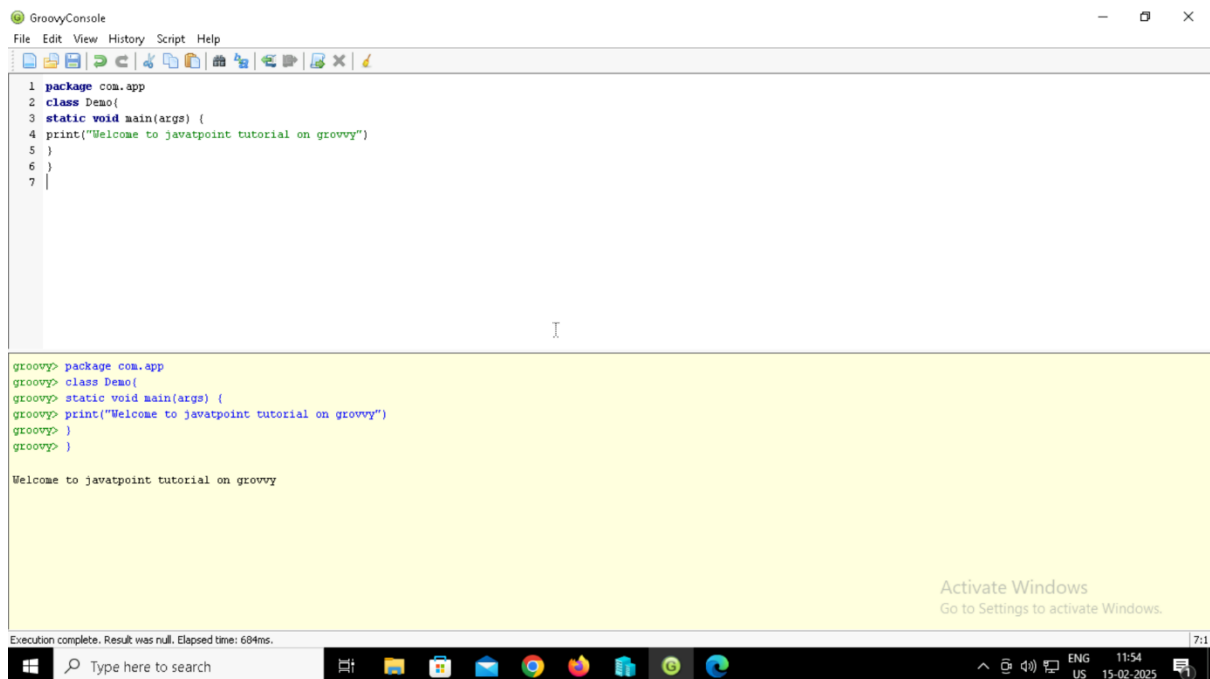


## 1) Print a line without using round brackets



The screenshot shows the GroovyConsole application window. The top menu bar includes File, Edit, View, History, Script, and Help. Below the menu is a toolbar with various icons. The main text area contains the following Groovy code:

```
1 package com.app
2 class Demo{
3     static void main(args) {
4         print("Welcome to javatpoint tutorial on groovy")
5     }
6 }
7 |
```

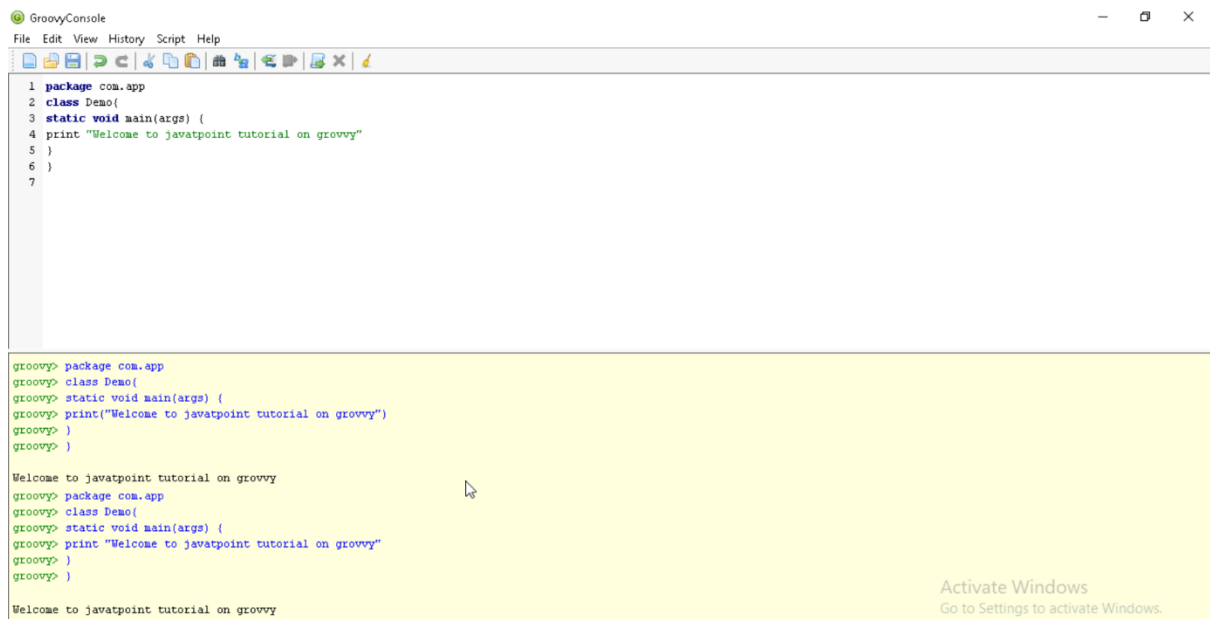
The output area below the code shows the execution results:

```
groovy> package com.app
groovy> class Demo{
groovy> static void main(args) {
groovy> print("Welcome to javatpoint tutorial on groovy")
groovy> }
groovy> }

Welcome to javatpoint tutorial on groovy
```

At the bottom, a status bar indicates "Execution complete. Result was null. Elapsed time: 684ms." and a Windows taskbar is visible at the very bottom.

## 2) Double quotes as well as single quotes can be used in a string.



The screenshot shows the GroovyConsole application window. The top menu bar includes File, Edit, View, History, Script, and Help. Below the menu is a toolbar with various icons. The main text area contains the following Groovy code:

```
1 package com.app
2 class Demo{
3     static void main(args) {
4         print "Welcome to javatpoint tutorial on groovy"
5     }
6 }
7
```

The output area below the code shows the execution results:

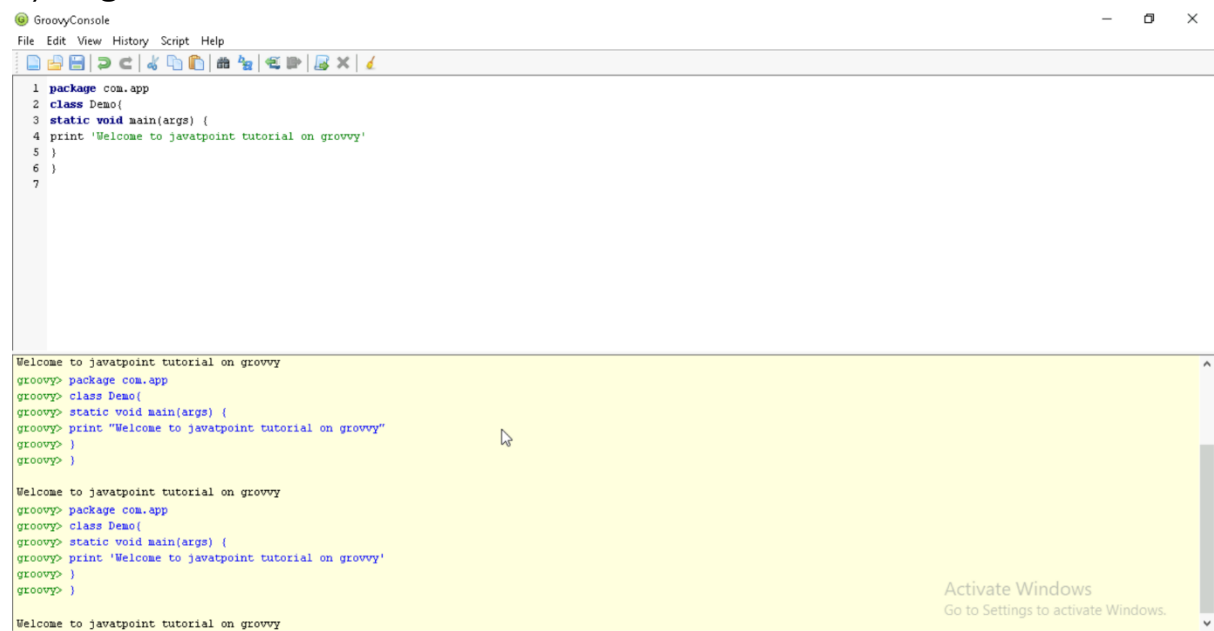
```
groovy> package com.app
groovy> class Demo{
groovy> static void main(args) {
groovy> print "Welcome to javatpoint tutorial on groovy"
groovy> }
groovy> }

Welcome to javatpoint tutorial on groovy
groovy> package com.app
groovy> class Demo{
groovy> static void main(args) {
groovy> print "Welcome to javatpoint tutorial on groovy"
groovy> }
groovy> }

Welcome to javatpoint tutorial on groovy
```

At the bottom, a status bar indicates "Execution complete. Result was null. Elapsed time: 684ms." and a Windows taskbar is visible at the very bottom.

### 3) Single line comment as well as a multi-line comment



The screenshot shows the GroovyConsole application. The script editor contains the following code:

```
1 package com.app
2 class Demo{
3     static void main(args) {
4         print 'Welcome to javatpoint tutorial on groovy'
5     }
6 }
7
```

The console output shows the result of running the script:

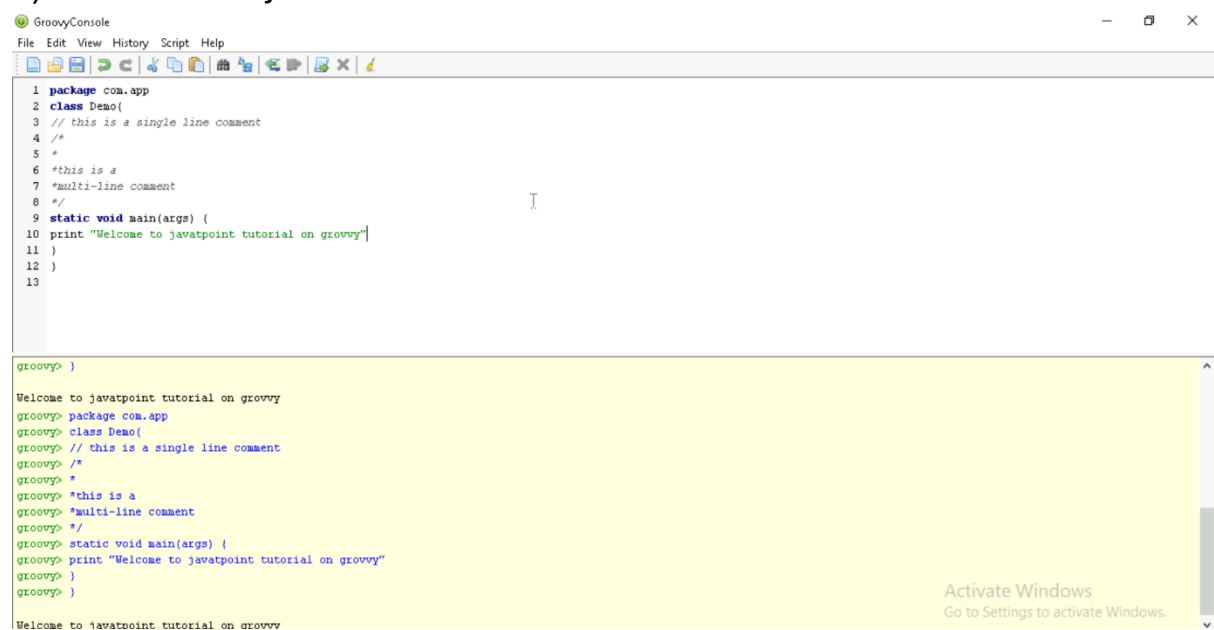
```
Welcome to javatpoint tutorial on groovy
groovy> package com.app
groovy> class Demo{
groovy> static void main(args) {
groovy> print "Welcome to javatpoint tutorial on groovy"
groovy> }
groovy> }

Welcome to javatpoint tutorial on groovy
groovy> package com.app
groovy> class Demo{
groovy> static void main(args) {
groovy> print 'Welcome to javatpoint tutorial on groovy'
groovy> }
groovy> }

Welcome to javatpoint tutorial on groovy
```

An "Activate Windows" watermark is visible in the bottom right corner of the console area.

### 4) Not necessary to have a class or the main function



The screenshot shows the GroovyConsole application. The script editor contains the following code:

```
1 package com.app
2 class Demo{
3     // this is a single line comment
4     /*
5     *
6     *this is a
7     *multi-line comment
8     */
9     static void main(args) {
10        print "Welcome to javatpoint tutorial on groovy"
11    }
12 }
13
```

