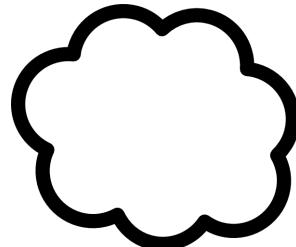
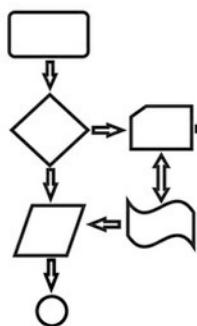


San Jose State University

CMPE 281
Cloud Technologies

Quick Introduction to Groovy



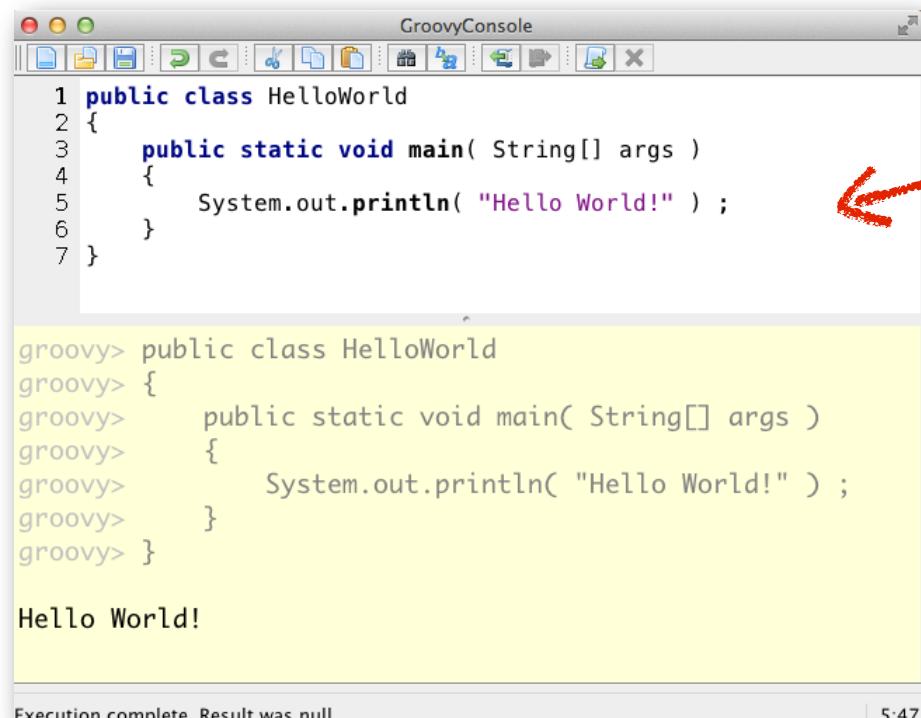
The screenshot shows a terminal window with several tabs open. The current tab, 'ZREADME.md — sjsu', displays a Groovy script for SDK setup and references. The script includes URLs for Groovy download and documentation, as well as commands for installing the Groovy SDK via sdkman. Below the script, there are sections for '# References' with links to the Groovy website and a Groovy console appspot page.

```
1
2
3 # Groovy SDK Setup
4
5     http://www.groovy-lang.org/download.html
6
7     curl -s get.sdkman.io | bash
8     source "$HOME/.sdkman/bin/sdkman-init.sh"
9     sdk install groovy
10    groovy -version
11
12 # References
13
14     http://www.groovy-lang.org/
15     https://groovyconsole.appspot.com/
16
17
```

The left sidebar shows a file tree with a focus on a 'groovy' directory containing numerous Groovy script files (EX01 through EX26, plus groovyx.jar and run.sh) and a 'gumball' directory with versions v1, v2, and v3, along with 'hellobooks' and 'helloworld' directories.

Groovy

Hello World



A screenshot of a GroovyConsole window titled "GroovyConsole". The window has a toolbar at the top with various icons. The code area contains the following Java code:

```
1 public class HelloWorld
2 {
3     public static void main( String[] args )
4     {
5         System.out.println( "Hello World!" ) ;
6     }
7 }
```

Below the code, the console area shows the output of running the code:

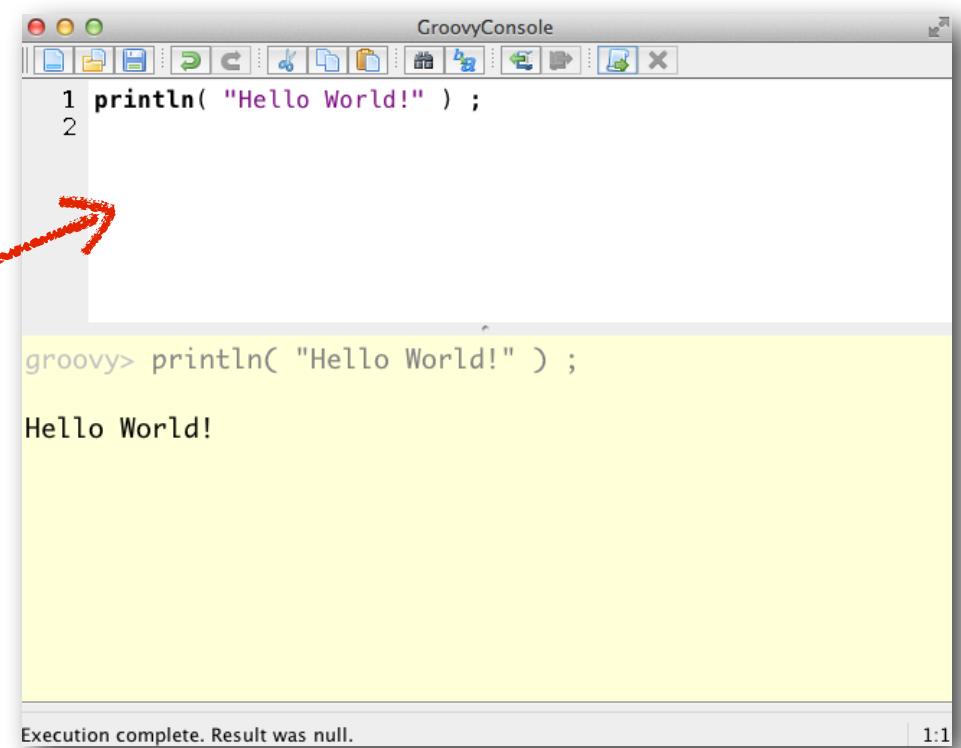
```
groovy> public class HelloWorld
groovy> {
groovy>     public static void main( String[] args )
groovy>     {
groovy>         System.out.println( "Hello World!" ) ;
groovy>     }
groovy> }
```

The output "Hello World!" is displayed in bold black text. At the bottom of the console, the message "Execution complete. Result was null." is shown.

Groovy is "Java"



Yes, Groovy is "Java", but
without the ceremony!



A screenshot of a GroovyConsole window titled "GroovyConsole". The window has a toolbar at the top with various icons. The code area contains the following Groovy code:

```
1 println( "Hello World!" ) ;
```

Below the code, the console area shows the output of running the code:

```
groovy> println( "Hello World!" ) ;
```

The output "Hello World!" is displayed in bold black text. At the bottom of the console, the message "Execution complete. Result was null." is shown.

Execution complete. Result was null.

