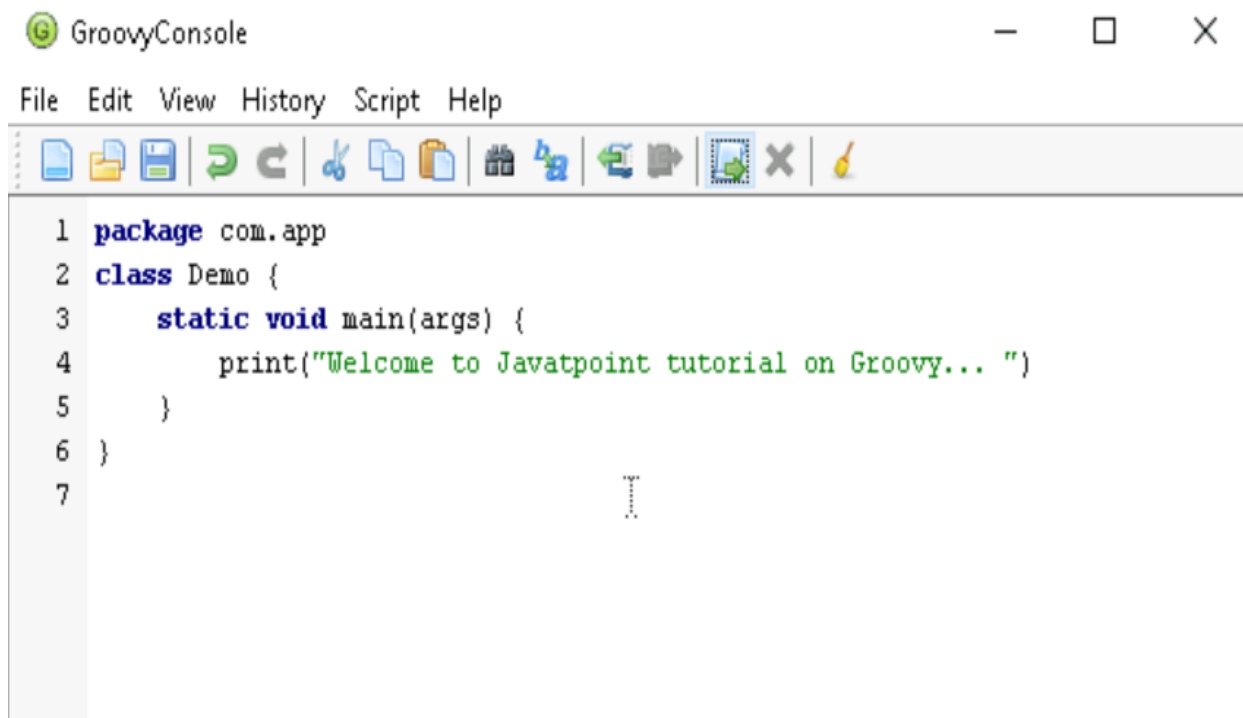


GROOVY

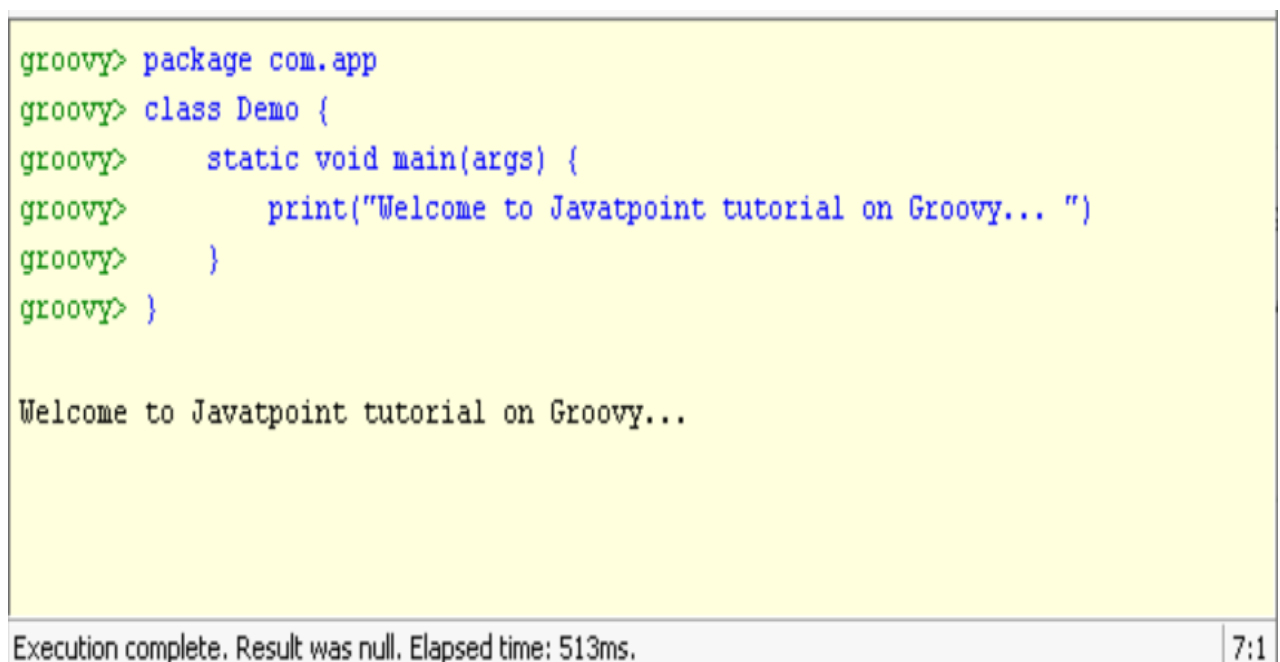
Basic syntax in groovy

- In groovy, it is not necessary to put a semicolon at the end of the statement like in java.

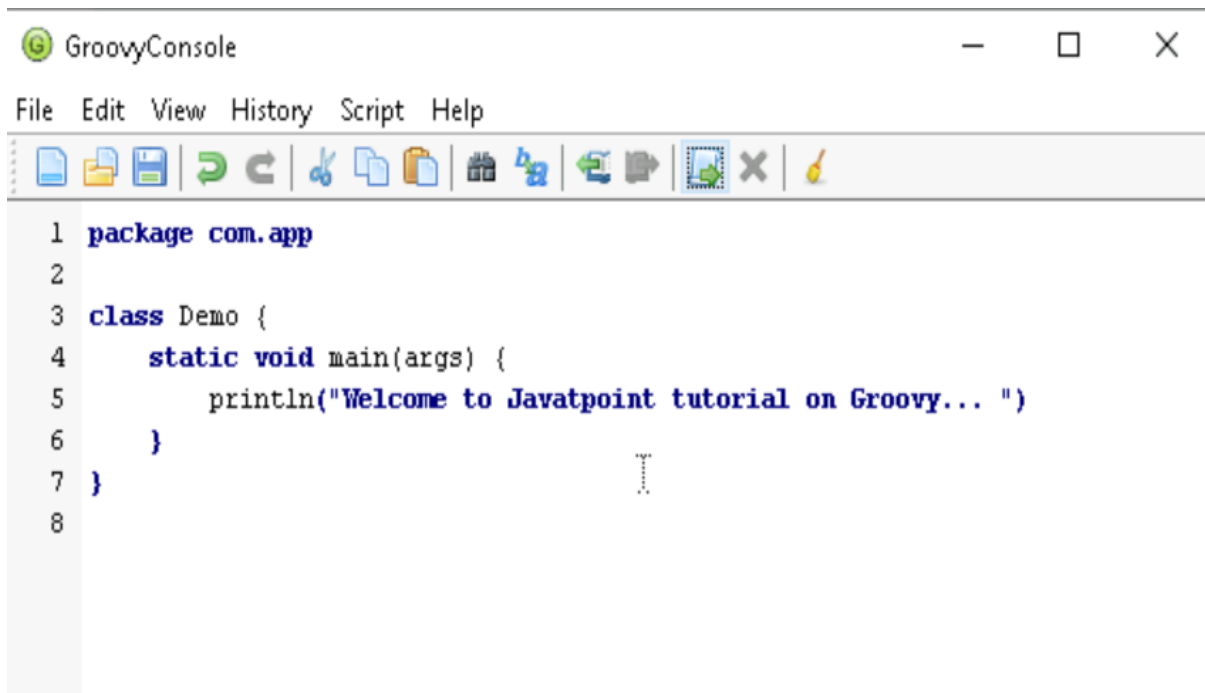
Groovy Script to print the basic script.