The console output shows the result of running the script:

```
groovy> }

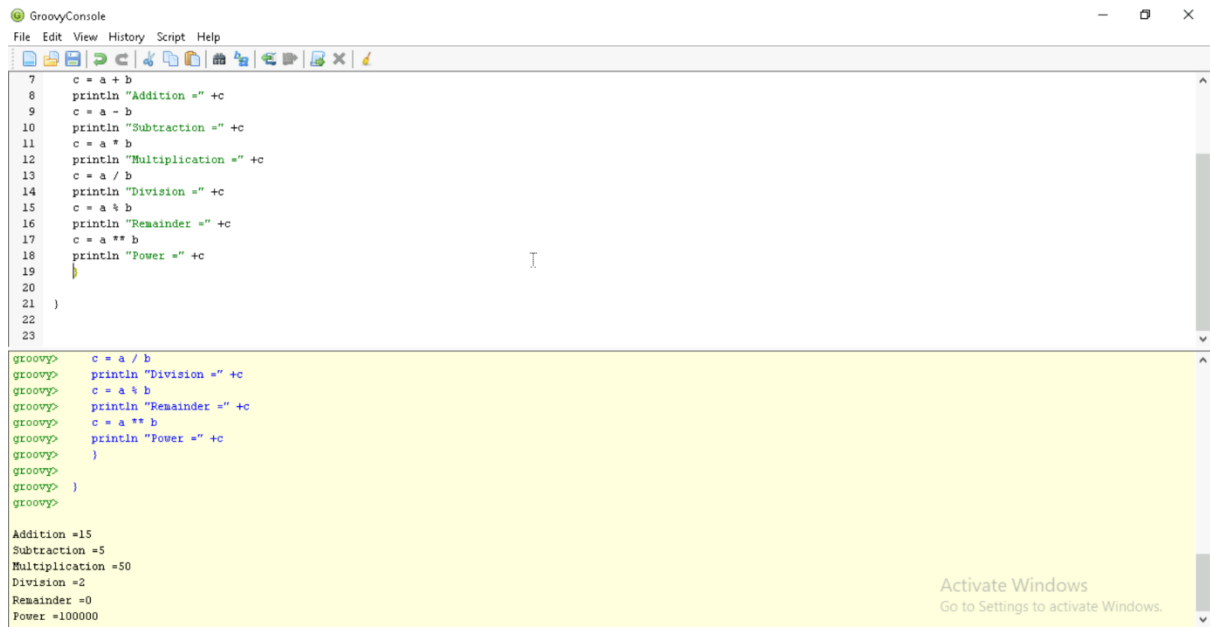
Welcome to javatpoint tutorial on groovy
groovy> package com.app
groovy> class Demo{
groovy> // this is a single line comment
groovy> /*
groovy> *
groovy> *this is a
groovy> *multi-line comment
groovy> */
groovy> static void main(args) {
groovy> print "Welcome to javatpoint tutorial on groovy"
groovy> }
groovy> }

Welcome to javatpoint tutorial on groovy
```

An "Activate Windows" watermark is visible in the bottom right corner of the console area.

# Operators in groovy

## 6) Arithmetic operator



The screenshot shows the GroovyConsole application. The top pane contains a Groovy script with the following code:

```
7  c = a + b
8  println "Addition =" + c
9  c = a - b
10 println "Subtraction =" + c
11 c = a * b
12 println "Multiplication =" + c
13 c = a / b
14 println "Division =" + c
15 c = a % b
16 println "Remainder =" + c
17 c = a ** b
18 println "Power =" + c
19 }
20
21 )
22
23
```

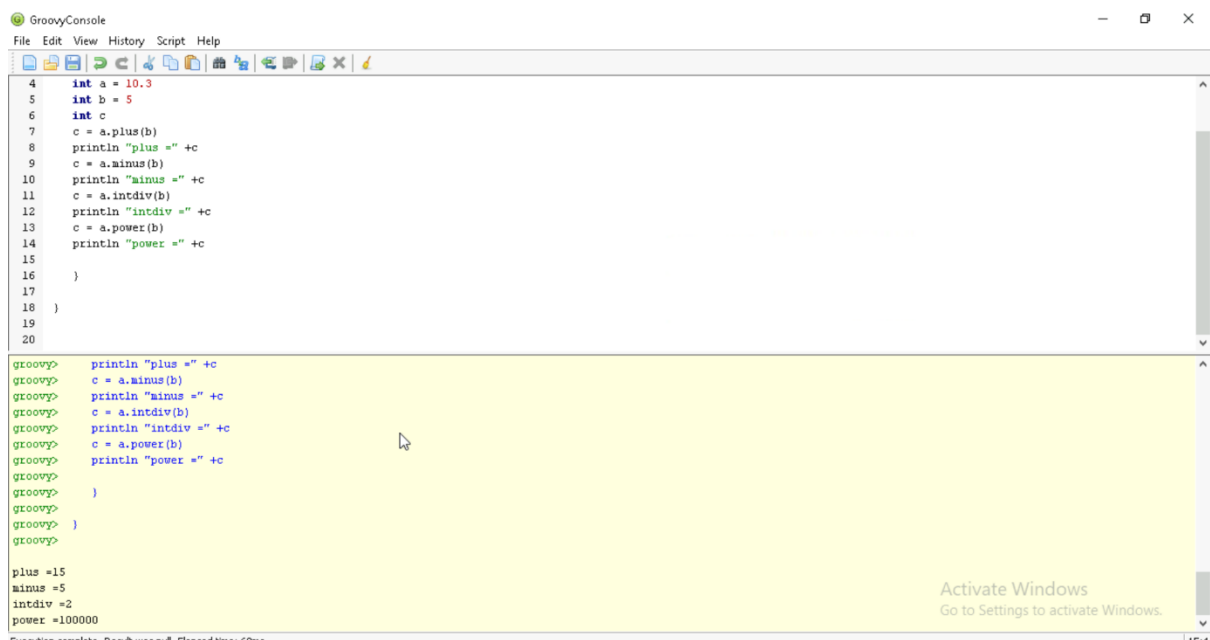
The bottom pane shows the console output:

```
groovy> c = a / b
groovy> println "Division =" + c
groovy> c = a % b
groovy> println "Remainder =" + c
groovy> c = a ** b
groovy> println "Power =" + c
groovy> }
groovy> )
groovy>

Addition =15
Subtraction =5
Multiplication =50
Division =2
Remainder =0
Power =100000
```

An "Activate Windows" watermark is visible in the bottom right corner of the console output area.

## 7) Arithmetic operators



The screenshot shows the GroovyConsole application. The top pane contains a Groovy script with the following code:

```
4  int a = 10.3
5  int b = 5
6  int c
7  c = a.plus(b)
8  println "plus =" + c
9  c = a.minus(b)
10 println "minus =" + c
11 c = a.intdiv(b)
12 println "intdiv =" + c
13 c = a.power(b)
14 println "power =" + c
15
16 }
17
18 )
19
20
```

