

# Bootstrapping Groovy

[Prashanth Babu](#) / [@P7h](#)

*You can navigate slides with → and ↓ or  
swipe.*

*Also ESC gives you an bird-eye view.*



**Jeremy Kahn**  
@jeremyckahn

Protip: Nobody is really "qualified" to give tech talks. We're all exploring and figuring it out. Just share what you've learned.

**1,034**  
RETWEETS

**427**  
FAVORITES



9:45 AM - 11 Apr 13

# What's on the menu?

- [Plan](#)
- [Groovy installation and env setup](#)
- [Groovy command line binaries](#)
- [Introduction to Groovy](#)
- [Groovy unique features \[Java comparison\]](#)
- [Closures](#)
- [Lists](#)
- [Ranges](#)
- [Maps](#)
- [Input Output](#)
- [Building and parsing XML or JSON](#)
- [Metaprogramming](#)
- [References](#)

# Plan

- Bird's eye view of Groovy
- Introduce few concepts
- Give sample code
- Do some exercises
- Explore this presentation yourself

*Nothing is taught until something is learned.*

# About these slides

You can get to these slides at <http://P7h.github.io/Groovy>.

You can get the matching source code at

<http://gist.github.com/P7h>.

# Groovy installation and env setup

# Download Groovy

- Download latest and greatest Groovy binary from:  
<http://groovy.codehaus.org/Download>.
- Latest version - as of this writing [08th December, 2013] - is:  
[Groovy 2.2.1](#).



# Groovy Installation and env setup

- Uncompress and move the folder to any location on your computer.
- Add GROOVY\_HOME env variable pointing to the above location.
- Finally add \$GROOVY\_HOME/bin in PATH env variable as well.

```
groovy -version
```

*Hat tip: ~/.groovy/lib folder*

# IDE Plugins for Groovy

- Groovy plugins for various IDEs and Text Editors.
- Groovy / Grails Tool Suite aka [GGTS](#).
- Spring Tool Suite aka [STS](#).

# Groovy command line binaries

# Groovy Commands

- groovy
- groovyc
- groovydoc
- groovysh
- groovyConsole
- java2groovy

# groovy

```
groovy -e "println 'Hello World'"  
groovy HelloWorld.groovy  
groovy HelloWorld
```

# groovyc

```
groovyc HelloWorld.groovy
```

# groovydoc

```
groovydoc HelloWorld.groovy
```

# groovysh

```
groovysh
```

# groovyConsole

```
groovyConsole
```

# java2groovy

```
java2groovy HelloWorld.java
```

# Introduction to Groovy



# Introduction

- What is Groovy?
- Type Systems
- Code snippet showdown: Groovy vs. Java
- How Groovy helps?
- Few demos
- Books on Groovy
- Recommended Book
- Groovy Twitterati

# Tip

*Any Java file can be renamed as a Groovy file.  
And it will compile and execute just as fine.  
Groovy runs on the JVM.*

# Groovy

*Groovy is like a super version of Java. It can leverage Java's enterprise capabilities but also has cool productivity features like closures, DSL support, builders and dynamic typing.*

```
Groovy = Java - boiler plate code
          + mostly dynamic typing
          + closures
          + domain specific languages
          + builders
          + metaprogramming
          + GDK library
```

# Type Systems

- In programming languages, a **type system** is a collection of rules that assign a property called a *type* to the various constructs — such as variables, expressions, functions or modules, etc — a computer program is composed of.