1:1

EX01_helloworld.groovy — sjsu

```
1 |  
2 public class HelloWorld  
3 {  
4     public static void main( String[] args )  
5     {  
6         System.out.println( "Hello World!" ) ;  
7     }  
8 }  
9  
10
```

281

- ▶ aws
- ▶ bios
- ▶ docker
- ▼ grails
- ▶ groovy
 - EX01_helloworld.groovy
 - EX02_helloworld.groovy
 - EX03_dynamictypes.groovy
 - EX04_operators.groovy
 - EX05_stringliterals.groovy
 - EX06_groovymaps.groovy
 - EX07_groovyarrays.groovy
 - EX08_closure1.groovy
 - EX09_closure2.groovy
 - EX10_closure3.groovy
 - EX10_closure4.groovy
 - EX11_method1.groovy
 - EX12_method2.groovy
 - EX13_method3.groovy
 - EX14_class1.groovy
 - EX16_xmlsample1.groovy
 - EX17_xmlsample2.groovy
 - EX18_jsonsample1.groovy
 - EX19_jsonsample2.groovy
 - EX20_jsonsample3.groovy
 - EX20_jsonsample3.json
 - EX21_jsonsample4.groovy
 - EX22_jsonsample5.groovy
 - EX23_httpbuilder1.groovy
 - EX24_httpbuilder2.groovy
 - EX25_restsample1.groovy
 - EX26_restsample2.groovy
 - groovyx.jar
 - run.sh
 - ZREADME.md
- ▶ gumball-v1
- ▶ gumball-v2
- ▶ gumball-v3
- ▶ hellobooks

Line 1, Column 1 Tab Size: 4 Groovy

The screenshot shows a code editor window with a dark theme. On the left is a file tree showing a directory structure under '281'. The 'groovy' folder contains several Groovy script files, with 'EX02_helloworld.groovy' selected and shown in the main editor area. The code in the editor is:

```
1 System.out.println( "Hello World!" ) ;
```

The status bar at the bottom indicates 'Line 1, Column 1' and includes tabs for 'Tab Size: 4' and 'Groovy'.

The screenshot shows a code editor window with a dark theme. On the left is a file tree for a directory named '281'. The 'groovy' folder contains several Groovy script files, with 'EX03_dynamictypes.groovy' selected and shown in the main editor area. The code in the editor is:

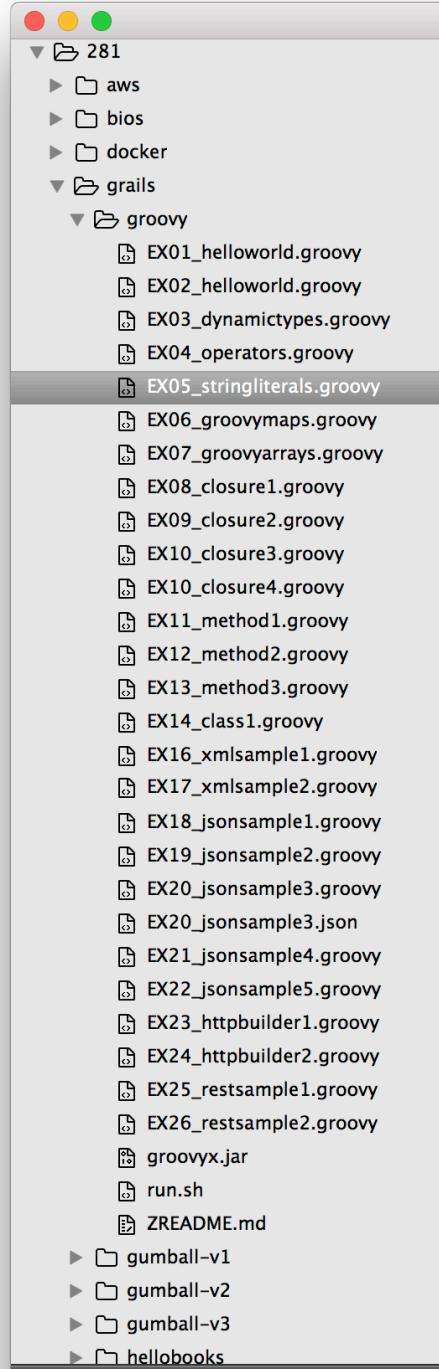
```
1 int age=25
2 def number=age
3 message = "Hello World"
4
5 println age.getClass().getName()
6 println number.getClass().getName()
7 println message.getClass().getName()
8
9
10 number=message
11 println number.getClass().getName()
12
13
14 5.dump()
15 5.metaClass.methods
```

The status bar at the bottom indicates 'Line 1, Column 1' and includes tabs for 'Tab Size: 4' and 'Groovy'.

The screenshot shows a code editor window with a dark theme. On the left is a file tree for a project named '281'. The 'groovy' folder contains numerous Groovy script files, with 'EX04_operators.groovy' currently selected and highlighted in blue. The main editor area displays the contents of this file:

```
1 println 5<3
2 println 5.compareTo(3) < 0
3
4 println 5>3
5 println 5.compareTo(3) > 0
6
7 println 5==3
8 println 5.equals(3)
9
10 println 5!=3
11 println !(5.equals(3))
12
13
14
15
16
```

The status bar at the bottom indicates 'Line 1, Column 1' on the far left, 'Tab Size: 4' in the middle, and 'Groovy' on the far right.



EX05_stringliterals.groovy — sjsu

```
1 | 
2 def age=25
3 |
4 println 'My age is ${age}'
5 println "My age is ${age}"
6 println "My age is \$${age}"
7 |
8 |
9 s1="Hello"
10 s2="World"
11 s3=123
12 printf (" %-2s | %-2s| %5d\n", s1, s2, s3)
13 |
14 |
15 a='Hello'
16 b='World'
17 println a.concat(' ').concat(b)
18 println a + ' ' + b
19 println "${a} ${b}"
20 printf "%s %s\n", a , b
```

The screenshot shows a Java IDE interface with a dark theme. On the left is a file tree under a folder named '281'. The 'groovy' folder contains several files, with 'EX06_groovymaps.groovy' being the active tab. The code in the editor window is:

```
1 def map = [ 'name' : 'James', 'location' : 'London' ]
2 println map.size() == 2
3 println map.get('name') == 'James'
4 println map['name'] == 'James'
5
6
7
8 def map = [ 'name':'James', 'location':'London' , 'a' : ['b':'value'] ]
9 println map.size()
10 println map['location']
11 map['job'] = 'Coder'
12 println map.size()
13 println map
```

The status bar at the bottom indicates 'Line 1, Column 1' and 'Tab Size: 4'. The title bar says 'EX06_groovymaps.groovy — sjsu'.

The screenshot shows a code editor window with a dark theme. On the left is a file tree for a project named '281'. The 'groovy' folder contains several Groovy script files, with 'EX07_groovyarrays.groovy' selected and highlighted in grey. The main editor area displays the contents of this script.

```
1 def list = [1, 2, 3]
2 for (i in list) { println i }
3
4 println list[0]
5 println list[1]
6 println list[2]
7
8
9
10 def list = ["hello" ,2 , "goodbye", [1,2,3]]
11
12 for ( item in list )
13 {
14     println item.getClass().getName()
15     println item ;
16     if ( item.getClass().getName().equals("java.util.ArrayList") )
17     {
18         //println item.size() ;
19         for ( innerItem in item )
20             println innerItem ;
21     }
22 }
```

The status bar at the bottom indicates 'Line 1, Column 1' on the far left, 'Tab Size: 4' in the middle, and 'Groovy' on the far right.