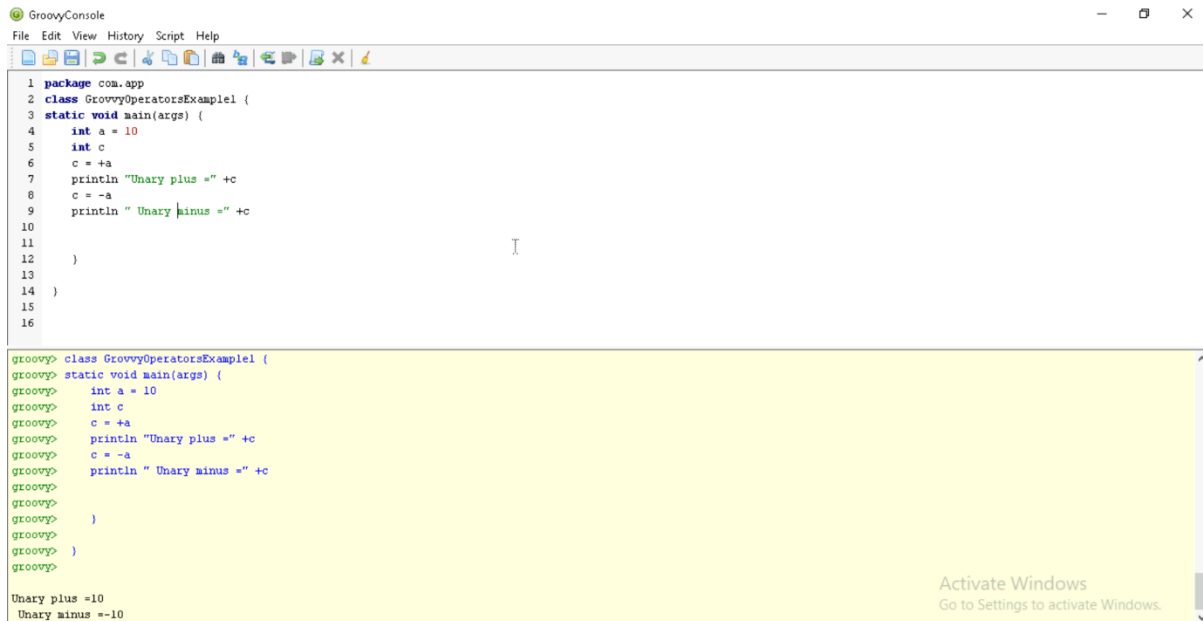
The bottom pane shows the console output:

```
groovy> println "plus =" + c
groovy> c = a.minus(b)
groovy> println "minus =" + c
groovy> c = a.intdiv(b)
groovy> println "intdiv =" + c
groovy> c = a.power(b)
groovy> println "power =" + c
groovy> }
groovy> )
groovy>

plus =15
minus =5
intdiv =2
power =100000
```

An "Activate Windows" watermark is visible in the bottom right corner of the console output area.

## 8) Unary operators



The screenshot shows the GroovyConsole application. The top pane contains a Groovy script defining a class `GroovyOperatorsExample1` with a `main` method. The script initializes `a = 10` and `c`, then performs unary plus and minus operations on `a` and prints the results. The bottom pane shows the command-line execution of the script, with the same code pasted and the output displayed: `Unary plus -10` and `Unary minus --10`. A Windows activation watermark is visible in the bottom right corner.

```
1 package com.app
2 class GroovyOperatorsExample1 {
3     static void main(args) {
4         int a = 10
5         int c
6         c = +a
7         println "Unary plus =" + c
8         c = -a
9         println "Unary minus =" + c
10
11     }
12 }
13
14 }
15
16
```

```
groovy> class GroovyOperatorsExample1 {
groovy> static void main(args) {
groovy>     int a = 10
groovy>     int c
groovy>     c = +a
groovy>     println "Unary plus =" + c
groovy>     c = -a
groovy>     println "Unary minus =" + c
groovy> }
groovy> }
groovy> }
```

Unary plus -10  
Unary minus --10

Activate Windows  
Go to Settings to activate Windows.

## 9) Unary operators



The screenshot shows the GroovyConsole application. The top pane contains a Groovy script defining a class `GroovyOperatorsExample1` with a `main` method. The script initializes `a = 10` and `c`, then performs unary plus and minus operations on `a` and prints the results. The bottom pane shows the command-line execution of the script, with the same code pasted and the output displayed: `Unary plus -10` and `Unary minus --10`. A Windows activation watermark is visible in the bottom right corner.

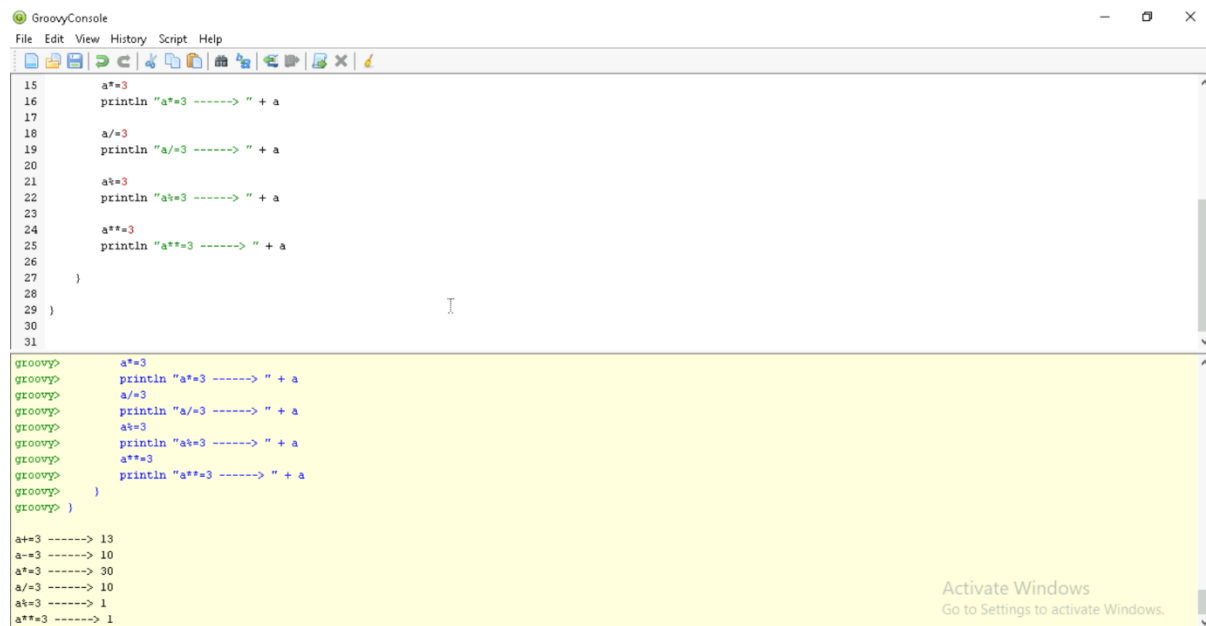
```
15 c = b--
16
17 println "Post decrement =" + c
18
19 println "Value of a after Post decrement =" + b
20
21 c = --b
22
23 println "Pre decrement =" + c
24
25 println "Value of a after Pre decrement =" + b
26
27 }
28
29 }
30
31
```

```
groovy> println "Post decrement =" + c
groovy> println "Value of a after Post decrement =" + b
groovy> c = --b
groovy> println "Pre decrement =" + c
groovy> println "Value of a after Pre decrement =" + b
groovy> }
groovy> }
```

Post Increment = 10  
Value of a after Post Increment = 11  
Pre Increment = 12  
Value of a after Pre Increment = 12  
Post decrement = 10  
Value of a after Post decrement = 9  
Pre decrement = 8  
Value of a after Pre decrement = 8

Activate Windows  
Go to Settings to activate Windows.

## 10) Assignment arithmetic operators



The screenshot shows the GroovyConsole application with a script and its output. The script defines a variable `a` and performs several arithmetic operations on it, printing the results. The output shows the values of `a` after each operation.