Output:



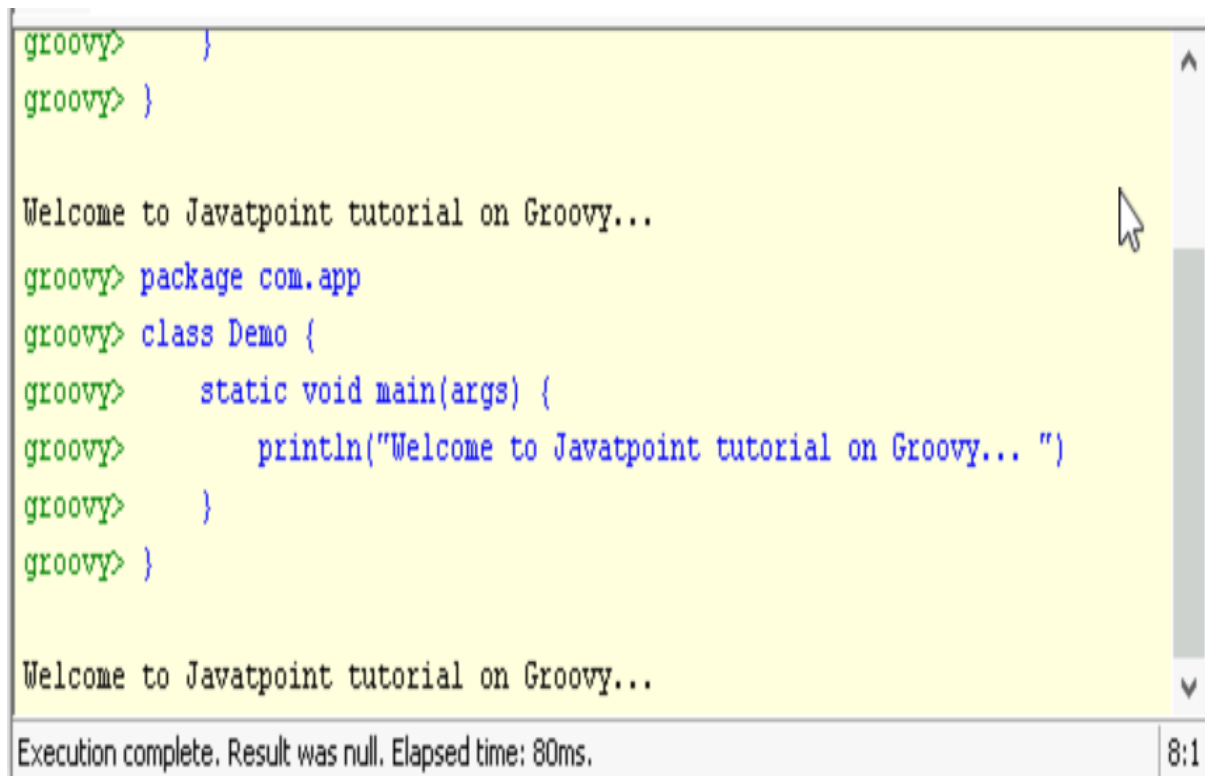
- In groovy, we can print a line without using round brackets



The screenshot shows the GroovyConsole application window. The title bar reads "GroovyConsole" with standard window controls. The menu bar includes "File", "Edit", "View", "History", "Script", and "Help". The toolbar contains icons for file operations (new, open, save, print), editing (undo, redo, cut, copy, paste), and execution (run, stop, refresh). The script area contains the following code:

```
1 package com.app
2
3 class Demo {
4     static void main(args) {
5         println("Welcome to Javatpoint tutorial on Groovy... ")
6     }
7 }
8
```

Output:



The screenshot shows the output window of the GroovyConsole application. The background is yellow. The output text is as follows:

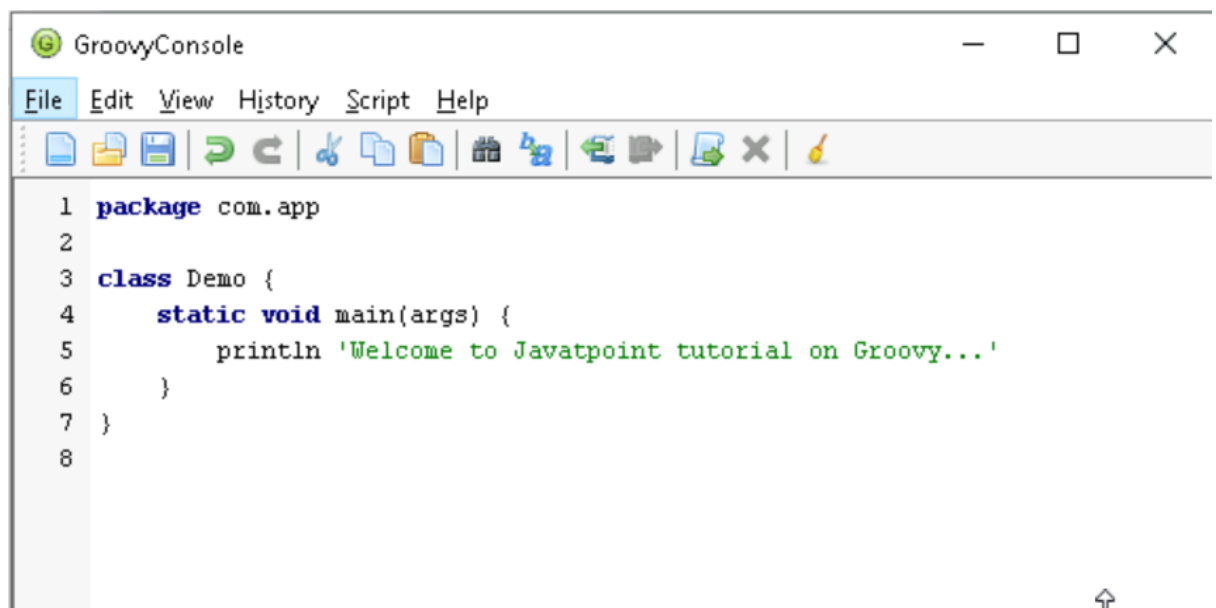
```
groovy> }
groovy> }

Welcome to Javatpoint tutorial on Groovy...

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

At the bottom of the output area, the text "Welcome to Javatpoint tutorial on Groovy..." is displayed. The status bar at the bottom of the window shows "Execution complete. Result was null. Elapsed time: 80ms." and the cursor position "8:1".

- In groovy, double quotes as well as single quotes can be used in a string.

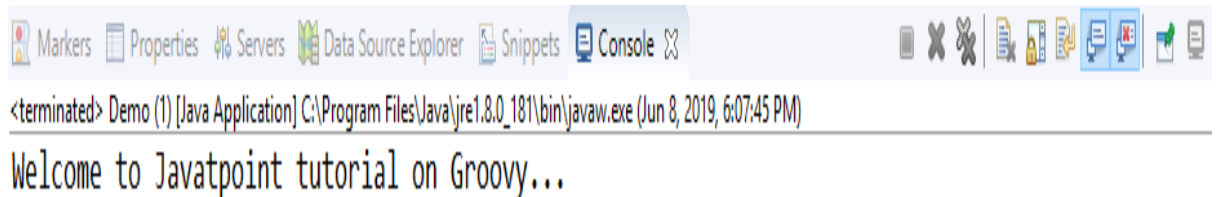


The screenshot shows the GroovyConsole application window. The title bar reads "GroovyConsole". The menu bar includes "File", "Edit", "View", "History", "Script", and "Help". The toolbar contains icons for file operations (new, open, save, delete), editing (undo, redo, cut, copy, paste), and execution (run, stop, refresh). The main text area contains the following Groovy code:

```
1 package com.app
2
3 class Demo {
4     static void main(args) {
5         println 'Welcome to Javatpoint tutorial on Groovy...'
6     }
7 }
8
```

At the bottom right of the window, there is an upward-pointing arrow icon.

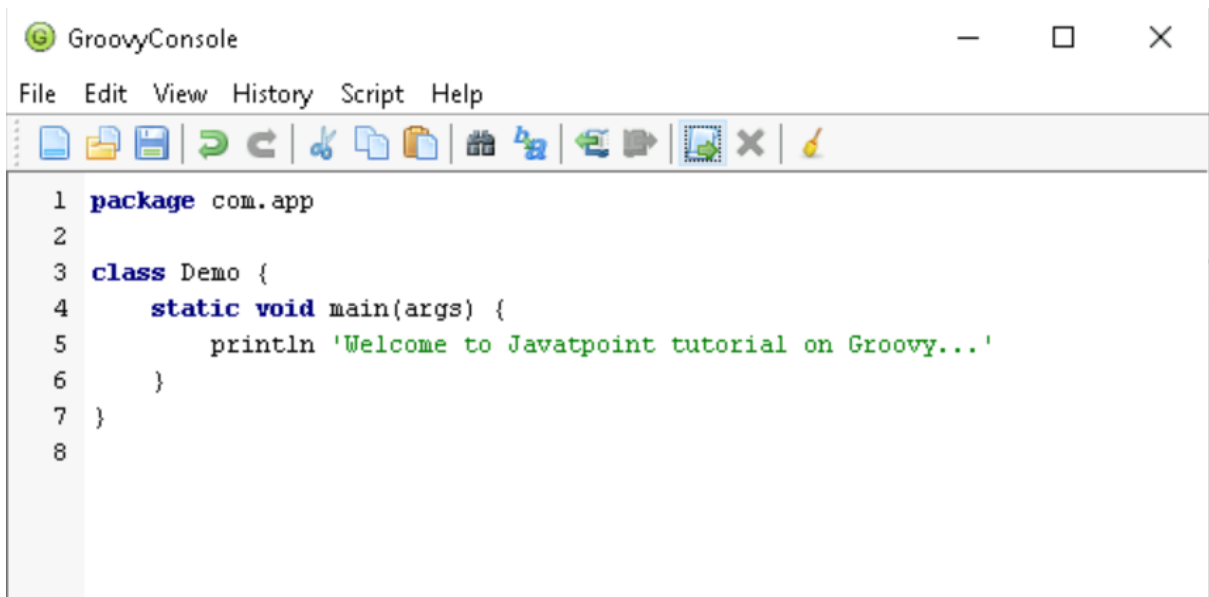
Output:



The screenshot shows the IDE's console window. The title bar includes "Markers", "Properties", "Servers", "Data Source Explorer", "Snippets", and "Console". The console output shows the execution of the Groovy script:

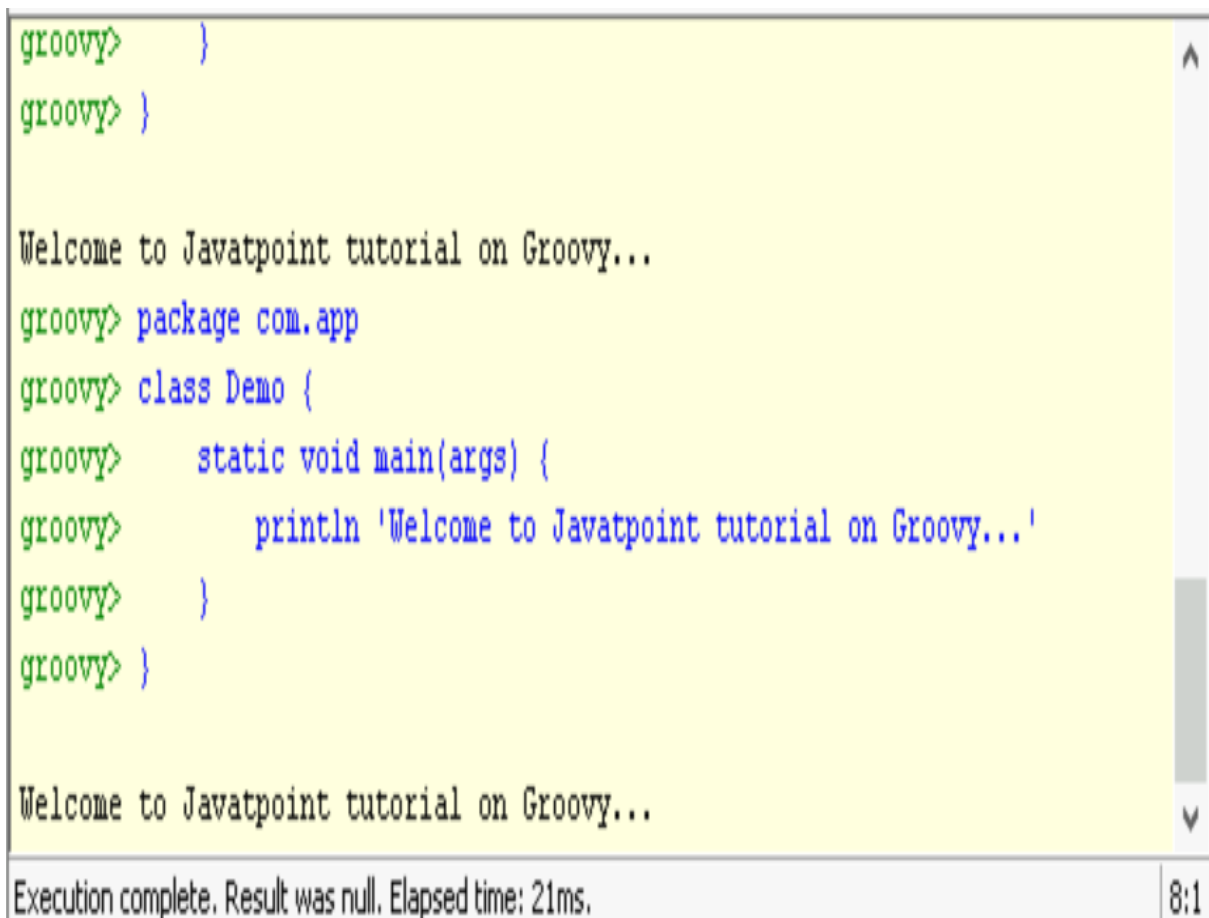
```
<terminated> Demo (1) [Java Application] C:\Program Files\Java\jre1.8.0_181\bin\javaw.exe (Jun 8, 2019, 6:07:45 PM)
Welcome to Javatpoint tutorial on Groovy...
```

We can have a single line comment as well as a multi-line comment just like in java.

A screenshot of the GroovyConsole application window. The window has a title bar with the Groovy logo and the text "GroovyConsole". Below the title bar is a menu bar with "File", "Edit", "View", "History", "Script", and "Help". Below the menu bar is a toolbar with various icons for file operations and execution. The main area of the window displays a Groovy script with line numbers 1 through 8 on the left. The script defines a package, a class, and a static main method that prints a welcome message.

```
1 package com.app
2
3 class Demo {
4     static void main(args) {
5         println 'Welcome to Javatpoint tutorial on Groovy...'
6     }
7 }
8
```

Output:

A screenshot of the GroovyConsole application window showing the output of the script. The window has a yellow background. The output is displayed in a monospaced font. It shows the Groovy prompt "groovy>" followed by the script code being entered. After the code is entered, the output "Welcome to Javatpoint tutorial on Groovy..." is displayed. At the bottom of the window, a status bar shows "Execution complete. Result was null. Elapsed time: 21ms." and a line number "8:1".

```
groovy> }
groovy> }

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

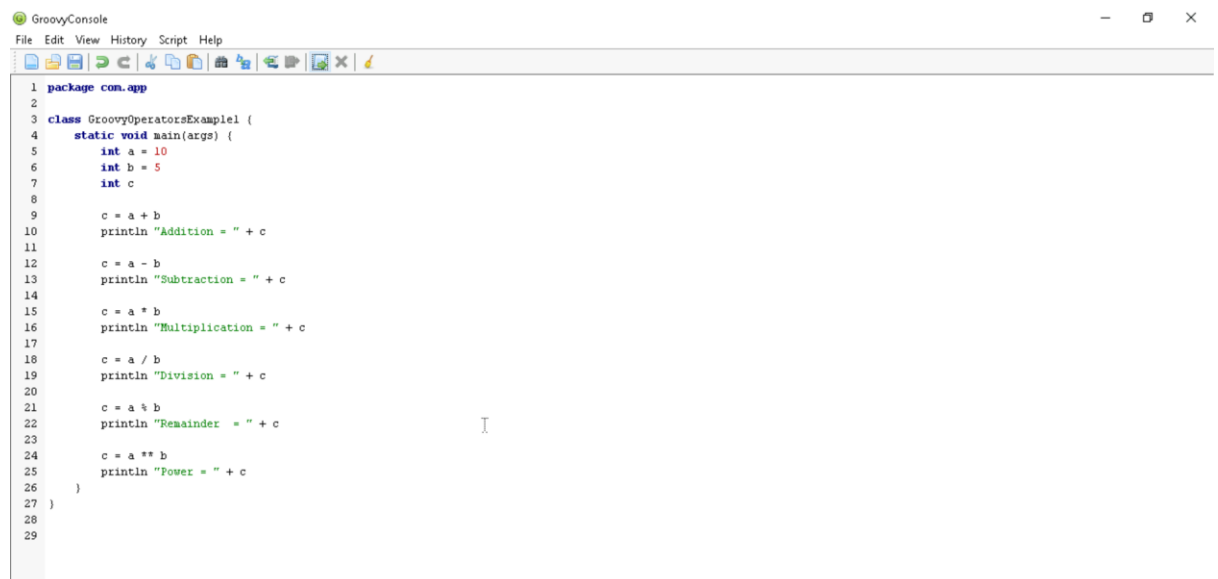
Welcome to Javatpoint tutorial on Groovy...