The screenshot shows a code editor window with a dark theme. On the left is a file tree with a sidebar containing project and file navigation. The main area displays Groovy code for closures.

```
EX08_closure1.groovy — sjsu
2
3 /*
4 ** http://groovy-lang.org/closures.html
5 */
6
7
8 def closure = { param -> println( "hello ${param}" ) }
9 closure.call( "world!" )
10
11 closure = { greeting, name -> println( greeting + name) }
12 closure.call( "hello ", "world!" )
13
14
15 def sayHello(name)
16 {
17     println "Hello There ${name}"
18     return "Fun!"
19 }
20
21 println sayHello('John')
22
23 def closure = { param -> println( "hello ${param}" ) }
24 closure.call( "world!" )
25
26 def displayItem = { x -> println x+1 }
27 [1, 2, 3, 4, 5].each( displayItem )
28
29 printMe( it )
30 {
31     println it
32 }
33 list = [1, 2, 3, 4, 5]
34 for ( item in list )
35     printMe(item)
36
37 [1, 2, 3, 4, 5].each ( {println it} )
38
```

The screenshot shows a Java IDE interface with a dark theme. On the left is a file tree with several project modules and source files. The main area contains a code editor with tab bars labeled "EX08_closure1.groovy" and "EX09_closure2.groovy". The code in "EX09_closure2.groovy" is as follows:

```
1 |
2
3 /*
4 ** http://groovy-lang.org/closures.html
5 */
6
7
8 def closure = { println "hello " + it }
9 closure.call( "world!" )
10
11 [1, 2, 3].each ( { item -> println "${item}" } )
12
13 [ "k1":"v1", "k2":"v2" ].each ( {key, value -> println key + "=" + value
   }
)
14
15
```

EX10_closure3.groovy — sjsu

281

- aws
- bios
- docker
- grails
 - groovy
 - EX01_helloworld.groovy
 - EX02_helloworld.groovy
 - EX03_dynamictypes.groovy
 - EX04_operators.groovy
 - EX05_stringliterals.groovy
 - EX06_groovymaps.groovy
 - EX07_groovyarrays.groovy
 - EX08_closure1.groovy
 - EX09_closure2.groovy
 - EX10_closure3.groovy
 - EX10_closure4.groovy
 - EX11_method1.groovy
 - EX12_method2.groovy
 - EX13_method3.groovy
 - EX14_class1.groovy
 - EX16_xmlsample1.groovy
 - EX17_xmlsample2.groovy
 - EX18_jsonsample1.groovy
 - EX19_jsonsample2.groovy
 - EX20_jsonsample3.groovy
 - EX20_jsonsample3.json
 - EX21_jsonsample4.groovy
 - EX22_jsonsample5.groovy
 - EX23_httpbuilder1.groovy
 - EX24_httpbuilder2.groovy
 - EX25_restsample1.groovy
 - EX26_restsample2.groovy
 - groovyx.jar
 - run.sh
 - ZREADME.md
- gumball-v1
- gumball-v2
- gumball-v3
- helloworlds
- hellograils

EX08_closure1.groovy EX10_closure3.groovy

```
1 |
2
3 /*
4 **  http://groovy-lang.org/closures.html
5 */
6
7
8 greeting='Hello'
9 def clos={ param -> println "${greeting} ${param}" }
10 clos.call( 'World' )
11
12 greeting='Welcome'
13 clos.call( 'World' )
14
15 def demo(c) {
16     def greeting='Bonjour'
17     c.call( 'Ken' )
18 }
19 demo(clos)
20
21 def outer() {
22     def greeting='Goodbye'
23     return { p -> println "${greeting} ${p}" }
24 }
25 greeting='Welcome'
26 clos=outer()
27 clos.call( 'Ken' )
```

Line 1, Column 1

Tab Size: 4

Groovy

The screenshot shows a code editor window with a dark theme. On the left is a file tree for a project named '281'. The 'groovy' directory contains several files, including 'EX10_closure4.groovy', which is currently selected and shown in the main editor area. The editor tabs at the top show 'EX08_closure1.groovy' and 'EX10_closure4.groovy'. The code in the editor is:

```
1 |
2 /*
3 ** http://groovy-lang.org/closures.html
4 */
5
6 greeting = 'Hello'
7
8 def demo() {
9     greeting = "Bonjour"
10    def closure = { p -> print "${greeting} ${p}\n" }
11    return closure
12 }
13
14 c = demo()
15 c.call( 'Ken' )
```

The status bar at the bottom indicates 'Line 1, Column 1' and includes tabs for 'Tab Size: 4' and 'Groovy'.

The screenshot shows a code editor window with a dark theme. On the left is a file tree with several project folders like 'aws', 'bios', 'docker', and 'grails'. Inside 'grails/groovy', there are many files listed, including 'EX01_helloworld.groovy' through 'EX10_closure4.groovy', 'EX11_method1.groovy' (which is selected), 'EX12_method2.groovy', 'EX13_method3.groovy', 'EX14_class1.groovy', 'EX16_xmlsample1.groovy', 'EX17_xmlsample2.groovy', 'EX18_jsonsample1.groovy', 'EX19_jsonsample2.groovy', 'EX20_jsonsample3.groovy', 'EX20_jsonsample3.json', 'EX21_jsonsample4.groovy', 'EX22_jsonsample5.groovy', 'EX23_httpbuilder1.groovy', 'EX24_httpbuilder2.groovy', 'EX25_restsample1.groovy', 'EX26_restsample2.groovy', 'groovyx.jar', 'run.sh', and 'ZREADME.md'. Below these are folder icons for 'gumball-v1', 'gumball-v2', 'gumball-v3', 'helloworld', and 'hellograils'. The main editor area has tabs for 'EX08_closure1.groovy' and 'EX11_method1.groovy'. The code in 'EX11_method1.groovy' is:

```
1 |  
2 def greetings() {  
3     print 'Hello'  
4     print ' and '  
5     println 'welcome'  
6 }  
7  
8 greetings()  
9  
10
```

The screenshot shows a Java IDE interface with a dark theme. On the left is a file tree titled '281' containing various Groovy script files and project folders like 'aws', 'bios', 'docker', 'grails', and 'groovy'. The main workspace contains two tabs: 'EX08_closure1.groovy' and 'EX12_method2.groovy'. The 'EX12_method2.groovy' tab is active, displaying the following Groovy code:

```
1
2
3 def greetings( name ) {
4     println "Hello and welcome, ${name}"
5 }
6
7
8 greetings( 'John' )
9
10
```

The code defines a method 'greetings' that prints a welcome message. It is then called with the argument 'John'. The code editor shows syntax highlighting for keywords like 'def' and 'println', and variables like 'name'.

