and y are 5 and 7 respectively and I am checking if 5 is exactly equal to 7 or not. No, it is not and hence the output is false.

The next operation is not equal to. It is denoted by an exclamation followed by an equal to symbol(`!=`). So, in this case the output is going to be true as long as x is not equal to y . So, this operator is frequently used when you are iterating through a loop and you want to run the loop or you want to iterate through the loop as long as a certain condition is obeyed. So, you can use not equal to in that case. Now, as long as x is not equal to y my output is always going to remain as true.

(Refer Slide Time: 11:31)

Logical operators

- Used when operands are conditional statements and returns boolean value
- In python, logical operators are designed to work with scalars or boolean values

Symbol	Operation	Example
or	Logical OR	In [9] <code>print((x>y) or (x<y))</code> True
and	Logical AND	In [10]: <code>print((x>y) and (x<y))</code> False
not	Logical NOT	In [11]: <code>print(not (x==y))</code> True

Python for Data Science 23

So, the next set of operators are the logical operators. A logical operator is used when the operands are conditional statements. The output for logical operators are Boolean in nature which means they return true or false. So, strictly from the point of view of python logical operators are designed to work only with scalars and Boolean values. So, if you want to compare two arrays, then a logical operator cannot be used. So, let us take the first logical operation which is logical or it is denoted by the letters or, both letters in lowercase.

Now, let us take a small example here, I am retaining the same values for x and y; x is 5, y is 7. Now, if I give $(x > y)$ or $(x < y)$, I get an output that says true. Now, why does it happen? So, a logical OR is designed to give an output true when one of the statement is satisfied. In this case $x > y$ it is not satisfied. So, it is a false, but however, $x < y$ that statement is satisfied. So, this gives you an output which is true. So, the inputs to the or operator is basically false and true. So, whenever you have a false and a true operand, then the resultant is always true. So, hence you are also getting an output which is true.

The next is the logical AND which is represented by the letters and, all letters again in lower case. Let us take a small example here, I am taking the same expressions that I have considered above. Now, in this case instead of or I am replacing them by and. So, for the same conditions I am getting a different output and in this case it is false. So, why does this happen? If you look at the conditional statements the first is $x > y$, we know

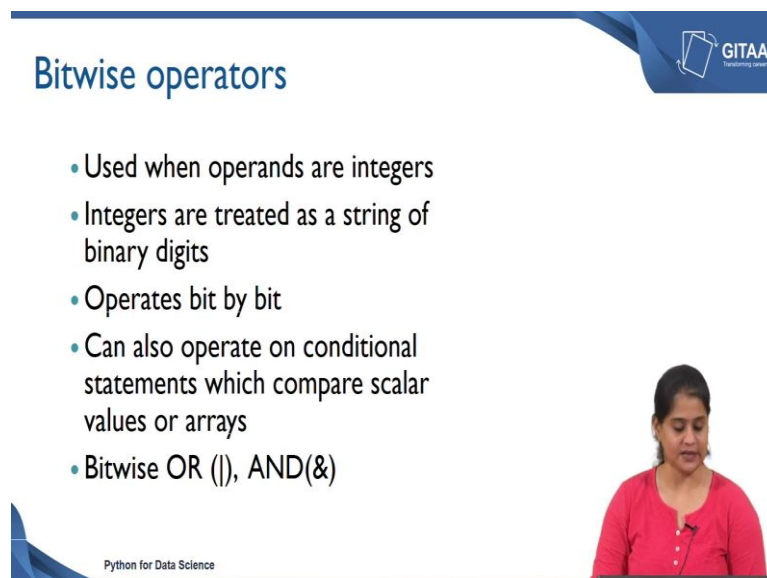
that this is a false statement it gives you a Boolean value of false. The second conditional statement is $x < y$. Now, this is true like I earlier said.