*Groovy is dynamically typed programming language; while Java is statically typed programming language.*

- Groovy

```
def groovyString = "Peter Higgs"  
println groovyString.class
```

- Java

```
String javaString = "Satyendra Nath Bose";
```

# Obligatory "Hello World"

## Java

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

## Groovy

```
println "Hello World"
```

# System Properties

## Java

```
import java.util.Iterator;
import java.util.Properties;
import java.util.Set;
import java.util.SortedMap;
import java.util.TreeMap;

public class SystemProperties {
    public static void main(String[] args) {
        Properties properties = System.getProperties();
        SortedMap sortedSystemProperties = new TreeMap(properties);
        Set keySet = sortedSystemProperties.keySet();
        Iterator iterator = keySet.iterator();
        String key = null;
        while (iterator.hasNext()) {
            key = (String) iterator.next();
            System.out.println(key + " = " + properties.getProperty(key));
        }
    }
}
```

## Groovy

```
System.properties.sort().each { key, value ->
    println "$key = $value"
}
```

# Trivia

Guess what might be the output of this Groovy snippet?

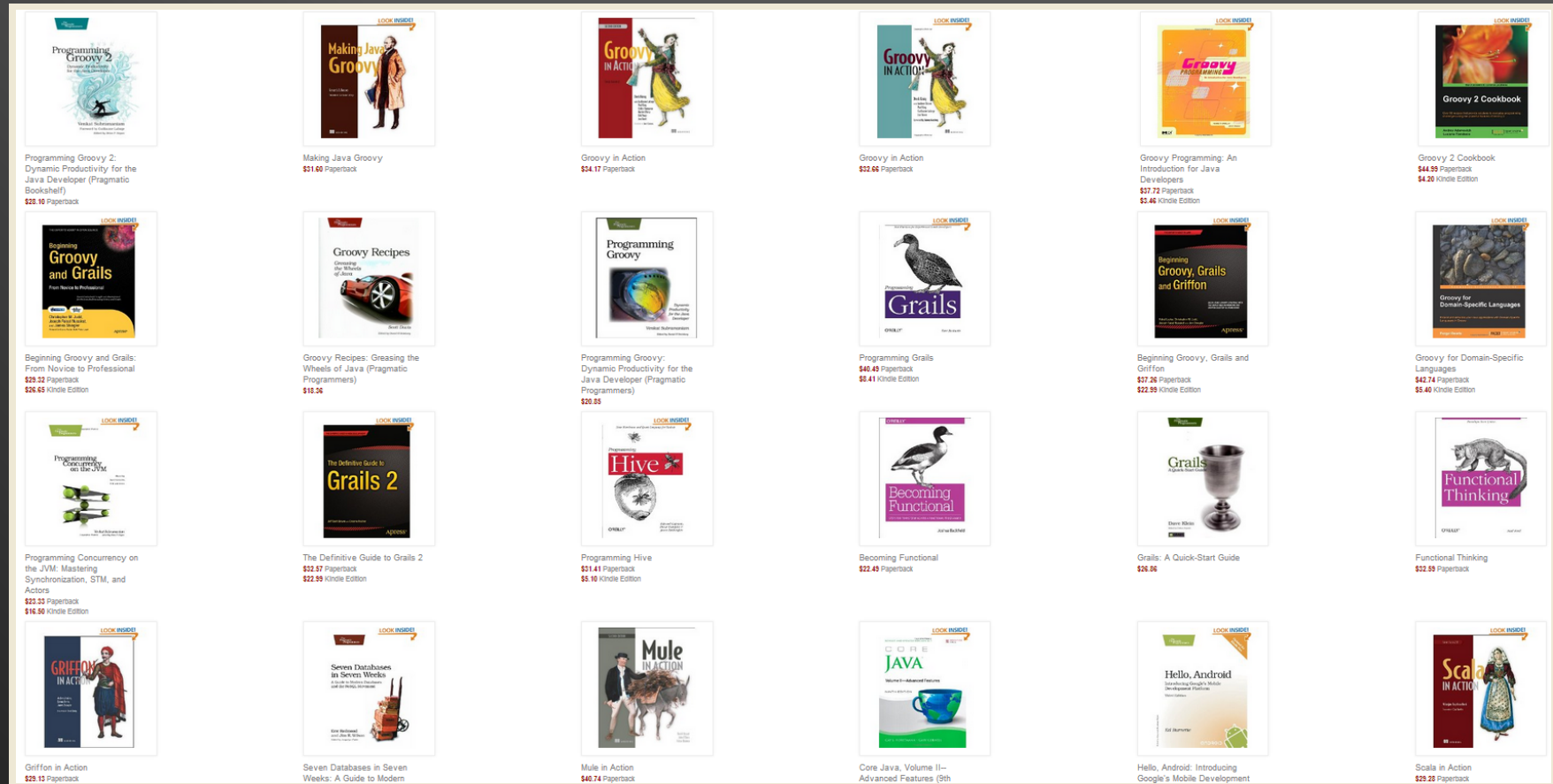
```
String.methods.each {  
    println it  
}
```