The screenshot shows a Java IDE interface with a dark theme. On the left is a file tree titled '281' containing several Groovy script files and some directory structures like 'aws', 'bios', 'docker', and 'grails'. The main workspace contains two tabs: 'EX08_closure1.groovy' and 'EX13_method3.groovy'. The 'EX13_method3.groovy' tab is active and displays the following Groovy code:

```
1 def greetings( salutation, name='Ken' ) {  
2     println "${salutation} ${name}"  
3 }  
4 greetings( 'Hello', 'John' )  
5 greetings( 'Welcome' )
```

The status bar at the bottom indicates 'Line 1, Column 1' and includes tabs for 'Tab Size: 4' and 'Groovy'.

```
EX14_class1.groovy — sjsu
class Person {
    String name
    Integer age
    Person( name, age ) {
        this.name = name
        this.age = age
    }
}
def person1 = new Person( 'Marie', 1 )
def person2 = ['Marie', 2] as Person
Person person3 = [ 'Marie', 3]
```

The screenshot shows a code editor window with a dark theme. On the left is a file tree with a sidebar containing project and file navigation. The main area displays Groovy code. The code defines a `Person` class with `name` and `age` properties, both of which are dynamically typed. It includes a constructor that takes `name` and `age` parameters and assigns them to the class's properties using the `this` keyword. Below the class definition, there are three `def` statements: `person1` is created using the `new` keyword and passed two arguments; `person2` is created using a closure and converted to a `Person` object using the `as` keyword; and `person3` is created using a closure directly. The code editor interface includes tabs for other files like `EX08_closure1.groovy` and `EX16_xmlsample1.groovy`, and status bars at the bottom indicating the current line and column (Line 1, Column 1) and the language (Groovy). A tab size of 4 is also visible.

The screenshot shows a Java-based IDE interface with a dark theme. On the left is a file tree titled '281' containing various Groovy script files and some directory entries like 'aws', 'bios', 'docker', and 'grails'. The main workspace contains two tabs: 'EX08_closure1.groovy' and 'EX18_jsonsample1.groovy'. The 'EX18_jsonsample1.groovy' tab is active and displays the following Groovy code:

```
1 |  
2 import groovy.json.JsonBuilder  
3 |  
4 def builder = new JsonBuilder()  
5 builder.weeks {  
6     capacity '8'  
7     tasks(  
8         [ {  
9             done '0'  
10            total '4'  
11            title 'build web service'  
12        }, {  
13            done '0'  
14            total '1'  
15            title 'build web service client'  
16        }]  
17    )  
18}  
19|  
20 //println builder.toString()  
21 println builder.toPrettyString()  
22|  
23|  
24|
```

The screenshot shows a Java-based IDE interface with a dark theme. On the left is a file tree titled '281' containing various Groovy script files. The main workspace contains two tabs: 'EX08_closure1.groovy' and 'EX19_jsonsample2.groovy'. The 'EX19_jsonsample2.groovy' tab is active, displaying the following Groovy code:

```
1 import groovy.json.JsonBuilder
2
3 def builder = new JsonBuilder()
4
5 builder {
6     invoices(1..3) { day ->
7         invoice(date: "2015-01-0$day") {
8             item(count: day) {
9                 product(name: 'ULC', dollar: 1499)
10            }
11        }
12    }
13 }
14
15 println builder.toString()
16 //println builder.toPrettyString()
17
18
```

The code uses a closure-based builder pattern to generate a JSON string. It defines a `JsonBuilder` object and uses it to build an array of `invoices`. Each invoice has a date of '2015-01-0\$day' and an item with a count of `day` and a product named 'ULC' costing 1499 dollars.

The screenshot shows a Java IDE interface with a file tree on the left and two code tabs on the right.

File Tree:

- 281
 - aws
 - bios
 - docker
 - grails
 - groovy
 - EX01_helloworld.groovy
 - EX02_helloworld.groovy
 - EX03_dynamictypes.groovy
 - EX04_operators.groovy
 - EX05_stringliterals.groovy
 - EX06_groovymaps.groovy
 - EX07_groovyarrays.groovy
 - EX08_closure1.groovy
 - EX09_closure2.groovy
 - EX10_closure3.groovy
 - EX10_closure4.groovy
 - EX11_method1.groovy
 - EX12_method2.groovy
 - EX13_method3.groovy
 - EX14_class1.groovy
 - EX16_xmlsample1.groovy
 - EX17_xmlsample2.groovy
 - EX18_jsonsample1.groovy
 - EX19_jsonsample2.groovy
 - EX20_jsonsample3.groovy
 - EX20_jsonsample3.json
 - EX21_jsonsample4.groovy
 - EX22_jsonsample5.groovy**
 - EX23_httpbuilder1.groovy
 - EX24_httpbuilder2.groovy
 - EX25_restsample1.groovy
 - EX26_restsample2.groovy
 - groovyx.jar
 - run.sh
 - ZREADME.md
 - gumball-v1
 - gumball-v2
 - gumball-v3
 - helloworlds
 - hellograils

Code Tabs:

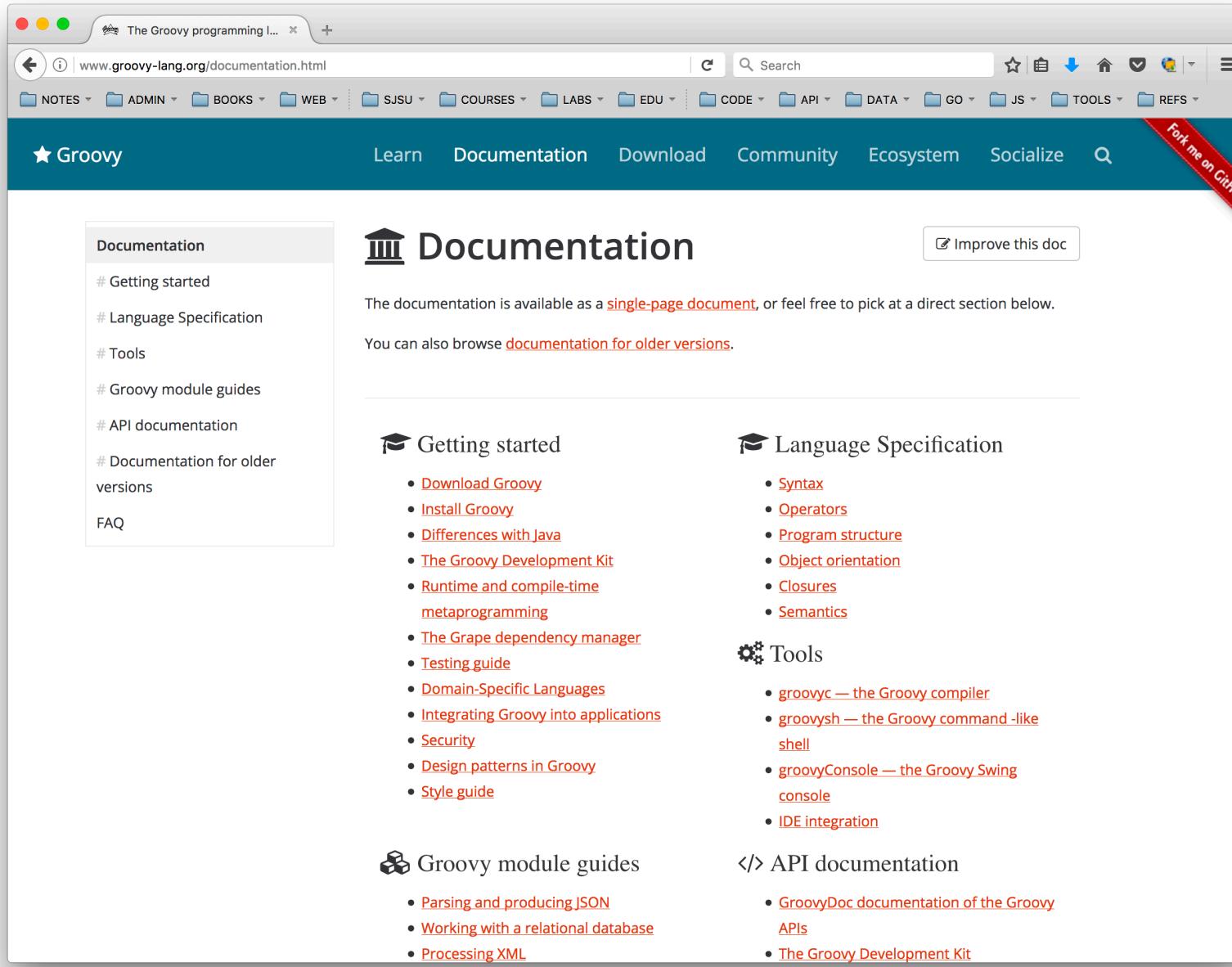
 - EX08_closure1.groovy
 - EX22_jsonsample5.groovy**

```
1 /*
2
3 http://www.groovy-tutorial.org/basic-json/
4
5 BUILDING JSON
6
7 */
8
9 import groovy.json.JsonBuilder
10
11 def json = new JsonBuilder()
12 def staffListExport = json.staff {
13     '1234' {
14         id 1234
15         name 'Fred Nurk'
16         position (
17             department: 'Accounts',
18             role: 'Manager'
19         )
20     }
21 }
22
23 println json.toPrettyString()
24
25
26
```