So, the way a logical AND works is that whenever you have a false and a true condition as the operands you will basically get a false output and this is because logical AND expects you to satisfy both the conditions and unless both these values are true it will never return the output as true. So, even when you have false true or true false the output is always false.

So, the next logical operator is not represented by the letter not, again in lowercase. So, not basically negates your statement. So, I have taken the example of $x == y$. Now, we know the value of x is 5, value of y is 7 of course, both of them are not equal. Now, the output that you will get from this conditional statement is false. So, we are trying to negate false which means not false. So, that gives you a result which is true. So, that is why you get the output as true.

Now, another important point to note in logical operators is that, whenever you giving these conditional or relational statements, make sure that you enclose them within parentheses because if you are not going to do it then you are likely to get an error.

(Refer Slide Time: 14:55)



The slide features a blue header with the text "Bitwise operators" and the GITAA logo. A list of five bullet points is displayed on the left side. In the bottom right corner, a woman in a red top is visible, likely the presenter. The footer of the slide reads "Python for Data Science".

- Used when operands are integers
- Integers are treated as a string of binary digits
- Operates bit by bit
- Can also operate on conditional statements which compare scalar values or arrays
- Bitwise OR (`|`), AND (`&`)

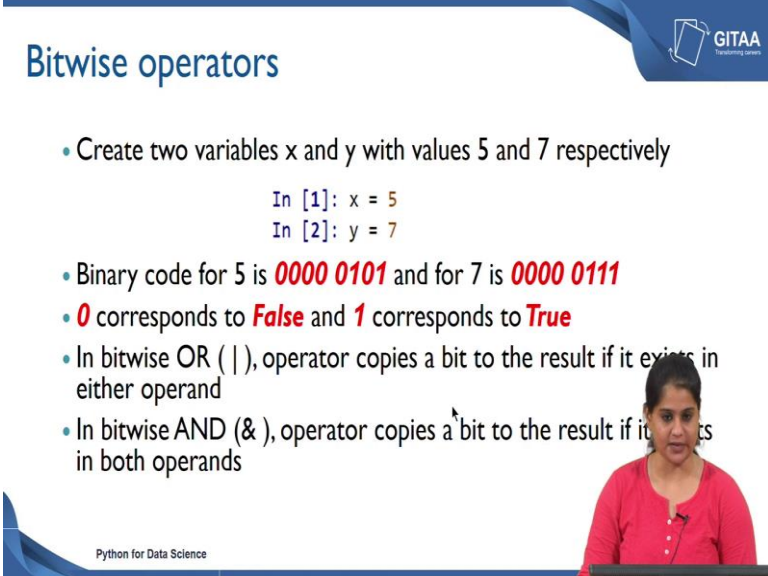
So, let us move on to bitwise operators. So, bitwise operators are used when the operands are integers. So, these integers are treated as a string of binary digits and are binary

encoded. So, when you are going to use a bitwise operator on two integers which are binary coded, the operator is going to compare bit by bit of the binary code and that is how the operator got its name bitwise.

The other advantage of using a bitwise operator is that, they can operate on conditional statements. Now, these conditional statements can compare scalar values or they can also compare arrays. Now, if you would like to compare arrays you would be using a bitwise operator. We earlier saw that we cannot use logical operators to handle arrays and this is where bitwise operators step in.

So, throughout the course we are going to be looking into two bitwise operators. The first says bitwise OR which is represented by a pipe and second operator is the bitwise AND represented by an ampersand.

(Refer Slide Time: 15:53)



Bitwise operators

- Create two variables x and y with values 5 and 7 respectively

```
In [1]: x = 5  
In [2]: y = 7
```

- Binary code for 5 is **0000 0101** and for 7 is **0000 0111**
- **0** corresponds to **False** and **1** corresponds to **True**
- In bitwise OR (|), operator copies a bit to the result if it exists in either operand
- In bitwise AND (&), operator copies a bit to the result if it exists in both operands

Python for Data Science

GITAA
Transforming Learning

(A video inset of a woman in a red shirt is visible in the bottom right corner of the slide.)