# Demos

- Groovy program to fetch and display kitten pictures from Flickr.
- Java build script showdown: Maven vs. Gradle.
- Visualization app built using Groovy and Grails.
- Griffon sample app for loading dummy data to a database.

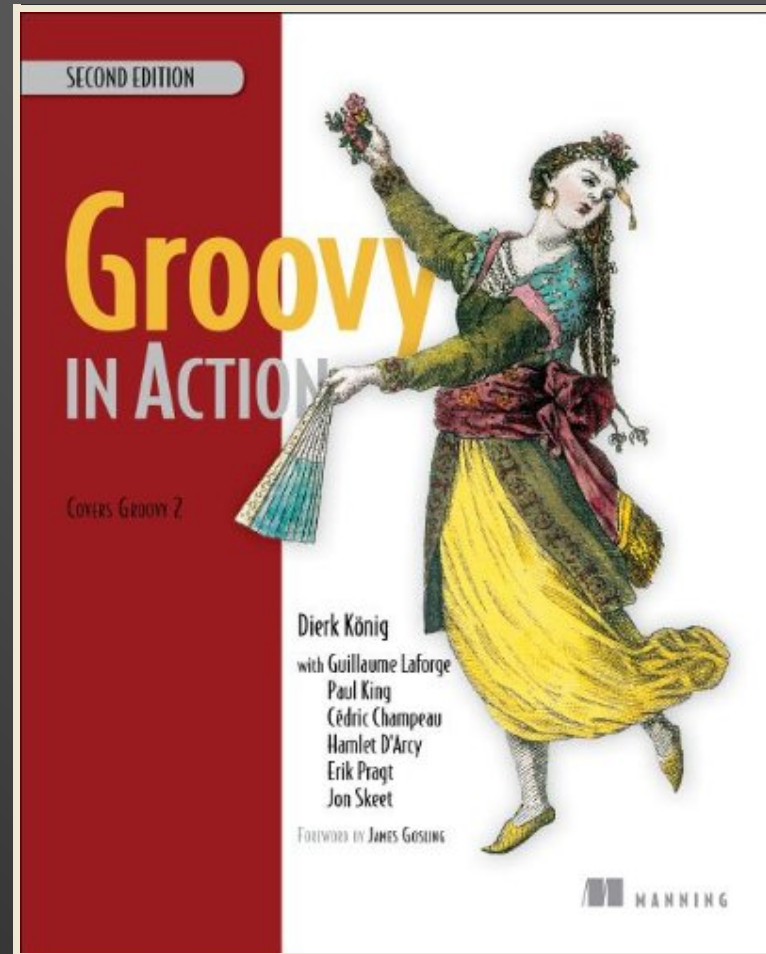


# Books on Groovy



Various Groovy Books on Amazon

# A thorough introduction to Groovy



Groovy in Action aka GinA

# Groovy Twitterati

- [Guillaume Laforge's public Groovy Ecosystem List](#)

## Groovy unique features [Java comparison]

# Groovy Features

- Automatic imports
- Import aliasing
- Optional semicolons & parens
- Optional return statements
- Optional datatype declaration
- Operator overloading
- Safe dereferencing
- Autoboxing
- Groovy Truth
- Embedded Quotes & Heredocs
- GStrings
- POJO vs. POGO