Documentation

GDK

<http://www.groovy-lang.org/documentation.html>



The screenshot shows a web browser displaying the Groovy documentation homepage. The URL in the address bar is <http://www.groovy-lang.org/documentation.html>. The page has a dark blue header with the Groovy logo and navigation links for Learn, Documentation, Download, Community, Ecosystem, Socialize, and a search icon. A red ribbon in the top right corner says "Fork me on GitHub". The main content area features a large title "Documentation" with a graduation cap icon. Below it, a message says the documentation is available as a [single-page document](#), or you can browse direct sections like "Getting started", "Language Specification", "Tools", "Groovy module guides", "API documentation", and "FAQ". On the left, a sidebar lists "Documentation" sections: Getting started, Language Specification, Tools, Groovy module guides, API documentation, and FAQ. The "Getting started" section contains links to download, install, differences from Java, development kit, runtime metaprogramming, Grape dependency manager, testing guide, domain-specific languages, integrating into applications, security, design patterns, and style guide. The "Language Specification" section lists syntax, operators, program structure, object orientation, closures, and semantics. The "Tools" section lists groovyc, groovysh, groovyConsole, and IDE integration. The "Groovy module guides" section lists JSON parsing, relational databases, and XML processing. The "API documentation" section lists GroovyDoc documentation, APIs, and the Groovy Development Kit.

The documentation is available as a [single-page document](#), or feel free to pick at a direct section below.

You can also browse [documentation for older versions](#).

Getting started

- [Download Groovy](#)
- [Install Groovy](#)
- [Differences with Java](#)
- [The Groovy Development Kit](#)
- [Runtime and compile-time metaprogramming](#)
- [The Grape dependency manager](#)
- [Testing guide](#)
- [Domain-Specific Languages](#)
- [Integrating Groovy into applications](#)
- [Security](#)
- [Design patterns in Groovy](#)
- [Style guide](#)

Groovy module guides

- [Parsing and producing JSON](#)
- [Working with a relational database](#)
- [Processing XML](#)

Language Specification

- [Syntax](#)
- [Operators](#)
- [Program structure](#)
- [Object orientation](#)
- [Closures](#)
- [Semantics](#)

Tools

- [groovyc — the Groovy compiler](#)
- [groovysh — the Groovy command-line shell](#)
- [groovyConsole — the Groovy Swing console](#)
- [IDE integration](#)

API documentation

- [GroovyDoc documentation of the Groovy APIs](#)
- [The Groovy Development Kit](#)

Some Strings Examples

```

'Hello'.compareToIgnoreCase('hello')           // 0
'Hello'.concat('world')                      // Hello world
'Hello'.endsWith('lo')                       // true
'Hello'.equalsIgnoreCase('hello')             // true
'Hello'.indexOf('lo')                        // 3
'Hello world'.indexOf('o', 6)                // 7
'Hello'.matches('Hello')                     // true
'Hello'.matches('He')                        // false
'Hello'.replaceAll('l', 'L')                 // HELLo
'Hello world'.split('l')                     // 'He', 'o wor', 'd'
'Hello'.substring(1)                         // ello
'Hello'.substring(1, 4)                      // ell
'Hello'.toUpperCase()                        // HELLO
def message = 'Hello'
message.center(11)                           //□□□Hello□□□
message.center(3)                           // Hello
message.center(11, '#')                     // ###Hello###
message.eachMatch('.') { ch ->
    println ch }
message.getValueAt(0)
message.getValueAt(0..<3)
message.getValueAt([0, 2, 4])
message.leftShift('world')
message << 'world'
message.minus('ell')
message-'ell'
message.padLeft(4)
message.padLeft(11)
message.padLeft(11, '#')
message.padRight(4)
message.padRight(11)
message.padRight(11, '#')
message.plus('world')
message+'world'
message.replaceAll('[a-z]') { ch ->
    ch.toUpperCase() }
message.reverse()                           // olleH
message.toList()                            // ['H', 'e', 'l', 'l', 'o']
def message='Hello world'
message.tokenize()                          // ['Hello', 'world']
message.tokenize('l')                      // ['He', 'o wor', 'd']

def greeting='Hello world'
'Hello'+ 'world'                         // Hello world      concatenate
'Hello'*3                                // HelloHelloHello repeat
greeting-'o world'                        // Hell
greeting.size()                           // 11
greeting.length()                         // 11
greeting.count('o')                        // 2
greeting.contains('ell')                  // true

```

In a regular expression, two special *positional characters* are used to denote the beginning and end of a line: caret (^) and dollar sign (\$):

```

def rhyme='Humpty Dumpty sat on a wall'
rhyme=~ '^Humpty'                         // true
rhyme=~ 'wall$'                           // true

```

Regular expressions can also include *quantifiers*. The plus sign (+) represents one or more times, applied to the preceding element of the expression. The asterisk (*) is used to represent zero or more occurrences. The question mark (?) denotes zero or once. The metacharacter { and } is used to match a specific number of instances of the preceding character. The following all yield true:

```

'aaaaab' =~ 'a+b'
'b' =~ 'a*b'
'aaacd' =~ 'a*c?d'
'aaad' =~ 'a*c?d'
'aaaaab' =~ 'a{5}b'
!('aab' =~ 'a{5}b')

```

Collections

Example	Description
[11, 12, 13, 14]	A list of integer values
['Ken', 'John', 'Andrew']	A list of Strings
[1, 2, [3, 4], 5]	A nested list
['Ken', 21, 1.69]	A heterogeneous list of object references
[]	An empty list

```
def numbers = [11, 12, 13, 14]      // list with four items
numbers [0]                         // 11
numbers [3]                         // 14

numbers [-1]                        // 14
numbers [-2]                        // 13
```

```
[11, 12, 13, 14][2]    // 13

numbers [0..2]      // [11, 12, 13]
numbers [1..<3]     // [12, 13]

numbers [1]=22       // [11, 22, 13, 14]
numbers [1]=[33, 44]  // [11, [33, 44], 13, 14]
```

```
numbers<<15      // [11, [33, 44], 13, 14, 15]

numbers=[11, 12, 13, 14]    // list with four items
numbers+[15, 16]           // [11, 12, 13, 14, 15, 16]
```

```
numbers=[11, 12, 13, 14]      // list with four items
numbers-[13]                  // [11, 12, 14]
```