Execution complete. Result was null. Elapsed time: 21ms. 8:1
```

Arithmetic operator

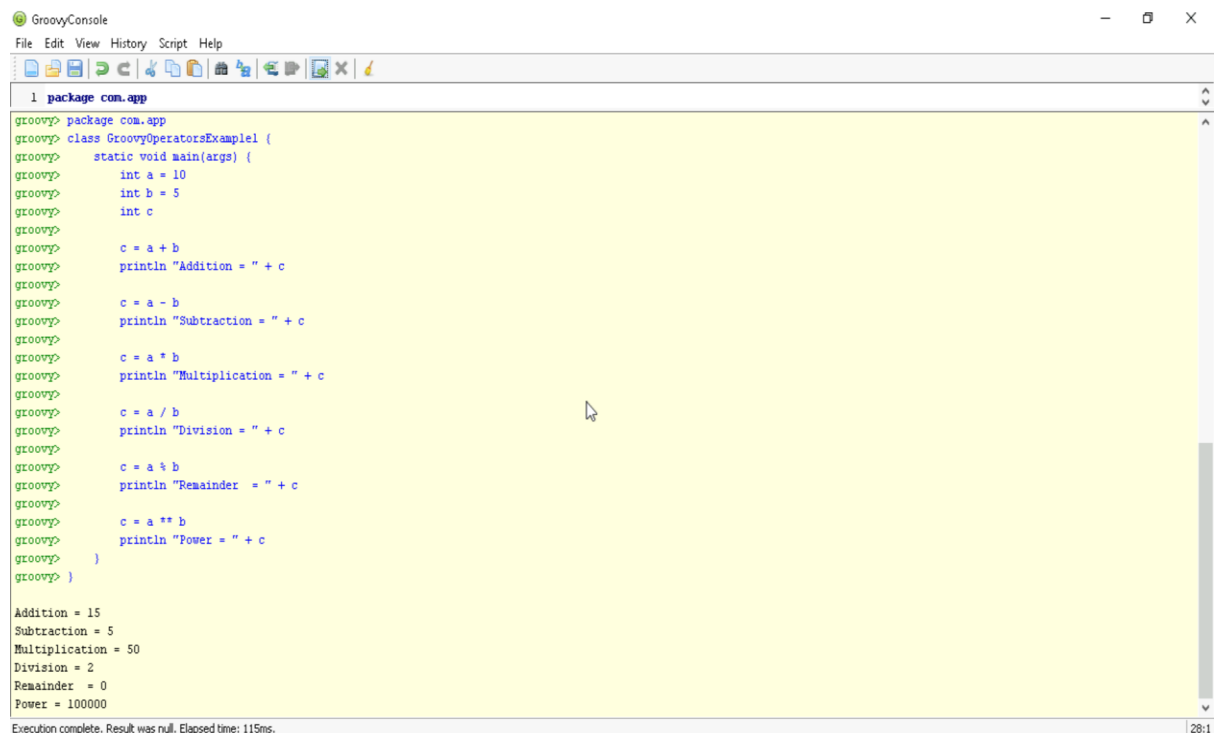
Arithmetic operators are the basic mathematical operators, which are used to perform addition, subtraction, multiplication, division, Remainder and Power.

Example 1:



```
1 package com.app
2
3 class GroovyOperatorsExample1 {
4     static void main(args) {
5         int a = 10
6         int b = 5
7         int c
8
9         c = a + b
10        println "Addition = " + c
11
12        c = a - b
13        println "Subtraction = " + c
14
15        c = a * b
16        println "Multiplication = " + c
17
18        c = a / b
19        println "Division = " + c
20
21        c = a % b
22        println "Remainder = " + c
23
24        c = a ** b
25        println "Power = " + c
26    }
27 }
28
29
```

Output:



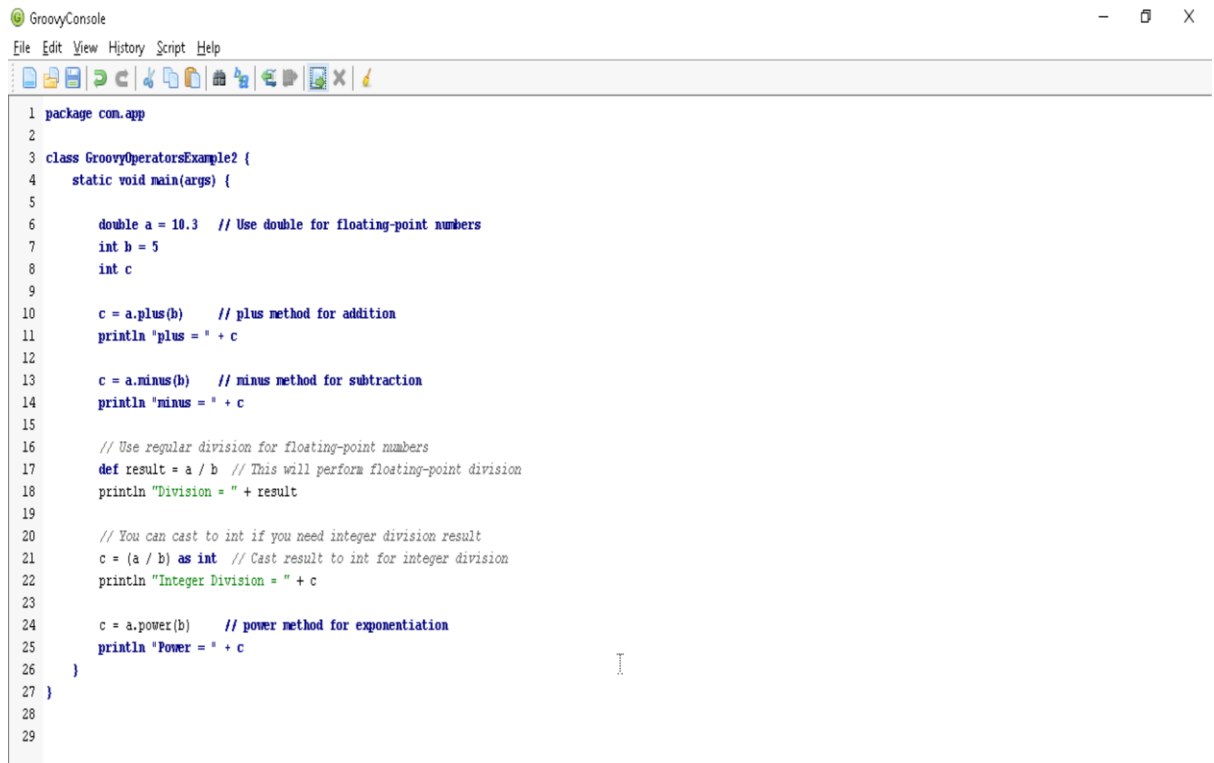
```
1 package com.app
2
3 class GroovyOperatorsExample1 {
4     static void main(args) {
5         int a = 10
6         int b = 5
7         int c
8
9         c = a + b
10        println "Addition = " + c
11
12        c = a - b
13        println "Subtraction = " + c
14
15        c = a * b
16        println "Multiplication = " + c
17
18        c = a / b
19        println "Division = " + c
20
21        c = a % b
22        println "Remainder = " + c
23
24        c = a ** b
25        println "Power = " + c
26    }
27 }
28
29
```

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

Execution complete. Result was null. Elapsed time: 115ms.

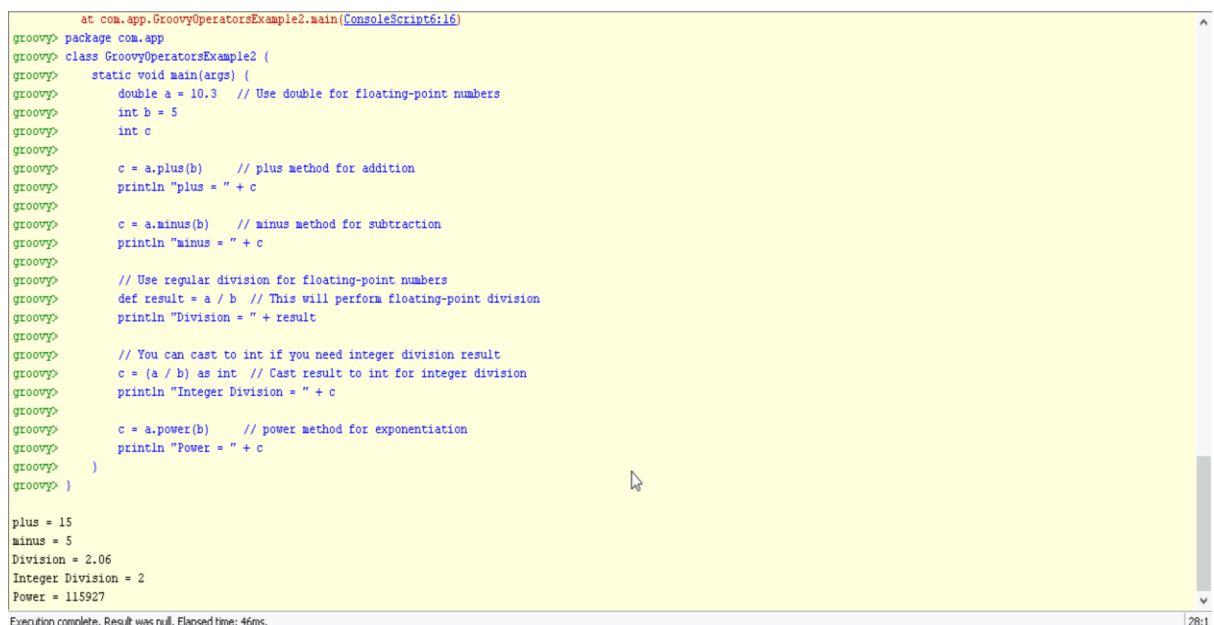
In groovy, we also have some functions which are used to perform Arithmetic operations like plus, minus, intdiv and power. The use of these functions are shown in the example which is given below.

Example 2:



```
1 package com.app
2
3 class GroovyOperatorsExample2 {
4     static void main(args) {
5
6         double a = 10.3 // Use double for floating-point numbers
7         int b = 5
8         int c
9
10        c = a.plus(b) // plus method for addition
11        println "plus = " + c
12
13        c = a.minus(b) // minus method for subtraction
14        println "minus = " + c
15
16        // Use regular division for floating-point numbers
17        def result = a / b // This will perform floating-point division
18        println "Division = " + result
19
20        // You can cast to int if you need integer division result
21        c = (a / b) as int // Cast result to int for integer division
22        println "Integer Division = " + c
23
24        c = a.power(b) // power method for exponentiation
25        println "Power = " + c
26    }
27 }
28
29
```

Output:



```
at com.app.GroovyOperatorsExample2.main(ConsoleScript6:16)
groovy> package com.app
groovy> class GroovyOperatorsExample2 {
groovy>     static void main(args) {
groovy>         double a = 10.3 // Use double for floating-point numbers
groovy>         int b = 5
groovy>         int c
groovy>
groovy>         c = a.plus(b) // plus method for addition
groovy>         println "plus = " + c
groovy>
groovy>         c = a.minus(b) // minus method for subtraction
groovy>         println "minus = " + c
groovy>
groovy>         // Use regular division for floating-point numbers
groovy>         def result = a / b // This will perform floating-point division
groovy>         println "Division = " + result
groovy>
groovy>         // You can cast to int if you need integer division result
groovy>         c = (a / b) as int // Cast result to int for integer division
groovy>         println "Integer Division = " + c
groovy>
groovy>         c = a.power(b) // power method for exponentiation
groovy>         println "Power = " + c
groovy>     }
groovy> }

plus = 15
minus = 5
Division = 2.06
Integer Division = 2
Power = 115927

Execution complete. Result was null. Elapsed time: 46ms. 28:1
```

Unary operators

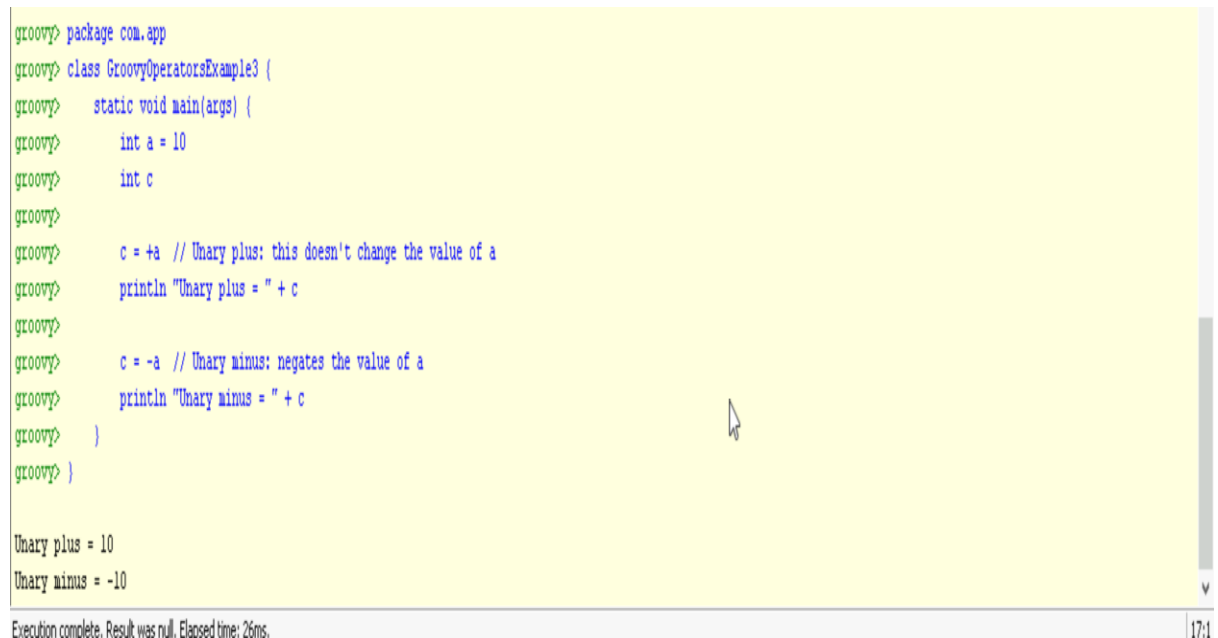
In groovy, Unary operators require only one operator to perform the operation. Unary operators are used to perform the operations such as increment/decrement, negating, and inverting the values of a Boolean.

Example 3:



```
1 package com.app
2
3 class GroovyOperatorsExample3 {
4     static void main(args) {
5         int a = 10
6         int c
7
8         c = +a // Unary plus: this doesn't change the value of a
9         println "Unary plus = " + c
10
11        c = -a // Unary minus: negates the value of a
12        println "Unary minus = " + c
13    }
14 }
15
16
17
```

Output:



```
groovy> package com.app
groovy> class GroovyOperatorsExample3 {
groovy>     static void main(args) {
groovy>         int a = 10
groovy>         int c
groovy>
groovy>         c = +a // Unary plus: this doesn't change the value of a
groovy>         println "Unary plus = " + c
groovy>
groovy>         c = -a // Unary minus: negates the value of a
groovy>         println "Unary minus = " + c
groovy>     }
groovy> }

Unary plus = 10
Unary minus = -10

Execution complete. Result was null. Elapsed time: 26ms. 17:1
```

Example 4:

```
GroovyConsole
File Edit View History Script Help

1 package con.app
2
3 class GroovyOperatorsExample4 {
4     static void main(args) {
5         int a = 10
6         int c
7
8         // Post-increment: Returns the value, then increments
9         c = a++
10        println "Post Increment = " + c
11        println "Value of a after Post Increment = " + a
12
13        // Pre-increment: Increments the value, then returns it
14        c = ++a
15        println "Pre Increment = " + c
16        println "Value of a after Pre Increment = " + a
17
18        int b = 10
19
20        // Post-decrement: Returns the value, then decrements
21        c = b--
22        println "Post Decrement = " + c
23        println "Value of b after Post Decrement = " + b
24
25        // Pre-decrement: Decrements the value, then returns it
26        c = --b
27        println "Pre Decrement = " + c
28        println "Value of b after Pre Decrement = " + b
29    }
30 }
31
32
```

Output:

```
groovy>
groovy> // Post-increment: Returns the value, then increments
groovy> c = a++
groovy> println "Post Increment = " + c
groovy> println "Value of a after Post Increment = " + a
groovy>
groovy> // Pre-increment: Increments the value, then returns it
groovy> c = ++a
groovy> println "Pre Increment = " + c
groovy> println "Value of a after Pre Increment = " + a
groovy>
groovy> int b = 10
groovy>
groovy> // Post-decrement: Returns the value, then decrements
groovy> c = b--
groovy> println "Post Decrement = " + c
groovy> println "Value of b after Post Decrement = " + b
groovy>
groovy> // Pre-decrement: Decrements the value, then returns it
groovy> c = --b
groovy> println "Pre Decrement = " + c
groovy> println "Value of b after Pre Decrement = " + b
groovy> }
groovy> }