- Automatic imports

```
import java.lang.*  
import java.util.*  
import java.net.*  
import java.io.*  
import java.math.BigInteger  
import java.math.BigDecimal
```

```
import groovy.lang.*  
import groovy.util.*
```

- Import aliasing

```
import java.text.SimpleDateFormat as SDF  
SDF sdf = new SDF("MM/dd/yyyy")  
println sdf.format(new Date())
```

- Optional semicolons and parentheses
- Optional return statements

```
firstName = "James"  
lastName = "Strachan"  
  
String getFullName() {  
    "${firstName} ${lastName}"  
}  
println getFullName()
```

- Optional datatype declaration
- Operator overloading

```
def date = new Date()
date.next()
(1..3).each {
    println date++
}
```



- Safe dereferencing  
Groovy

```
def str = "A String"
println str.size()

str = null
str.size() // Caught: java.lang.NullPointerException: Cannot invoke method si
           // on null object at ConsoleScript3.run(ConsoleScript3:5)
println str?.size()
```

- Safe dereferencing  
Java

```
if (order != null) {  
    if (order.getCustomer() != null) {  
        if (order.getCustomer().getAddress() != null) {  
            System.out.println(order.getCustomer().getAddress());  
        }  
    }  
}
```

## Groovy

```
println order?.customer?.address
```

- Autoboxing

```
def someNumber = 2
println someNumber.class

//Assign the same variable a different value or datatype.
someNumber = 2.0d
println someNumber.class
```

- Groovy Truth

```
//true
if(1) // any non-zero value is true
if(-1)
if(!null) // any non-null value is true
if( "John" ) // any non-empty string is true
Map family = [dad: "John" , mom: "Jane" ]
if(family) // true since the map is populated
String[] sa = new String[1]
if(sa) // true since the array length is greater than 0
StringBuilder sb = new StringBuilder()
sb.append( "Hi" )
if(sb) // true since the StringBuilder is populated

//false
if(0) // zero is false
if(null) // null is false
if( "" ) // empty strings are false
Map family = [].
```

- Embedded Quotes

```
def s1 = 'You either die a "hero" or you live long enough to see yourself bec
def s2 = "You either die a 'hero' or you live long enough to see yourself bec
def s3 = "You either die a \"hero\" or you live long enough to see yourself b
```

- Heredocs

```
String groovyHereDocs1 = """Harvey said "You either die a hero or
you live long enough to see yourself become the villain."""
def groovyHereDocs2 = '''You either die a hero or
you live long enough to see yourself become the villain.'''
println groovyHereDocs2.class
```

- GStrings

```
def firstName = "James"
def lastName = "Strachan"
println "Ahoy ${firstName} ${lastName}, today is ${new Date()}"
```

- POJO

```
public class Person {  
  
    private String firstName;  
    private String lastName;  
  
    public String getFirstName() {  
        return firstName;  
    }  
  
    public void setFirstName(String firstName) {  
        this.firstName = firstName;  
    }  
  
    public String getLastName() {  
        return lastName;  
    }  
  
    public void setLastName(String lastName) {
```

- POGO

```
class Person {
    String firstName
    String lastName
    String toString() {
        firstName + " " + lastName
    }
}

//One way of instantiating an object in Groovy.
def person01 = new Person()
person01.firstName = "James"
person01.lastName = "Strachan"
println "Name: " + person01
```

```
//Another way of instantiating an object in Groovy.
person01.with {
    firstName = "Guillaume"
    lastName = "Laforge"
}
println "Name again: " + person01
```

```
//Yet another way of instantiating an object in Groovy.
def person02 = new Person(lastName: "Rocher", firstName: "Graeme")
println "Name yet again: " + person02
```

- Groovy coding guidelines

Groovy style and language feature guidelines for Java  
developers



# Closures

# Closures

- Not to be confused with Clojure, another JVM based dynamic programming language.
- Closure is a free-standing, named block of code.
- It is a behavior that doesn't have a surrounding class.
- Helps in greater flexibility.
- Metaprogramming to enhance existing libraries.
- As of this writing, Java does not have this feature.
- Java 8 is scheduled to come with this most wanted feature coined as Lambdas in March, 2014.