TABLE 4.2 List methods

Name	Signature/description
add	boolean add(Object value) Append the new value to the end of this List.
add	void add(int index, Object value) Inserts a new value into this List at the given index position.
addAll	boolean addAll(Collection values) Append the new values on to the end of this List.
contains	boolean contains(Object value) Returns true if this List contains the specified value.
flatten *	List flatten() Flattens this List and returns a new List.
get	Object get(int index) Returns the element at the specified position in this List.

```
[11, 12, 13, 14].add(15)          // [11, 12, 13, 14, 15]
[11, 12, 13, 14].add(2, 15)       // [11, 12, 15, 13, 14]
[11, 12, 13, 14].add([15, 16])    // [11, 12, 13, 14, 15, 16]
[11, 12, 13, 14].get(1)           // 12
[11, 12, 13, 14].isEmpty()        // false
[14, 13, 12, 11].size()          // 4
[11, 12, [13, 14]].flatten()      // [11, 12, 13, 14]
[11, 12, 13, 14].getAt(1)         // 12
[11, 12, 13, 14].getAt(1..2)      // [12, 13]
[11, 12, 13, 14].getAt([2, 3])    // [13, 14]
[11, 12, 13, 14].intersect([13, 14, 15]) // [13, 14]
[11, 12, 13, 14].pop()            // 14
[11, 12, 13, 14].reverse()        // [14, 13, 12, 11]
[14, 13, 12, 11].sort()           // [11, 12, 13, 14]
```

Maps

Example	Description
['Ken' : 'Barclay', 'John' : 'Savage']	Forename/surname collection
[4 : [2], 6 : [3, 2], 12 : [6, 4, 3, 2]]	Integer keys and their list of divisors
[:]	Empty map

```
def names=['Ken' : 'Barclay', 'John' : 'Savage']
def divisors=[4 : [2], 6 : [3, 2], 12 : [6, 4, 3, 2]]
names['Ken']                                // 'Barclay'
names.Ken                                    // 'Barclay'
names['Jessie']                             // null
divisors[6]                                  // [3, 2]
```

```
divisors[6]=[6, 3, 2, 1]                    // [4 : [2], 6 : [6, 3, 2, 1],
                                            // 12 : [6, 4, 3, 2]]
```

```
def careful=[ 1 : 'Ken', '1' : 'Barclay']
careful[1]                                    // Ken
careful['1']                                 // Barclay
```

TABLE 4.4 Map methods

Name	Signature/description
containsKey	boolean containsKey(Object key) Does this Map contain this key?
get	Object get(Object key) Look up the key in this Map and return the corresponding value. If there is no entry in this Map for the key, then return null.
get *	Object get(Object key, Object defaultValue) Look up the key in this Map and return the corresponding value. If there is no entry in this Map for the key, then return the defaultValue.
getAt *	Object getAt(Object key) Support method for the subscript operator.
keySet	Set keySet() Obtain a Set of the keys in this Map.
put	Object put(Object key, Object value) Associates the specified value with the specified key in this Map. If this Map previously contained a mapping for this key, the old value is replaced by the specified value.
putAt *	Object putAt(Object key, Object value) Support method to allow Maps to operate with subscript assignment.
size	int size() Returns the number of key-value mappings in this Map.
values	Collection values() Returns a collection view of the values contained in this Map.

```
def mp=['Ken' : 2745, 'John' : 2746, 'Sally' : 2742]
mp.put('Bob', 2713)                         // [Bob:2713, Ken:2745, Sally:2742, John:2746]
mp.containsKey('Ken')                        // true
mp.get('David', 9999)                        // 9999
mp.get('Sally')                            // 2742
mp.get('Billy')                            // null
mp.keySet()                               // [David, Bob, Ken, Sally, John]
mp.size()                                  // 4
mp['Ken']                                   // 2745
```

Methods

```
def methodName() {  
    // Method code goes here  
}
```

```
def greetings() {  
    print 'Hello'  
    print ' and '  
    println 'welcome'  
}
```

```
greetings()
```

```
def methodName(para1, para2, para3) {  
    // Method code goes here  
}
```

```
def greetings(name) {  
    println "Hello and welcome, ${name}"  
}  
  
greetings('John')
```

Hello and welcome

```
def someMethod(para1, para2=0, para3=0) {  
    // Method code goes here  
}
```

```
def greetings(salutation, name='Ken') {  
    println "${salutation} ${name}"  
}  
  
greetings('Hello', 'John')          // Hello John  
greetings('Welcome')               // Welcome Ken
```

Hello and welcome, John

Hello John
Welcome Ken

Closures - Invocation & Parameters

9.1 CLOSURES

The syntax for defining a closure is:

```
{comma-separated-formal-parameter-list -> statement-list}
```

If no formal parameters are required, then the parameter List and the `->` separator are omitted. Here is a simple example of a closure with no parameters.

```
def clos={println 'Hello world'}
clos.call()
```

Hello world

```
def clos={param -> println "Hello ${param}"}
clos.call('world')           // actual argument is 'world'
clos.call('again')          // actual argument is 'again'
clos('shortcut')            // abbreviated form
```

Hello world
Hello again
Hello shortcut

Observe the third invocation in which the `call` has been omitted.

The next illustration repeats the previous example and produces the same result, but shows that an implicit single parameter referred to as `it` can be used.

```
def clos={println "Hello ${it}"}
clos.call('world')
clos.call('again')

clos('shortcut')
```



Default Single Parameter
"it"