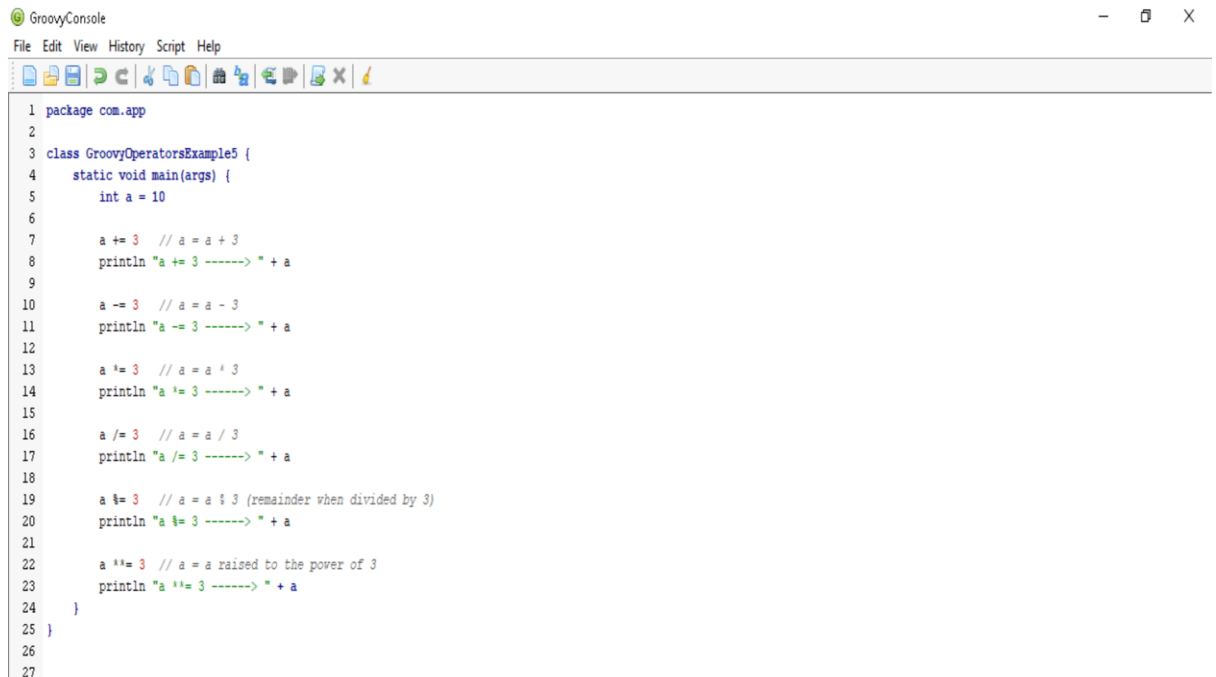
Post Increment = 10
Value of a after Post Increment = 11
Pre Increment = 12
Value of a after Pre Increment = 12
Post Decrement = 10
Value of b after Post Decrement = 9
Pre Decrement = 8
Value of b after Pre Decrement = 8

Execution complete. Result was null. Elapsed time: 47ms. 31:1
```


Assignment arithmetic operators

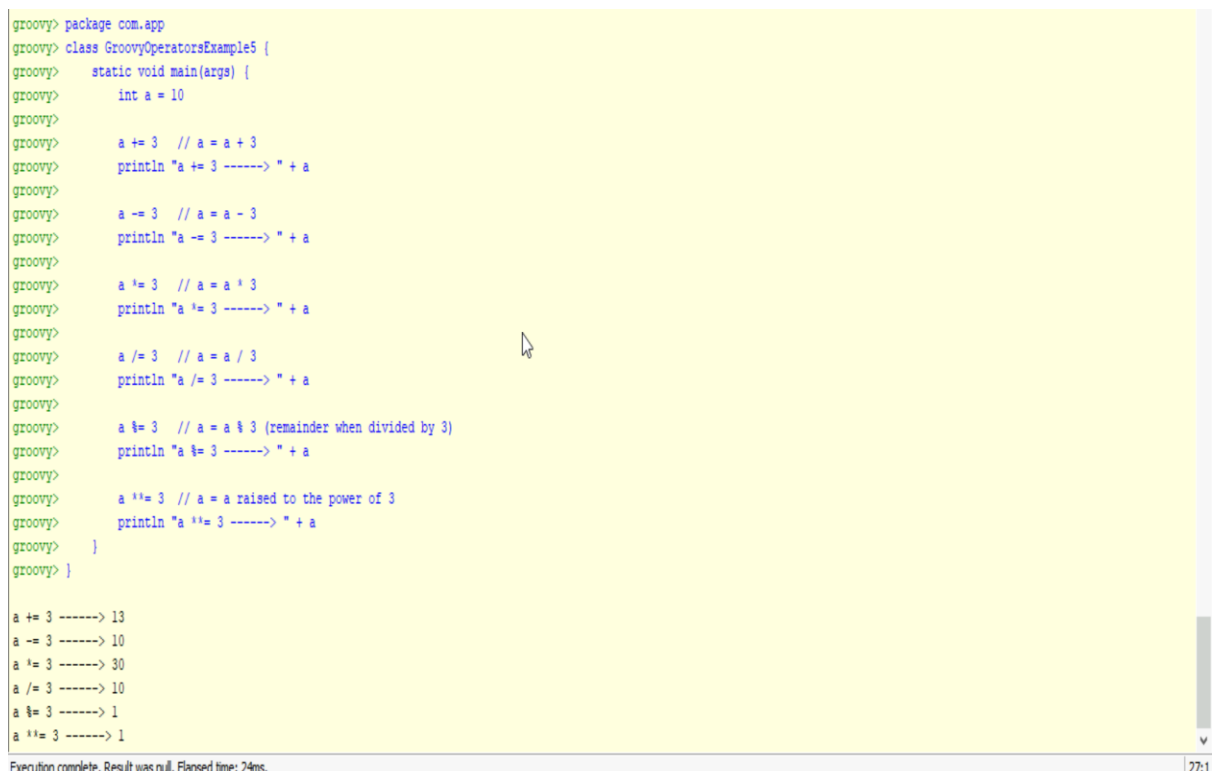
In groovy, assignment arithmetic operators are used to assign a new value to the variable.

Example 5:



```
1 package com.app
2
3 class GroovyOperatorsExample5 {
4     static void main(args) {
5         int a = 10
6
7         a += 3 // a = a + 3
8         println "a += 3 -----> " + a
9
10        a -= 3 // a = a - 3
11        println "a -= 3 -----> " + a
12
13        a *= 3 // a = a * 3
14        println "a *= 3 -----> " + a
15
16        a /= 3 // a = a / 3
17        println "a /= 3 -----> " + a
18
19        a %= 3 // a = a % 3 (remainder when divided by 3)
20        println "a %= 3 -----> " + a
21
22        a **= 3 // a = a raised to the power of 3
23        println "a **= 3 -----> " + a
24    }
25 }
26
27
```

Output:



```
groovy> package com.app
groovy> class GroovyOperatorsExample5 {
groovy>     static void main(args) {
groovy>         int a = 10
groovy>
groovy>         a += 3 // a = a + 3
groovy>         println "a += 3 -----> " + a
groovy>
groovy>         a -= 3 // a = a - 3
groovy>         println "a -= 3 -----> " + a
groovy>
groovy>         a *= 3 // a = a * 3
groovy>         println "a *= 3 -----> " + a
groovy>
groovy>         a /= 3 // a = a / 3
groovy>         println "a /= 3 -----> " + a
groovy>
groovy>         a %= 3 // a = a % 3 (remainder when divided by 3)
groovy>         println "a %= 3 -----> " + a
groovy>
groovy>         a **= 3 // a = a raised to the power of 3
groovy>         println "a **= 3 -----> " + a
groovy>     }
groovy> }

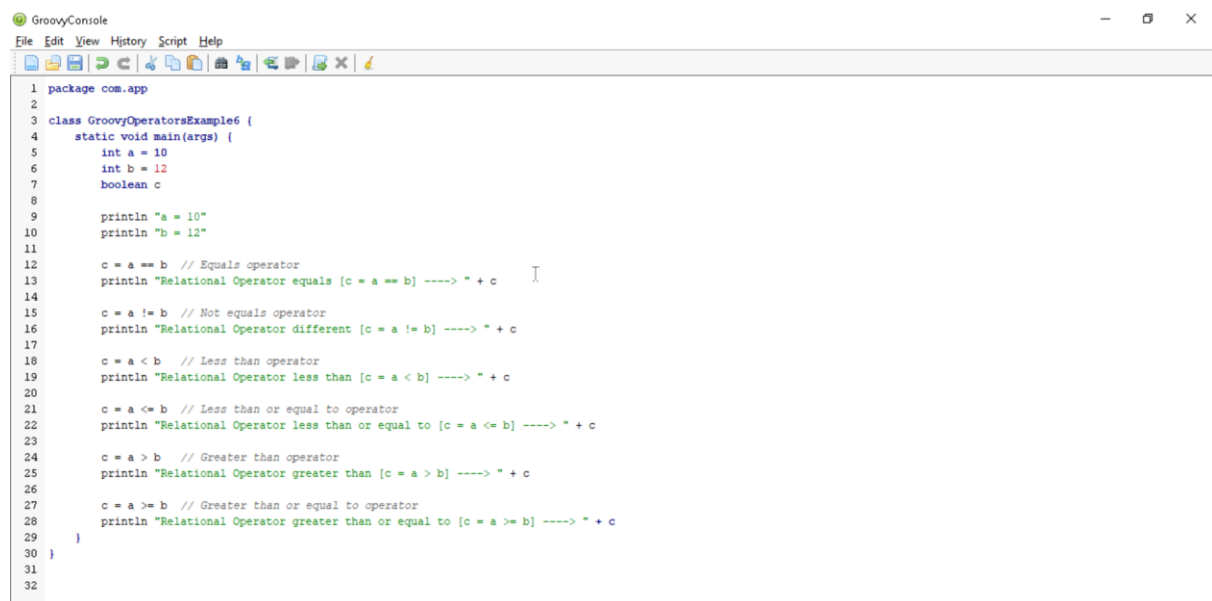
a += 3 -----> 13
a -= 3 -----> 10
a *= 3 -----> 30
a /= 3 -----> 10
a %= 3 -----> 1
a **= 3 -----> 1