So, create two variables x and y with values 5 and 7. Now, these are the binary codes for 5 and 7 and we are going to be using these variables for our example. So, 0 corresponds to false and 1 corresponds to true. And, in a bitwise OR the operator will copy bit by bit of the result if it is there in either of the operands. But, in a bitwise AND the operator will copy the bit only if it exists across both the operands.

So, let us take an example and see what these two statements mean.

(Refer Slide Time: 16:23)

Bitwise OR on integers


Code and output in console

```
In [47]: x | y  
Out[47]: 7
```

Binary code for 5: 0 0 0 0 0 1 0 1

Binary code for 7: 0 0 0 0 0 1 1 1

Resultant binary code: 0 0 0 0 0 1 0 1



Python for Data Science

Now, I am going to be illustrating a bitwise OR on integers. Now, I am using the bitwise OR operator which is a pipe symbol between x and y; x and y are my operands. The output that you will be getting is 7. So, let us see how this output was achieved.

So, I have created two arrays here. The cells in these arrays consists of the individual binary code for 5 and 7, and I have color coded them for reference. So, the first two positions of both the binary codes is 0. So, both these serve as my input operands. Now, both these positions have 0 and hence the resultant will also contain 0.

(Refer Slide Time: 17:07)

Bitwise OR on integers

Binary code for 5: 0 0 0 0 0 1 0 1

Binary code for 7: 0 0 0 0 0 1 1 1

Resultant binary code: 0 0 0 0 0 1 0 1

- 0 present in positions 2-5, therefore resultant cell will also contain 0
- In the 6th position, 1 is present in both operands and hence resultant will also contain 1

Python for Data Science

27

Now, let us take the second position. The second position also has 0 for both the binary codes and hence my corresponding position in the resultant binary code is also going to be 0. I have highlighted the positions using circles to just show you which cells I am referring to.

So, now you can also see that positions 3, 4 and 5 consists of 0's for both the binary codes. So, hence the corresponding positions of the resultant binary code will also contain 0. Another important point to note is that the sixth position of both the binary codes consists of 1. So, the binary code for 5 and the binary code for 7, in both of these codes the sixth position corresponds to 1 and hence in the resultant binary code I am copying 1 for the sixth position.

(Refer Slide Time: 17:59)

Bitwise OR on integers

- The 7th position has 0 in the first operand and 1 in the second

Binary code for 5: 0 0 0 0 0 1 0 1
Binary code for 7: 0 0 0 0 0 1 1 1

- Since this is an OR operator, only the True condition is considered

0 0 0 0 0 1 1 1

Binary code for 5: 0 0 0 0 0 1 0 1
Binary code for 7: 0 0 0 0 0 1 1 1

0 0 0 0 0 1 1 1

Python for Data Science 28

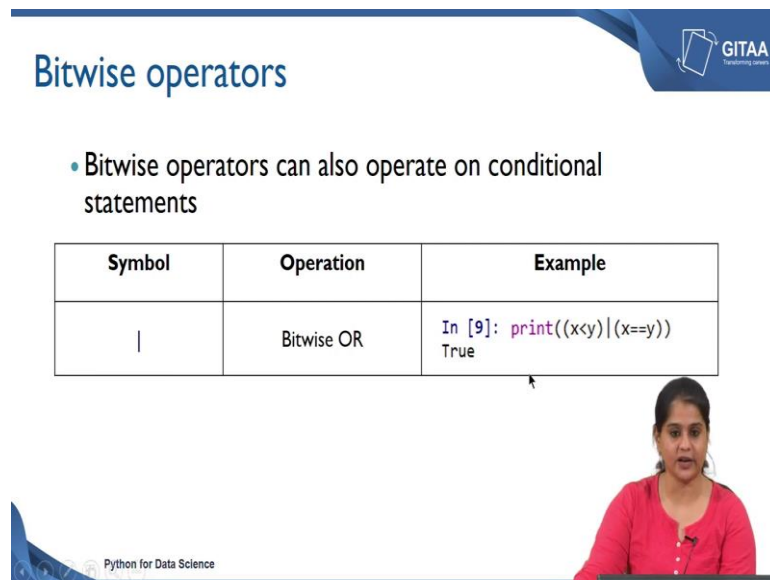
Moving further, if you compare the 7th position for both these binary codes you can see that for the binary code 5 the 7th position has 0 and for the binary code 7, the 7th position has 1. So, in this case there is a difference in values between both these operands. So, we are going to see how to fill in the corresponding position of the resultant binary code.