Closures - Enclosing Scope

```
def greeting='Hello' ← "greeting" in defined scope
def clos={param -> println "${greeting} ${param}"}
clos.call('world')

// Now show that changes to this variable change the closure.
greeting='Welcome'
clos.call('world')

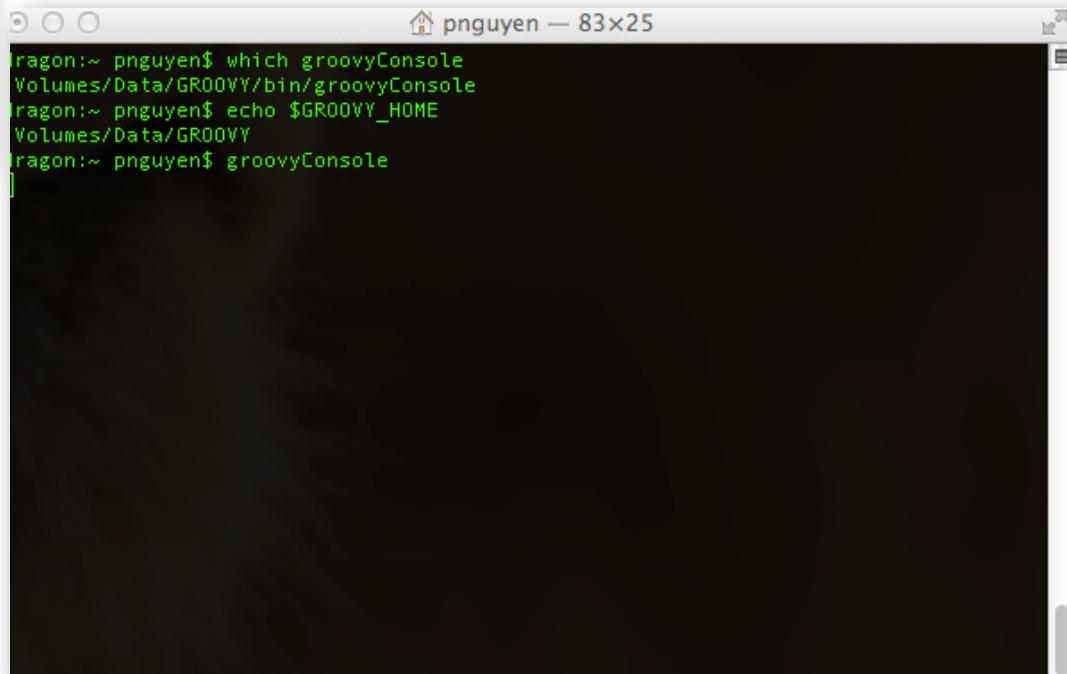
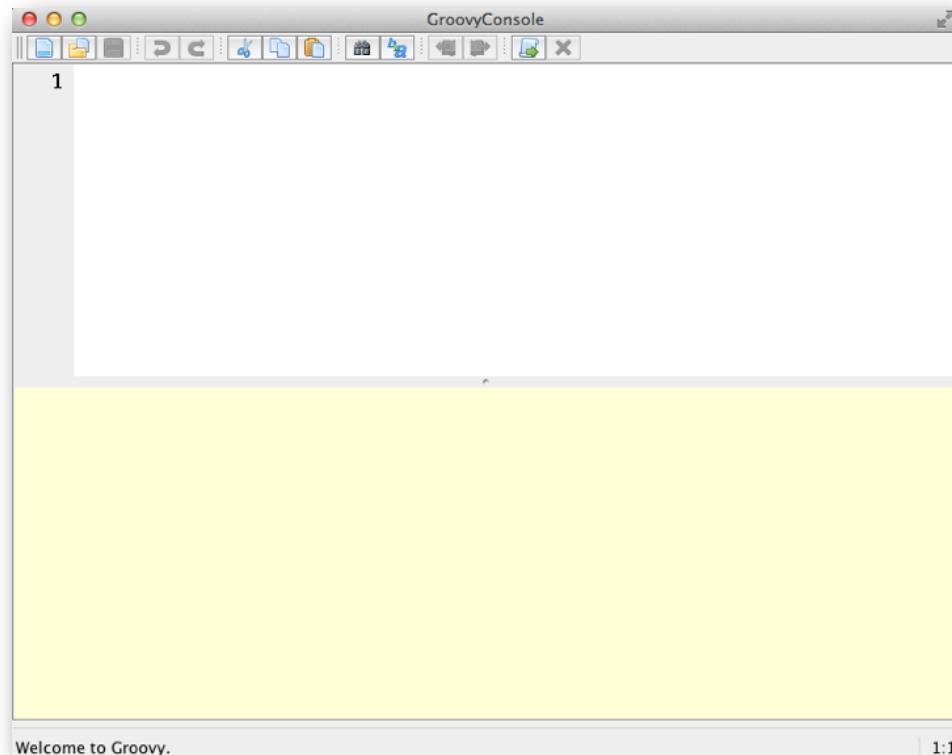
def demo(clo) {
    def greeting='Bonjour'           // does not affect closure
    clo.call('Ken')
}

demo(clos) ← "greeting" in call scope
```

The resulting output is:

```
Hello world
Welcome world
Welcome Ken
```

Groovy Console

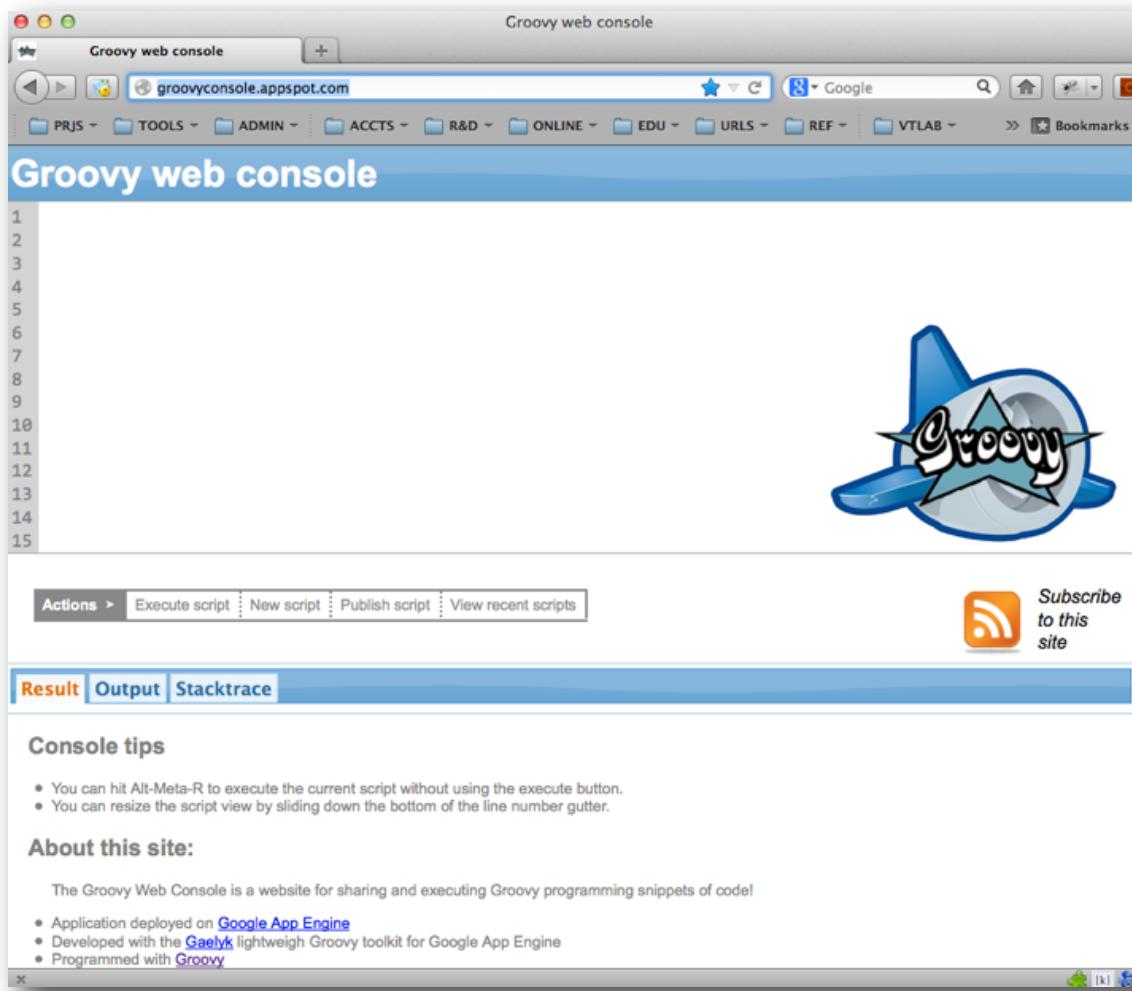


```
pnguyen ~ 83x25
dragon:~ pnguyen$ which groovyConsole
Volumes/Data/GROOVY/bin/groovyConsole
dragon:~ pnguyen$ echo $GROOVY_HOME
Volumes/Data/GROOVY
dragon:~ pnguyen$ groovyConsole
|
```