Execution complete. Result was null. Elapsed time: 24ms.
```

Relational operators


In groovy, relational operators are used to compare two objects to check whether they are same or different or one is greater than, less than or equal to other object.

Example 6:



```
1 package com.app
2
3 class GroovyOperatorsExample6 {
4     static void main(args) {
5         int a = 10
6         int b = 12
7         boolean c
8
9         println "a = 10"
10        println "b = 12"
11
12        c = a == b // Equals operator
13        println "Relational Operator equals [c = a == b] ----> " + c
14
15        c = a != b // Not equals operator
16        println "Relational Operator different [c = a != b] ----> " + c
17
18        c = a < b // Less than operator
19        println "Relational Operator less than [c = a < b] ----> " + c
20
21        c = a <= b // Less than or equal to operator
22        println "Relational Operator less than or equal to [c = a <= b] ----> " + c
23
24        c = a > b // Greater than operator
25        println "Relational Operator greater than [c = a > b] ----> " + c
26
27        c = a >= b // Greater than or equal to operator
28        println "Relational Operator greater than or equal to [c = a >= b] ----> " + c
29    }
30 }
31
32
```

Output:

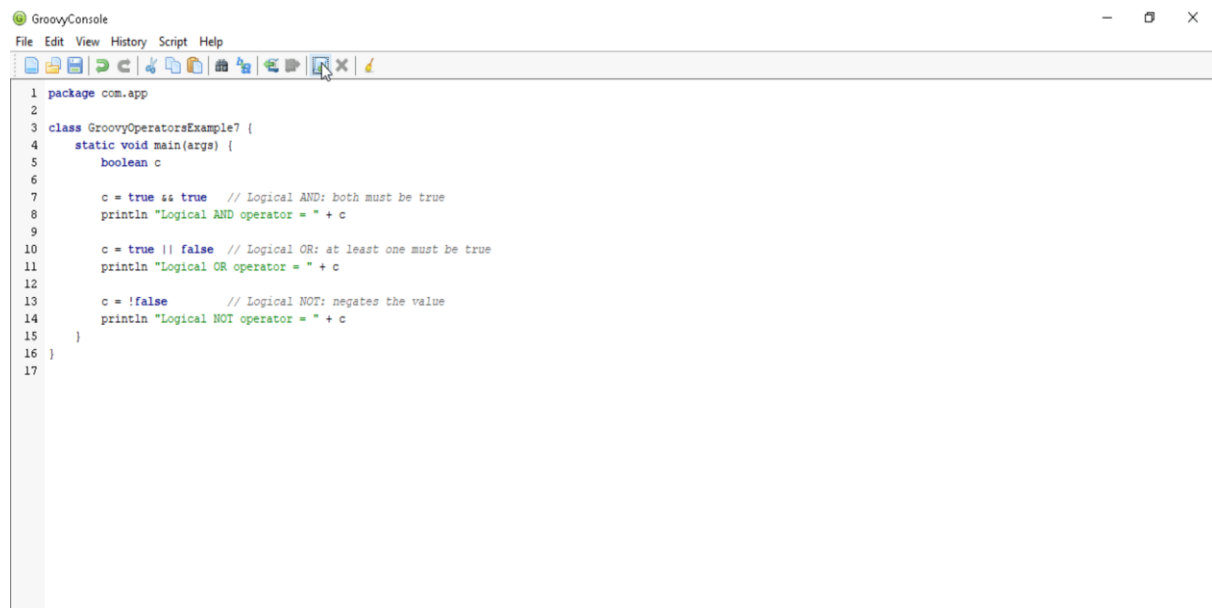


```
1 package com.app
2
3 boolean c
4
5 println "a = 10"
6 println "b = 12"
7
8 c = a == b // Equals operator
9 println "Relational Operator equals [c = a == b] ----> " + c
10
11 c = a != b // Not equals operator
12 println "Relational Operator different [c = a != b] ----> " + c
13
14 c = a < b // Less than operator
15 println "Relational Operator less than [c = a < b] ----> " + c
16
17 c = a <= b // Less than or equal to operator
18 println "Relational Operator less than or equal to [c = a <= b] ----> " + c
19
20 c = a > b // Greater than operator
21 println "Relational Operator greater than [c = a > b] ----> " + c
22
23 c = a >= b // Greater than or equal to operator
24 println "Relational Operator greater than or equal to [c = a >= b] ----> " + c
25 }
26
27 a = 10
28 b = 12
29 Relational Operator equals [c = a == b] ----> false
30 Relational Operator different [c = a != b] ----> true
31 Relational Operator less than [c = a < b] ----> true
32 Relational Operator less than or equal to [c = a <= b] ----> true
33 Relational Operator greater than [c = a > b] ----> false
34 Relational Operator greater than or equal to [c = a >= b] ----> false
35
36 Execution complete. Result was null. Elapsed time: 38ms.
```

Logical operators

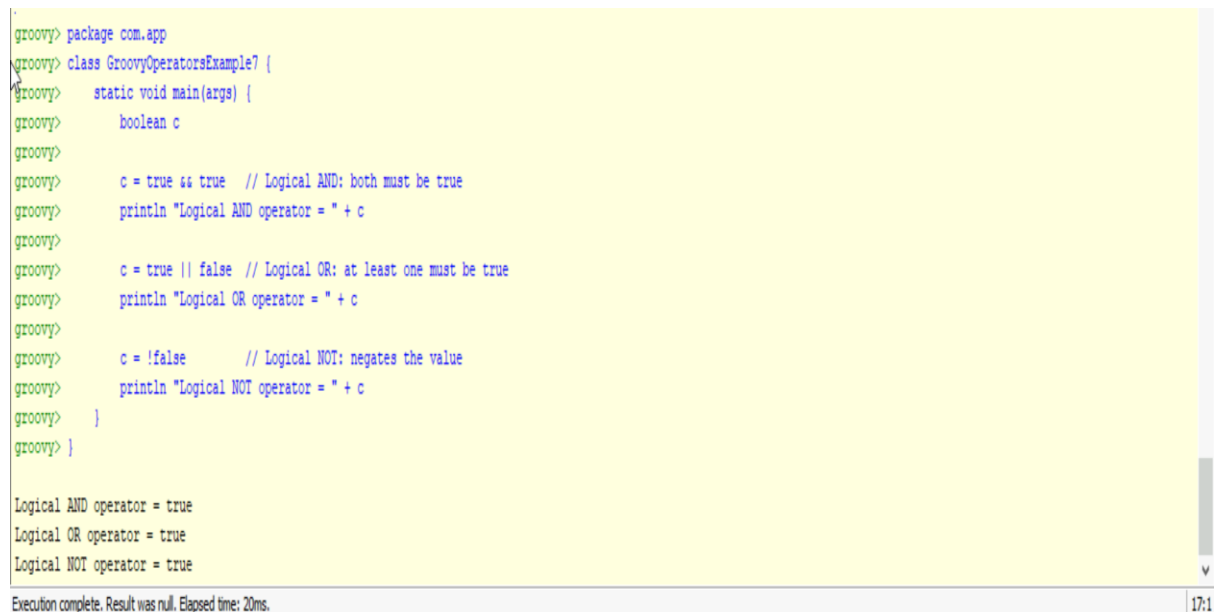
In groovy, there are 3 logical operators for Boolean expression, and these operators are AND(&&), OR(||) and NOT(!)

Example 7:



```
1 package com.app
2
3 class GroovyOperatorsExample7 {
4     static void main(args) {
5         boolean c
6
7         c = true && true // Logical AND: both must be true
8         println "Logical AND operator = " + c
9
10        c = true || false // Logical OR: at least one must be true
11        println "Logical OR operator = " + c
12
13        c = !false // Logical NOT: negates the value
14        println "Logical NOT operator = " + c
15    }
16 }
17
```

Output:

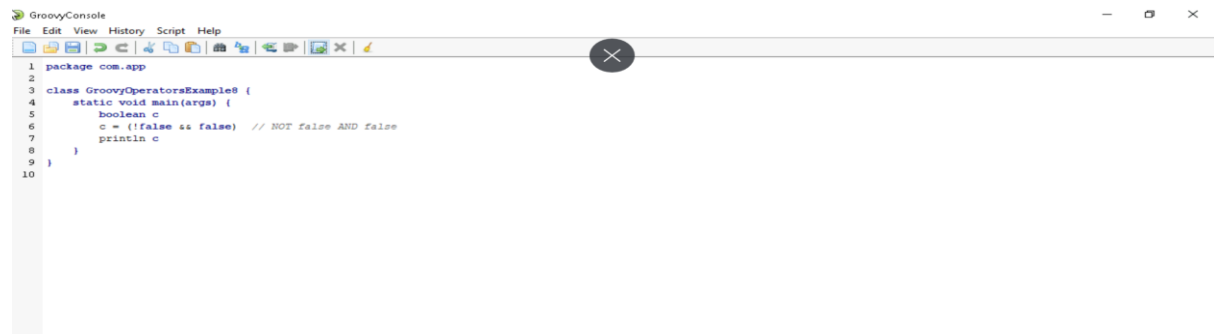