So, since we are using an OR operator, when one of the condition is true the resultant always becomes true. In this case, if you can recall so, like I earlier said 0 corresponds to false and 1 corresponds to true so, we have one true condition. So, the resultant will also

contain the true value which is 1. So, an OR operator will give you the output as true when one of the operands is true.

Now, we are left with the last position and for both these binary codes the last position is 1 and I am going to be copying this value to the corresponding position in my resultant binary code. So, this is the binary code that you get when you apply a bitwise OR between two integers. This is the binary code for 7 that we earlier started with and this is how a bitwise or operator works between two integers.

(Refer Slide Time: 19:17)



The slide is titled "Bitwise operators" and features a logo for "GITAA Traditional courses" in the top right corner. A bullet point states: "• Bitwise operators can also operate on conditional statements". Below this is a table with three columns: "Symbol", "Operation", and "Example". The table contains the following information:

Symbol	Operation	Example
	Bitwise OR	In [9]: <code>print((x<y) (x==y))</code> True

In the bottom right corner of the slide, there is a video inset showing a woman in a red shirt speaking. At the bottom left of the slide, the text "Python for Data Science" is visible.


We can also use bitwise operators for conditional statements. Now, if I were to use the bitwise OR for a relational statement, this is how it could look. So, I am giving two conditional statements here. The first is where I am checking if x is less than y; the second is where I am checking if x is equal to equal to y.

Now, I am in this case x is less than y that results in a value which is true and whereas, the second conditional statements which is x equal to equal to y will result in false. In this case, the first condition is true and hence the output that you get is also true.

(Refer Slide Time: 19:53)

Precedence of operators

Decreasing order of precedence	Operation
Parentheses	()
Exponent	**
Division	/
Multiplication	*
Addition and subtraction	+,-
Bitwise AND	&



Python for Data Science


So, now let us look at the precedence of operators, I have ordered the operators in the decreasing order of precedence. So, like I earlier mentioned parenthesis is not an operator. Now, any expression with operators that are enclosed within parentheses they get the topmost priority. So, that is why parentheses always occupy the first line in terms of precedence.

Now, after parentheses I have the exponential operation followed by division, multiplication, addition and subtraction are given the same precedence, I then follow it up with my bitwise AND bitwise OR, all relational operators are given the same precedence.

(Refer Slide Time: 20:31)

Precedence of operators

Decreasing order of precedence	Operation
Bitwise OR	
Relational/ comparison operators	==, !=, >, >=, <, <=
Logical NOT	not
Logical AND	and
Logical OR	or



Python for Data Science


GITAA
Teaching Assistants

And, then comes the logical NOT, logical AND, and logical OR. So, this is the decreasing order of precedence for all the operators put together.

(Refer Slide Time: 20:45)

Summary

- Important operators
 - Arithmetic
 - Assignment
 - Relational
 - Logical
 - Bitwise



Python for Data Science

GITAA
Teaching Assistants

So, to summarize in this lecture we saw what are the important operators. We looked at what arithmetic, assignment, relational, logical and bitwise operators do. We also took an example in each of these case to illustrate how the operator works and what is the nature of the output.

Thank you.