- Closures without any parameters

```
def x = 5, y = 6;  
def printNum {  
  println x + " " + y  
}  
printNum()
```

- Closures with parameters

```
def printSum = { a, b ->  
  print a + b  
}  
printSum 5, 7
```

- **Traditional mainstream languages**

Data can be stored in variables, passed around, combined in structured ways to form more complex data; code stays put where it is defined.

- **Languages supporting closures**

Data *and* code can be stored in variables, passed around, combined in structured ways to form more complex algorithms and data.

```
doubleNum = {  
  num -> num * 2  
}  
assert doubleNum(3) == 6  
  
def processThenPrint = { num, closure ->  
  num = closure(num);  
}  
assert processThenPrint(3, doubleNum) == 6  
assert processThenPrint(10) { it / 2 } == 5
```

# Implicit variables

- A Closure that takes a single argument may omit the parameter definition of the Closure.

```
def printStr = {  
  print it  
}  
printStr "Print but with a custom method"
```

```
String.methods.each {  
  println it  
}
```

- *More about Closures in List and Map Comprehensions later.*

# Currying

- Transform function with particular number of parameters and returns a function with some of the parameter values fixed, creating a new function.
- Curry as many parameters as required.
- The first curry call fills in the leftmost parameter.
- Each subsequent call fills in the next parameter to the right.

```
/* Tax calculation example */  
  
def calculateTax = { taxRate, amount ->  
    amount + (taxRate * amount)  
}  
  
def tax = calculateTax.curry(0.1)  
  
(10..15).each {  
    println "Total cost: ${tax(it)}"  
}
```

# Loop variants

```
def result = ''
def compute = {
  if (!it) {
    result = '0'
  } else {
    result += it
  }
}

5.times compute
assert '01234' == result

0.upto 7, compute
assert '01234567' == result

0.step 10, 2, compute
assert '02468' == result
```

Lists



# Lists

```
def list = [1, 2, 3, 4]
assert list.size == 4
assert list.size() == 4
assert list.class == ArrayList
```

```
def list = []
assert list.size() == 0
list << 5
assert list.size() == 1
list << 'a' << 'e' << 'i'
assert list == [5, 'a', 'e', 'i']
```

# Lists

```
assert [0, 1, 2] + 3 + [4, 5] + 6 == [0, 1, 2, 3, 4, 5, 6]
assert [0, 1, 2].plus(3).plus([4, 5]).plus(6) == [0, 1, 2, 3, 4, 5, 6]

def list = [0, 1, 2, 3]
list += 4
list += [5, 6]
assert list == [0, 1, 2, 3, 4, 5, 6]
assert list[3] == 3
assert list.getAt(5) == 5

list.putAt(5, -5)
assert list[5] == -5
assert list[-4] == 3
assert list[-4] * 2 == 6
assert list * 2 == [0, 1, 2, 3, 4, -5, 6, 0, 1, 2, 3, 4, -5, 6]

assert ('a'..'g')[3..5] == ['d', 'e', 'f']
```

# Lists

```
assert [1,[2,3]].flatten() == [1,2,3]
assert [1,2,3].intersect([4,3,1]) == [3,1]
assert [1,2,3].disjoint([4,5,6])
```

```
list = [1,2,3]
popped = list.pop()
assert popped == 3
assert list == [1,2]
```

```
assert [1,2].reverse() == [2,1]
assert [3,1,2].sort() == [1,2,3]
```

# Lists

```
def list = [1, 2, 3, 4, 5]
def sublist = list.subList(2, 4)
sublist[0] = 9
assert list == [1, 2, 9, 4, 5]
list[3] = 11
assert sublist == [9, 11]
```

```
def range = ('a'..'f')
range.forEachWithIndex{ it, i ->
    println "$i: $it"
}
assert range.join(', ') == 'a, b, c, d, e, f'
```