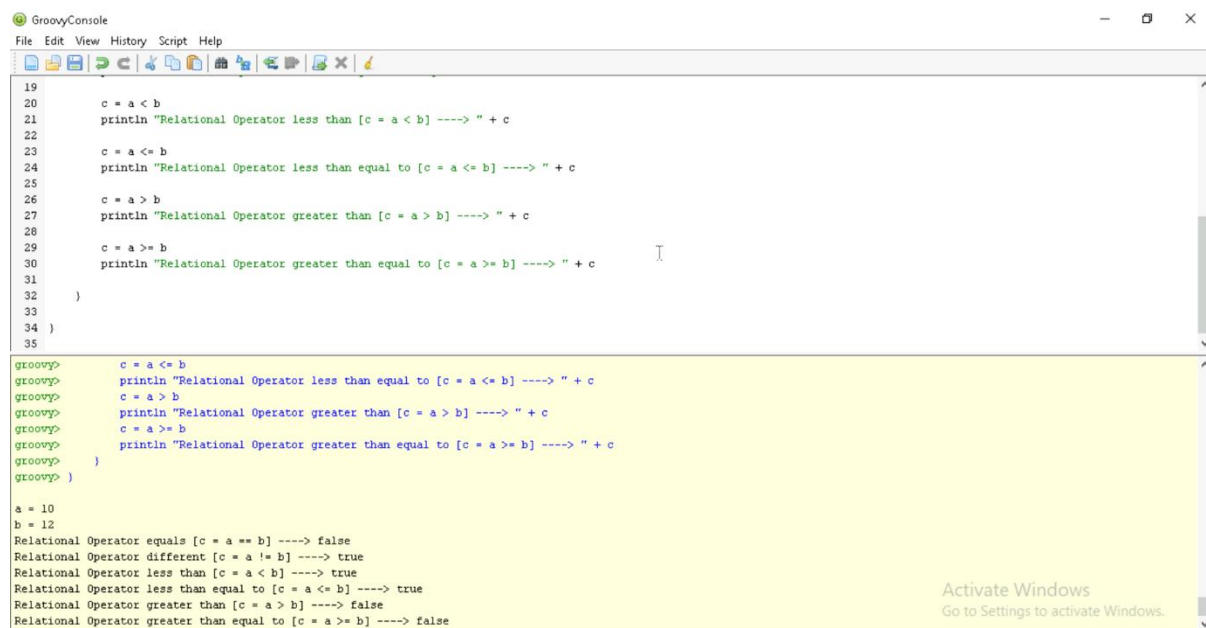
```
15     a*=3
16     println "a*=3 ----> " + a
17
18     a/=3
19     println "a/=3 ----> " + a
20
21     a%=3
22     println "a%=3 ----> " + a
23
24     a**=3
25     println "a**=3 ----> " + a
26
27 }
28
29 }
30
31
```

```
groovy> a*=3
groovy> println "a*=3 ----> " + a
groovy> a/=3
groovy> println "a/=3 ----> " + a
groovy> a%=3
groovy> println "a%=3 ----> " + a
groovy> a**=3
groovy> println "a**=3 ----> " + a
groovy> }
groovy> }
```

```
a*=3 ----> 13
a/=3 ----> 10
a%=3 ----> 30
a/=3 ----> 10
a%=3 ----> 1
a**=3 ----> 1
```

Activate Windows  
Go to Settings to activate Windows.

## 11) Relational operators



The screenshot shows the GroovyConsole application with a script and its output. The script defines variables `a` and `b` and performs several relational operations on them, printing the results. The output shows the boolean results of each relational operation.

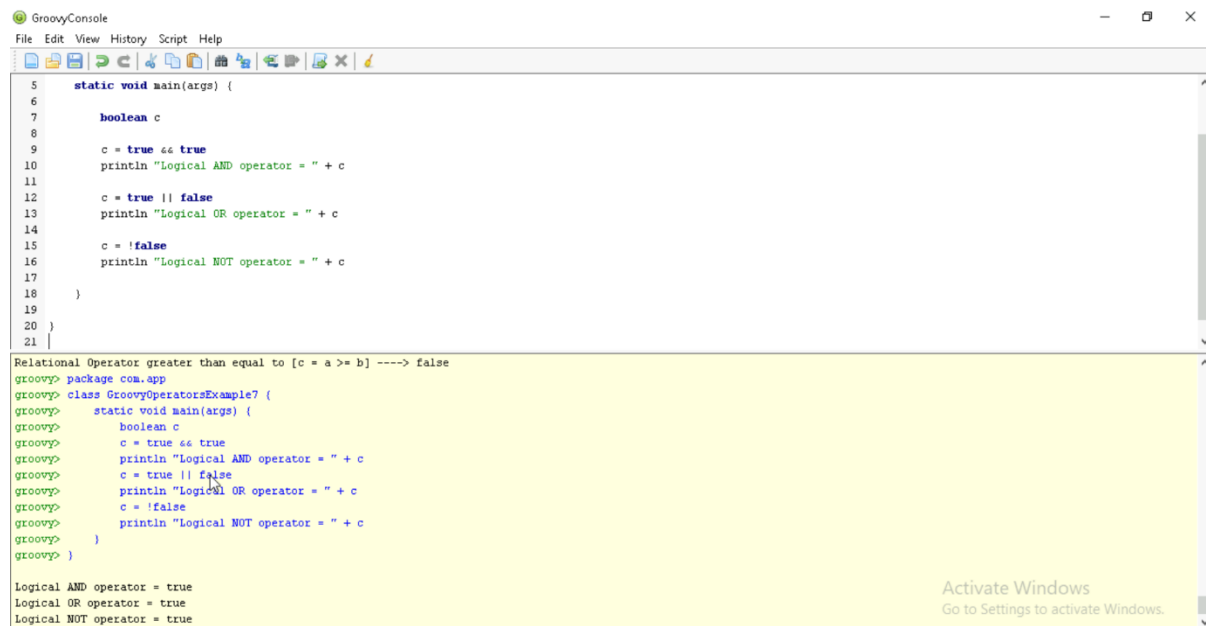
```
19
20     c = a < b
21     println "Relational Operator less than [c = a < b] ----> " + c
22
23     c = a <= b
24     println "Relational Operator less than equal to [c = a <= b] ----> " + c
25
26     c = a > b
27     println "Relational Operator greater than [c = a > b] ----> " + c
28
29     c = a >= b
30     println "Relational Operator greater than equal to [c = a >= b] ----> " + c
31
32 }
33
34 }
35
```

```
groovy> c = a < b
groovy> println "Relational Operator less than equal to [c = a <= b] ----> " + c
groovy> c = a > b
groovy> println "Relational Operator greater than [c = a > b] ----> " + c
groovy> c = a >= b
groovy> println "Relational Operator greater than equal to [c = a >= b] ----> " + c
groovy> }
```

```
a = 10
b = 12
Relational Operator equals [c = a == b] ----> false
Relational Operator different [c = a != b] ----> true
Relational Operator less than [c = a < b] ----> true
Relational Operator less than equal to [c = a <= b] ----> true
Relational Operator greater than [c = a > b] ----> false
Relational Operator greater than equal to [c = a >= b] ----> false
```

Activate Windows  
Go to Settings to activate Windows.