```
groovy> package com.app
groovy> class GroovyOperatorsExample7 {
groovy>     static void main(args) {
groovy>         boolean c
groovy>
groovy>         c = true && true // Logical AND: both must be true
groovy>         println "Logical AND operator = " + c
groovy>
groovy>         c = true || false // Logical OR: at least one must be true
groovy>         println "Logical OR operator = " + c
groovy>
groovy>         c = !false // Logical NOT: negates the value
groovy>         println "Logical NOT operator = " + c
groovy>     }
groovy> }

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

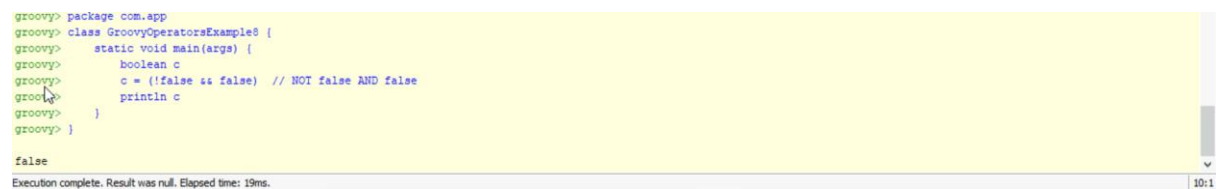
Execution complete. Result was null. Elapsed time: 20ms. 17:1
```

Example 8:



```
1 package com.app
2
3 class GroovyOperatorsExample8 {
4     static void main(args) {
5         boolean c
6         c = (!false && false) // NOT false AND false
7         println c
8     }
9 }
10
```

Output:



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

false
Execution complete. Result was null. Elapsed time: 19ms. 10:1
```

Example 9:



```
1 package com.app
2
3 class GroovyOperatorsExample1 {
4     static void main(args) {
5         boolean c
6         c = true || true && false // OR and AND operators
7         println c
8     }
9 }
10
```

Output:



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

true
Execution complete. Result was null. Elapsed time: 14ms. 10:1
```

Bitwise operators

In groovy, Bitwise operators are used for operating on binary digits or bits of an integer.

Example 10:

```
GroovyConsole
File Edit View History Script Help
[Icons]

1 package com.app
2
3 class GroovyOperatorsExample10 {
4     static void main(args) {
5         int a = 0b00101111 // Binary representation
6         println "a = 0b00101111 ----> " + a
7
8         int b = 0b000010101 // Binary representation
9         println "b = 0b000010101 ----> " + b
10
11        println "(a & a) ----> " + (a & a) // Bitwise AND (AND with itself)
12        println "(a & b) ----> " + (a & b) // Bitwise AND between a and b
13
14        println "(a | a) ----> " + (a | a) // Bitwise OR (OR with itself)
15        println "(a | b) ----> " + (a | b) // Bitwise OR between a and b
16
17        int c = 0b11111111 // Another binary value
18        println "c = 0b11111111"
19
20        println "((a ^ a) & c) ----> " + ((a ^ a) & c) // Bitwise XOR with AND
21        println "((a ^ b) & c) ----> " + ((a ^ b) & c) // Bitwise XOR between a and b with AND
22
23        println "((-a) & c) ----> " + ((-a) & c) // Bitwise NOT a with AND
24    }
25 }
26
```

Output:

```
groovy> static void main(args) {
groovy>     int a = 0b00101111 // Binary representation
groovy>     println "a = 0b00101111 ----> " + a
groovy>
groovy>     int b = 0b000010101 // Binary representation
groovy>     println "b = 0b000010101 ----> " + b
groovy>
groovy>     println "(a & a) ----> " + (a & a) // Bitwise AND (AND with itself)
groovy>     println "(a & b) ----> " + (a & b) // Bitwise AND between a and b
groovy>
groovy>     println "(a | a) ----> " + (a | a) // Bitwise OR (OR with itself)
groovy>     println "(a | b) ----> " + (a | b) // Bitwise OR between a and b
groovy>
groovy>     int c = 0b11111111 // Another binary value
groovy>     println "c = 0b11111111"
groovy>
groovy>     println "((a ^ a) & c) ----> " + ((a ^ a) & c) // Bitwise XOR with AND
groovy>     println "((a ^ b) & c) ----> " + ((a ^ b) & c) // Bitwise XOR between a and b with AND
groovy>
groovy>     println "((-a) & c) ----> " + ((-a) & c) // Bitwise NOT a with AND
groovy> }
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

Execution complete. Result was null. Elapsed time: 43ms. 26:1
```

Example 11:



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

Output:



```
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>
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: 25ms. 15:1
```

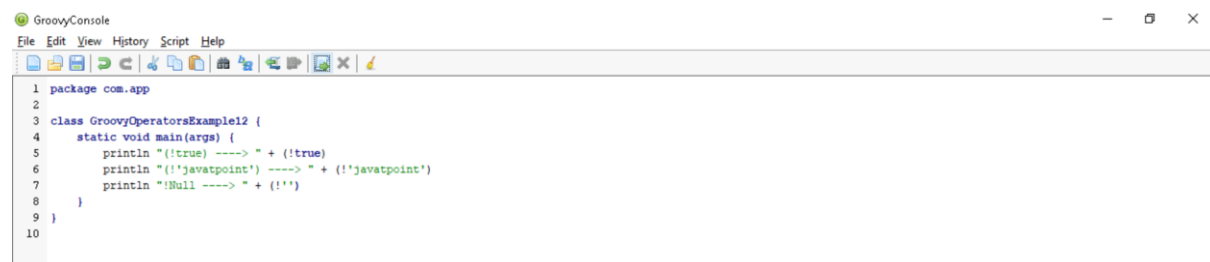
Conditional operators

In groovy, there are three types of conditional operators they are as follow:

- **Not operator**

In groovy, "not" operator is used invert the result of the Boolean expression.

Example 12:



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

Output:



```
groovy> package com.app
groovy> class GroovyOperatorsExample12 {
groovy>     static void main(args) {
groovy>         println "(!true) ----> " + (!true)
groovy>         println "(!'javatpoint') ----> " + (!'javatpoint')
groovy>         println "(!Null) ----> " + (!'')
groovy>     }
groovy> }
groovy>

(!true) ----> false
(!'javatpoint') ----> false
(!Null) ----> true
Execution complete. Result was null. Elapsed time: 18ms. 10:1
```

Example 13:



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

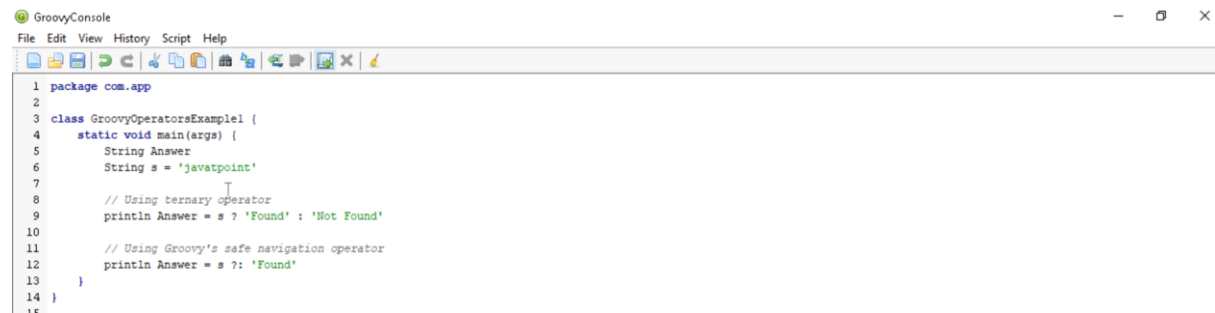
Output:

```
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
groovy>     }
groovy> }

Found

Execution complete. Result was null. Elapsed time: 29ms. 11:1
```

Example 14:



The screenshot shows the GroovyConsole application window. The title bar is 'GroovyConsole'. The menu bar includes 'File', 'Edit', 'View', 'History', 'Script', and 'Help'. The toolbar contains icons for file operations and execution. The code editor displays the following Groovy code:

```
1 package com.app
2
3 class GroovyOperatorsExample1 {
4     static void main(args) {
5         String Answer
6         String s = 'javatpoint'
7
8         // Using ternary operator
9         println Answer = s ? 'Found' : 'Not Found'
10
11        // Using Groovy's safe navigation operator
12        println Answer = s ?. 'Found'
13    }
14 }
```

Output:

```
groovy> package com.app
groovy> class GroovyOperatorsExample1 {
groovy>     static void main(args) {
groovy>         String s = 'javatpoint'
groovy>         // Using ternary operator
groovy>         String Answer = s ? 'Found' : 'Not Found'
groovy>         println Answer // Prints "Found"
groovy>         // Using Groovy's safe navigation operator
groovy>         Answer = s ?. 'Found'
groovy>         println Answer // Prints "javatpoint"
groovy>     }
groovy> }

Found
javatpoint

Execution complete. Result was null. Elapsed time: 20ms. 16:1
```