# Lists

- Double each element in a List using a closure

```
def doubled = [1,2,3].collect { item ->
  item * 2
}
assert doubled == [2,4,6]
```

- Divide a list into 2 lists with one list containing only unique elements and another only duplicates.

```
list=[1, 2, 7, 2, 2, 4, 7, 11, 5, 2, 5]
def uniques = [] as Set, dups = [] as Set
list.each {
  uniques.add(it) || dups.add(it)
}
uniques.removeAll(dups)
assert uniques == [1, 4, 11] as Set && dups == [2, 5, 7] as Set
```

Ranges

# Ranges

- Groovy offers a native datatype for Ranges.
- Ranges are specified using the double-dot range operator `..` between the left and right bounds.

```
def range = 1..10
println range.class
println range instanceof List
range = 1..<10
def alphabets = 'a'..'z'
println alphabets.class
assert alphabets.contains('b')
alphabets.each {
    print it
}
```

```
IntRange.methods.each {
    println it
}
```

# Few other methods of Range

```
def i = 1..5
println i.size()
println i.from
println i.to
println i.contains(1)
println i.contains(-9)
println i.reverse()
```

```
def today = new Date()
def nextWeek = today + 7
(today..nextWeek).each {
    println it
}
```



# Ranges in switch cases

```
def stage = ""
def years = 58
switch (years) {
  case 1..3:
    stage = "toddler"
    break;
  case 3..15:
    stage = "school"
    break;
  case 16..22:
    stage = "college"
    break;
  case 22..58:
    stage = "job"
    break;
  default:
    stage = "retired"
}
```

Maps

# Maps

- Simple examples

```
def map = [name:"Sachin Tendulkar", likes:"Cricket", id:10]
assert map instanceof java.util.Map

assert map.get("name") == "Sachin Tendulkar"
assert map["name"] == "Sachin Tendulkar"
assert map.name == "Sachin Tendulkar"

assert map.get("id") == 10
assert map['id'] == 10
assert map.id == 10

map['status'] = "Retired"
assert map == [name:'Sachin Tendulkar', likes:'Cricket', id:10, status:'Retired']
```

# Maps

- Iterating a Map

```
def map = ['Android': '4.4.2',  
          'iPhone': 'iOS7',  
          'Windows': 'Win8',  
          'BlackBerry': 'BlackBerry10']  
map.each { k, v ->  
  println "${k} = ${v}"  
}
```

# Maps

- **+** and **<<** operators can be used to add elements to the Map.
- But **<<** produces a new map while **+** modifies the Map.

```
def map01 = [name:"Sachin Tendulkar", likes:"Cricket", id:10, country:"India"]  
println map01
```

```
map01 << [name:"Roger Federer", likes:"Tennis", id:101]  
println map01
```

```
def map02 = [name:"Sachin Tendulkar", likes:"Cricket", id:10, status: "Retired"]  
println map02
```

```
map02 += map01  
println map02
```

# Maps

- Variants of iterating a Map

```
def p = new StringBuilder()
[1:'a', 2:'b', 3:'c'].each {
  p << it.key + ': ' + it.value + '; '
}
assert p.toString() == '1: a; 2: b; 3: c; '
```

```
def q = new StringBuilder()
[1:'a', 2:'b', 3:'c'].each { k, v ->
  q << k + ': ' + v + '; '
}
assert q.toString() == '1: a; 2: b; 3: c; '
```

```
def r = new StringBuilder()
[1:'a', 2:'b', 3:'c'].eachWithIndex { it, i ->
  r << it.key + ' : ' + it.value + '; '
}
assert r.toString() == '1 : a; 2 : b; 3 : c; '
```