## 12) Logical operators



The screenshot shows the GroovyConsole application. The top pane contains a Groovy script with a `main` method that tests logical operators. The bottom pane shows the output of the script, which is a repetition of the code in the top pane. A watermark "Activate Windows" is visible in the bottom right corner.

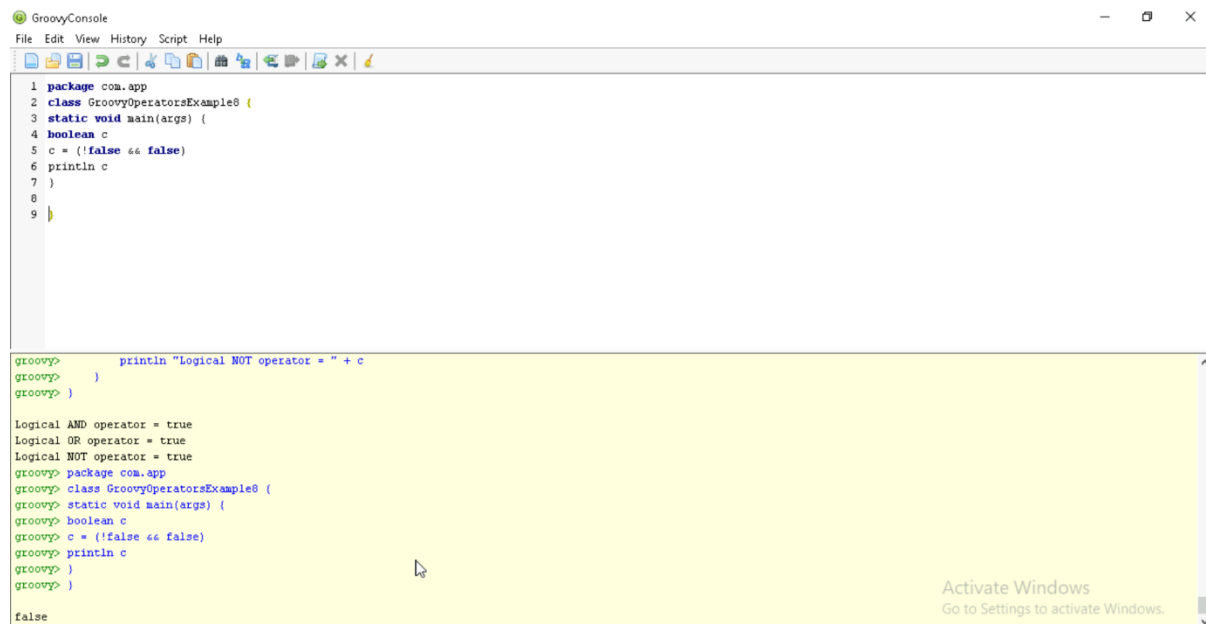
```
5 static void main(args) {
6     boolean c
7
8     c = true && true
9     println "Logical AND operator = " + c
10
11    c = true || false
12    println "Logical OR operator = " + c
13
14    c = !false
15    println "Logical NOT operator = " + c
16
17 }
18
19
20
21
```

Relational Operator greater than equal to [c = a >= b] ----> false

```
groovy> package com.app
groovy> class GroovyOperatorsExample7 {
groovy>     static void main(args) {
groovy>         boolean c
groovy>         c = true && true
groovy>         println "Logical AND operator = " + c
groovy>         c = true || false
groovy>         println "Logical OR operator = " + c
groovy>         c = !false
groovy>         println "Logical NOT operator = " + c
groovy>     }
groovy> }

Logical AND operator = true
Logical OR operator = true
Logical NOT operator = true
```

## 13) Logical operators



The screenshot shows the GroovyConsole application. The top pane contains a Groovy script with a `main` method that tests logical operators. The bottom pane shows the output of the script, which is a repetition of the code in the top pane. A watermark "Activate Windows" is visible in the bottom right corner.

```
1 package com.app
2 class GroovyOperatorsExample8 {
3     static void main(args) {
4         boolean c
5         c = (!false && false)
6         println c
7     }
8 }
9
```

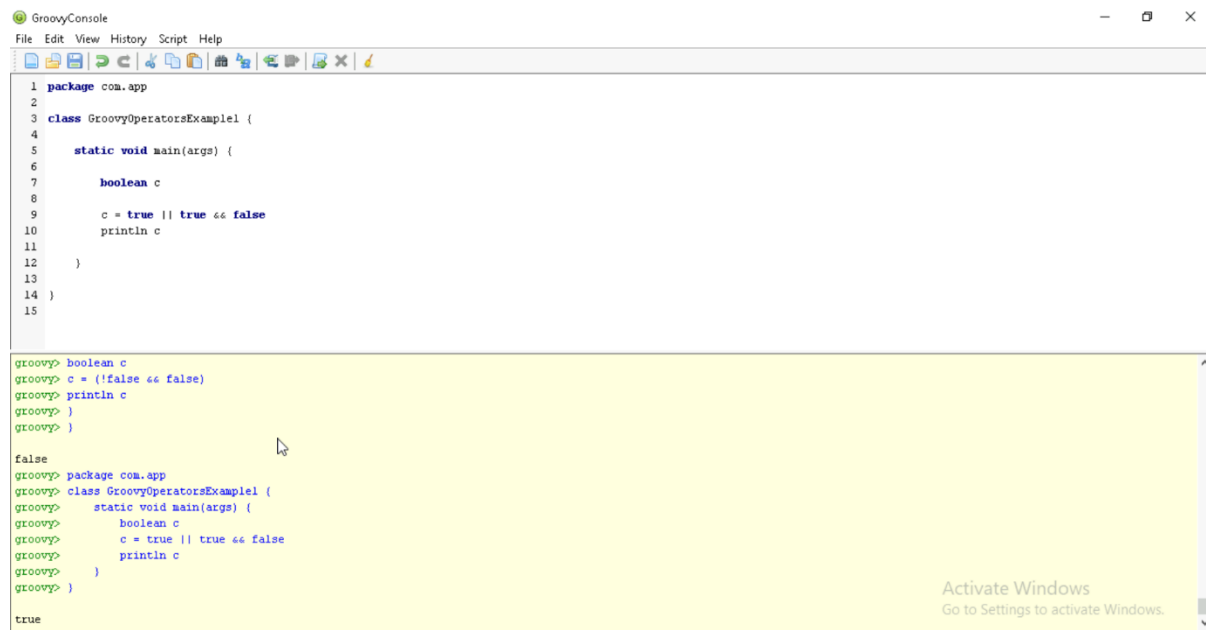
groovy> println "Logical NOT operator = " + c

```
groovy> }
groovy> }

Logical AND operator = true
Logical OR operator = true
Logical NOT operator = true
groovy> package com.app
groovy> class GroovyOperatorsExample8 {
groovy>     static void main(args) {
groovy>         boolean c
groovy>         c = (!false && false)
groovy>         println c
groovy>     }
groovy> }

false
```