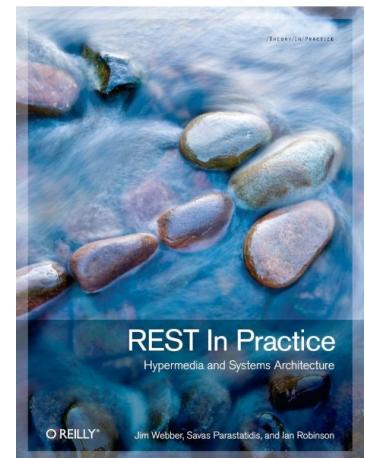
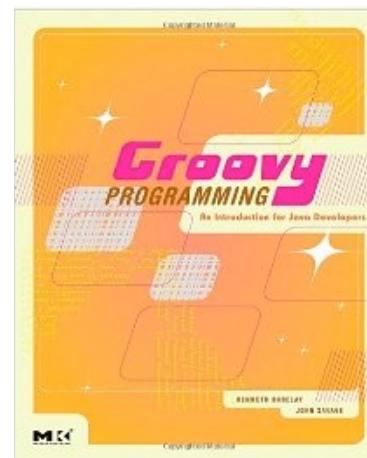
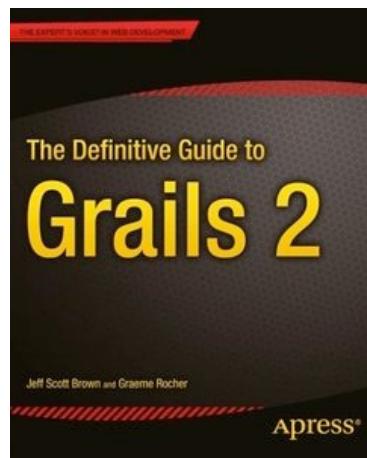
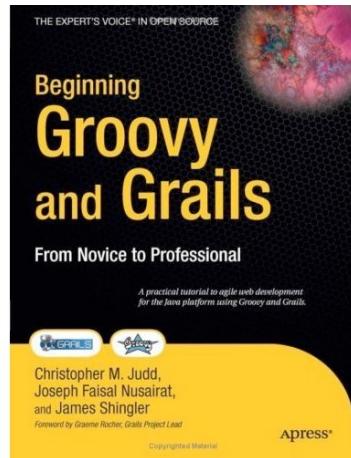
A screenshot of a terminal window titled "pnguyen — 83x25". It shows the user's home directory and the path to the Groovy console executable. The user then runs the Groovy console command, which results in a new terminal window appearing on the screen.

Groovy Web Console

<http://groovyconsole.appspot.com/>



References



#15

DZone Refcardz

[Get More Refcardz! Visit refcardz.com](http://www.refcardz.com)

Groovy
By Dierk König

CONTENTS INCLUDE:

- Groovy/Java Integration
- Language Elements
- Operators
- Collective Duties
- Meta Programming
- Hot Tips and more...

ABOUT GROOVY

Groovy is a dynamic language for the Java™ Virtual Machine (JVM). It shines with full object-orientation, scalability optional typing, operator customization, lexical declarations for the most common data types, advanced concepts like closures and ranges, compact project syntax and seamless Java™ integration. Groovy also provides exactly the kind of information you are likely to look up when programming Groovy.

STARTING GROOVY

Install Groovy from <http://groovy.codehaus.org> and you will have the following commands available:

Command	Purpose
groovy	Execute Groovy code
groovysh	Complete Groovy code
grape	Open Grails
grapeConsole	Open Grails UI console
gradle	Migration helper

The groovy command comes with -h and --help options to show all options and required arguments. Typical usages are:

```
Eval groovy <file>.groovy
groovy MyScript.groovy

Evaluate (e) on the command line
groovy <file> 12.5Math.PI

Print (p) for each line of input
echo 12.5 | groovy -pe
"line.toDouble() * Math.PI"

Inline edit (d) file data.txt by reversing each line and save a backup
groovy -l:bk -pe
"line.reverse()" data.txt
```

GROOVY/JAVA INTEGRATION

From Groovy, you can call any Java code like you would do from Java. It's identical!

From Java, you can call Groovy code in the following ways. Note that you need to have the groovy-all.jar in your classpath.

Cross-compilation

Use either the groovyc or ant task or your IDE integration to compile your groovy code together with your Java code. This enables you to use your Groovy code as if it was written in Java.

Eval

Use class groovy.util.Eval for evaluating simple code that is captured in a Java String: (int) Eval.xy(1,2,3,"x+y+2");

LANGUAGE ELEMENTS

A Groovy class declaration looks like in Java. Default visibility modifier is public.

```
class MyClass {
    void myMethod(String argument) {
    }
}
```

Classes & Scripts

A Groovy class declaration looks like in Java. Default visibility modifier is public.

```
class MyClass {
    void myMethod(String argument) {
    }
}
```

Get More Refcardz
(They're free!)

- Authoritative content
- Designed for developers
- Written by top experts
- Latest tools & technologies
- Hot tips & examples
- Bonus content online
- New issue every 1-2 weeks

Subscribe Now for FREE!
[Refcardz.com](http://refcardz.com)

#60

DZone Refcardz

[Get More Refcardz! Visit refcardz.com](http://www.refcardz.com)

Getting Started with Grails
By Dave Klein

CONTENTS INCLUDE:

- Getting Started with Grails
- Hot Tips and more...

GETTING STARTED WITH GROOVY

Grails is a full-stack web application framework built on top of such tried and true open source frameworks as Spring, Hibernate, and GORM. It makes it easy to get started, such as Convention over Configuration and the Repeat Yourself, and taking advantage of the dynamic Groovy programming language. Grails makes it incredibly easy to use these powerful tools. Grails doesn't reinvent the wheel; Grails makes a wheel that inflates itself and rolls where you want it to!

In case you are new to Grails, we'll start with a brief introduction. If you are already familiar with Java, you hooked and turn into a Grails developer. That's when this Refcard will come in handy; it is a cheat sheet for Grails developers, a quick source for those things you keep having to go back to the docs to look up. Controllers, Services and Views with a detailed GSP taglib reference.

Installing Grails

Download the Grails archive from <http://grails.org> and extract it to a local directory. Set a GRAILS_HOME environment variable to that directory and add GRAILS_HOME/bin to your path. (You also need a valid JAVA_HOME environment variable.) Now you're ready to go!

A Web App in the Blink of an Eye

To create a new Grails application, type:

```
$ grails create-app AutoMart
```

Now change to the AutoMart directory and create a domain class:

```
$ grails create-domain-class Car
```

Open AutoMart/grails-app/domain/Car.groovy and edit it, like so:

```
class Car {
    String make
    String model
    Integer year
}
```

Save this file, and run:

```
$ grails generate-all car
```

Create app and create-domain-class are Grails scripts. To see what other scripts are provided by Grails, run grails help from the command line.

You now have a complete working web application, with pages for creating, displaying, editing and listing Car instances. You can launch it with:

```
$ grails run-app
```

Grails runs on port 8080 by default. You can easily run on a different port like this:

Get More Refcardz
(They're free!)

- Authoritative content
- Designed for developers
- Written by top experts
- Latest tools & technologies
- Hot tips & examples
- Bonus content online
- New issue every 1-2 weeks

Subscribe Now for FREE!
[Refcardz.com](http://refcardz.com)