## 14) Logical operators

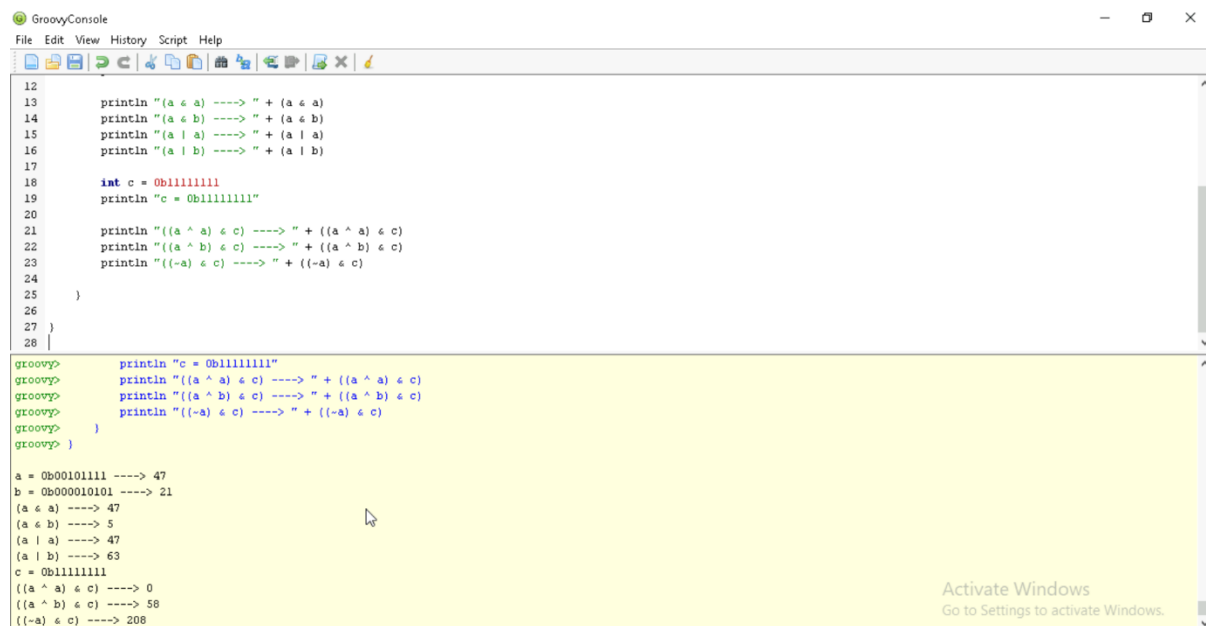


The screenshot shows the GroovyConsole application. The top pane contains Groovy code for a class `GroovyOperatorsExample1` with a `main` method. The code defines a `boolean c` and assigns it the value `true || true && false`, then prints `c`. The bottom pane shows the execution output, which includes the code being run and the final value of `c`, which is `true`.

```
1 package com.app
2
3 class GroovyOperatorsExample1 {
4
5     static void main(args) {
6
7         boolean c
8
9         c = true || true && false
10        println c
11    }
12 }
13
14
15
```

```
groovy> boolean c
groovy> c = (true || true && false)
groovy> println c
groovy> }
false
groovy> package com.app
groovy> class GroovyOperatorsExample1 {
groovy>     static void main(args) {
groovy>         boolean c
groovy>         c = true || true && false
groovy>         println c
groovy>     }
groovy> }
true
```

## 15) Bitwise operators



The screenshot shows the GroovyConsole application. The top pane contains Groovy code for a class `GroovyOperatorsExample1` with a `main` method. The code defines two integers `a` and `b` with binary values, and then prints the results of various bitwise operations: `a & a`, `a & b`, `a | a`, `a | b`, `a ^ a`, `a ^ b`, and `(-a) & c`. The bottom pane shows the execution output, which includes the code being run and the final values of the operations, such as `a & a` resulting in 47 and `a & b` resulting in 5.

```
12
13        println "(a & a) ----> " + (a & a)
14        println "(a & b) ----> " + (a & b)
15        println "(a | a) ----> " + (a | a)
16        println "(a | b) ----> " + (a | b)
17
18        int c = 0b11111111
19        println "c = 0b11111111"
20
21        println "((a ^ a) & c) ----> " + ((a ^ a) & c)
22        println "((a ^ b) & c) ----> " + ((a ^ b) & c)
23        println "((-a) & c) ----> " + ((-a) & c)
24    }
25 }
26
27
28
```

```
groovy> println "c = 0b11111111"
groovy> println "((a ^ a) & c) ----> " + ((a ^ a) & c)
groovy> println "((a ^ b) & c) ----> " + ((a ^ b) & c)
groovy> println "((-a) & c) ----> " + ((-a) & c)
groovy> }
a = 0b00101111 ----> 47
b = 0b000010101 ----> 21
(a & a) ----> 47
(a & b) ----> 5
(a | a) ----> 47
(a | b) ----> 63
c = 0b11111111
((a ^ a) & c) ----> 0
((a ^ b) & c) ----> 58
((-a) & c) ----> 208
```

## 16) Bitwise operators

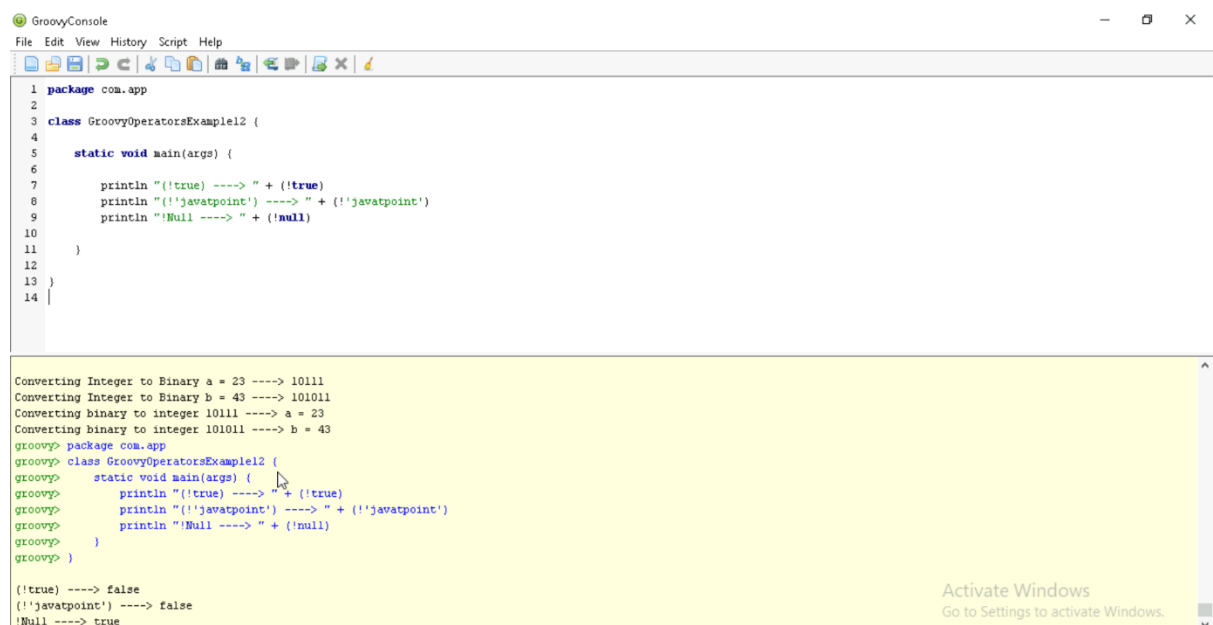


The screenshot shows the GroovyConsole application with a script named `GroovyOperatorsExample11`. The script defines a `main` method that prints the binary representations of integers 23 and 43, and then converts binary strings back to integers. The console output shows the execution of the script, including the package declaration, class definition, and the resulting output of the `main` method. The output shows the binary representations of 23 (10111) and 43 (101011), and the conversion of these binary strings back to integers (23 and 43).

```
3 class GroovyOperatorsExample11 {
4
5     static void main(args) {
6
7         int a = 23
8         int b = 43
9
10        println "Converting Integer to Binary a = 23 ----> " + Integer.toBinaryString(a)
11        println "Converting Integer to Binary b = 43 ----> " + Integer.toBinaryString(b)
12
13        println "Converting binary to integer 10111 ----> a = " + Integer.parseInt("10111", 2)
14        println "Converting binary to integer 101011 ----> b = " + Integer.parseInt("101011", 2)
15
16    }
17
18 }
19
```

```
((-a) & c) ----> 208
groovy> package com.app
groovy> class GroovyOperatorsExample11 {
groovy>     static void main(args) {
groovy>         int a = 23
groovy>         int b = 43
groovy>         println "Converting Integer to Binary a = 23 ----> " + Integer.toBinaryString(a)
groovy>         println "Converting Integer to Binary b = 43 ----> " + Integer.toBinaryString(b)
groovy>         println "Converting binary to integer 10111 ----> a = " + Integer.parseInt("10111", 2)
groovy>         println "Converting binary to integer 101011 ----> b = " + Integer.parseInt("101011", 2)
groovy>     }
groovy> }
Converting Integer to Binary a = 23 ----> 10111
Converting Integer to Binary b = 43 ----> 101011
Converting binary to integer 10111 ----> a = 23
Converting binary to integer 101011 ----> b = 43
Execution complete. Result was null. Elapsed time: 46ms.
```

## 17) Conditional operators



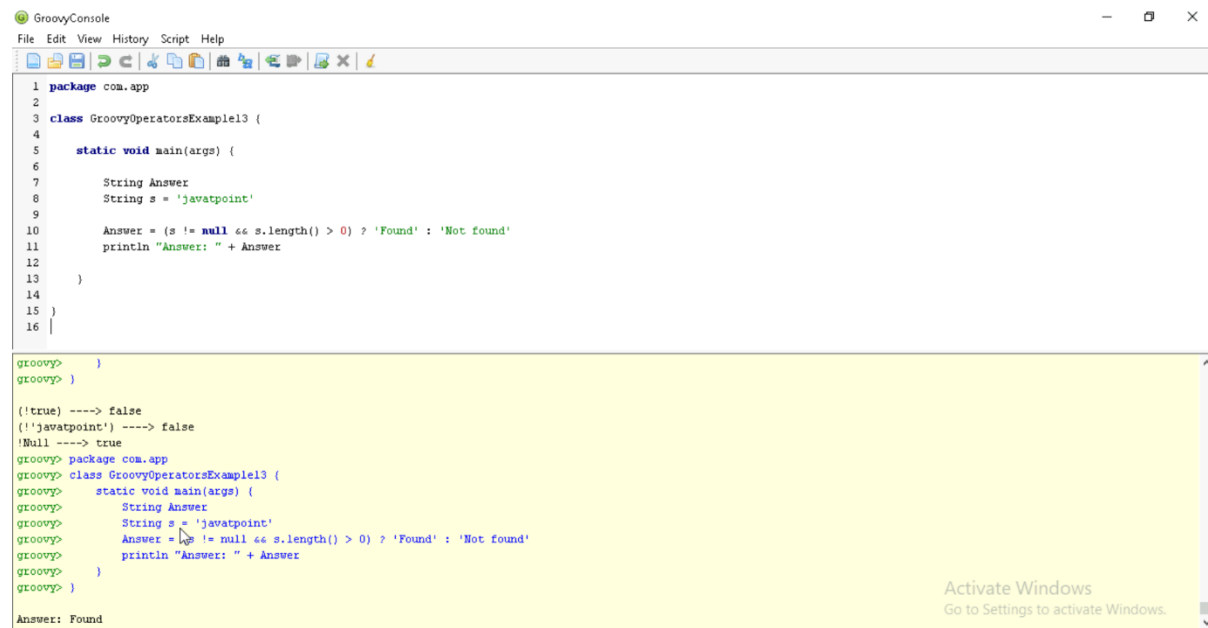
The screenshot shows the GroovyConsole application with a script named `GroovyOperatorsExample12`. The script defines a `main` method that prints the results of conditional expressions: `!true`, `!javatpoint`, and `!null`. The console output shows the execution of the script, including the package declaration, class definition, and the resulting output of the `main` method. The output shows the results of the conditional expressions: `!true` is `false`, `!javatpoint` is `false`, and `!null` is `true`.

```
1 package com.app
2
3 class GroovyOperatorsExample12 {
4
5     static void main(args) {
6
7         println "(!true) ----> " + (!true)
8         println "(!javatpoint) ----> " + (!javatpoint)
9         println "(!null) ----> " + (!null)
10
11    }
12
13 }
14
```

```
Converting Integer to Binary a = 23 ----> 10111
Converting Integer to Binary b = 43 ----> 101011
Converting binary to integer 10111 ----> a = 23
Converting binary to integer 101011 ----> b = 43
groovy> package com.app
groovy> class GroovyOperatorsExample12 {
groovy>     static void main(args) {
groovy>         println "(!true) ----> " + (!true)
groovy>         println "(!javatpoint) ----> " + (!javatpoint)
groovy>         println "(!null) ----> " + (!null)
groovy>     }
groovy> }
(!true) ----> false
(!javatpoint) ----> false
!null ----> true
```



## 18) Conditional operators



The screenshot shows a Groovy IDE window titled "GroovyConsole". The editor displays a Groovy script with a conditional operator. The script defines a class `GroovyOperatorsExample13` with a `main` method. Inside `main`, a string `s` is assigned the value `'javatpoint'`. A conditional expression is used to assign a value to `Answer` based on whether `s` is null and its length is greater than 0. The output of the script is shown in the console, confirming that the condition is true and the output is "Answer: Found".

```
1 package com.app
2
3 class GroovyOperatorsExample13 {
4
5     static void main(args) {
6
7         String Answer
8         String s = 'javatpoint'
9
10        Answer = (s != null && s.length() > 0) ? 'Found' : 'Not found'
11        println "Answer: " + Answer
12    }
13 }
14
15 }
16
```

```
groovy> }
groovy> }

(!true) ----> false
(!'javatpoint') ----> false
(!Null) ----> true
groovy> package com.app
groovy> class GroovyOperatorsExample13 {
groovy>     static void main(args) {
groovy>         String Answer
groovy>         String s = 'javatpoint'
groovy>         Answer = (s != null && s.length() > 0) ? 'Found' : 'Not found'
groovy>         println "Answer: " + Answer
groovy>     }
groovy> }
```

Answer: Found

Activate Windows  
Go to Settings to activate Windows.