Input Output

# Input Output

## Read a file line-by-line

```
new File("ReadTheFile.txt").eachLine { line ->  
    println(line)  
}
```



# Read a file with a Reader

```
def count=0, MAXSIZE=100
new File("ReadTheFile.txt").withReader { reader ->
  while (reader.readLine() != null) {
    if (++count > MAXSIZE) {
      println "Read 100 lines already!"
      break
    }
  }
}
```

# Write a file with a Writer

```
def fields = ["a":"1", "b":"2", "c":"3"]
new File("WriteTheFile.ini").withWriter { out ->
  fields.each() { key, value ->
    out.writeLine("${key}=${value}")
  }
}
```

# Loading a properties file in Groovy

```
java:oracle
```

```
def properties = new Properties()
new File("config.properties").withInputStream {
    stream -> properties.load(stream)
}

println "java = " + properties["java"]

properties.each { k, v ->
    println "${k} = ${v}"
}
```

Building and parsing XML or JSON

# Builders and Parsers

- Extensive support building and parsing HTML, XML and JSON.

# Building a XML file

```
def stringWriter = new StringWriter()
def builder = new groovy.xml.MarkupBuilder(stringWriter)
builder.numbers {
    description 'Squares and factors of 10..15'
    for (i in 10..15) {
        number (value: i, square: i*i) {
            for (j in 2..<i) {
                if (i % j == 0) {
                    factor (value: j)
                }
            }
        }
    }
}
println stringWriter
```

# Building a JSON file

```
import groovy.json.*

class Player {
    String name
    List faveGrounds = []
}

class FaveGround {
    String ground
    String city
}

Player playr = new Player(name:"Sachin Tendulkar")
playr.faveGrounds << new FaveGround(ground: "Wankede", city:"Mumbai")
playr.faveGrounds << new FaveGround(ground: "SCG", city:"Sydney")

def builder = new JsonBuilder()
def root = builder players {
```

# Reading an XML file

```
<langs type="languages">
  <language>Python</language>
  <language>Groovy</language>
  <language>Java</language>
</langs>
```

```
def langs = new XmlParser().parse("langs.xml")
println "type = ${langs.attribute("type")}"
langs.language.each{
    println it.text()
}
```



# Reading an XML file using XMLSlurper

```
def CAR_RECORDS = '''
  <records>
    <car name='HSV Maloo' make='Holden' year='2006'>
      <country>Australia</country>
      <record type='speed'>Production Pickup Truck with speed of 271kph</re
    </car>
    <car name='P50' make='Peel' year='1962'>
      <country>Isle of Man</country>
      <record type='size'>Smallest Street-Legal Car at 99cm wide and 59 kg
    </car>
    <car name='Royale' make='Bugatti' year='1931'>
      <country>France</country>
      <record type='price'>Most Valuable Car at $15 million</record>
    </car>
  </records>
'''

def records = new XmlSlurper().parseText(CAR_RECORDS)
```

# Reading a JSON file JSONSlurper

```
import groovy.json.*

def jsonText = '''
{
  "limit":{
    "track":1234
  }
}
'''

def json = new JsonSlurper().parseText(jsonText)

def limit = json.limit
assert limit.track == 1234
```

# Metaprogramming

# Metaprogramming

- Metaprogramming refers to writing code that can dynamically change its behavior at runtime.
- A Meta-Object Protocol [MOP] refers to the capabilities in a dynamic language that enable metaprogramming.
- In Groovy, the MOP consists of four distinct capabilities within the language: reflection, metaclasses, categories, and expandos.

```
println "Hello".toUpperCase()

String.metaClass.toUpperCase = {
    delegate.toLowerCase()
}

println "Hello".toUpperCase()
```

```
Integer.metaClass.isEven = { ->
    delegate % 2 == 0
}
println 7.isEven()
println 10.isEven()
```

```
Integer.metaClass.static.isEven = { number ->
    number % 2 == 0
}
Integer.isEven(1)
Integer.isEven(2)
```

## References

# Groovy references

## Web

[Groovy Website](#)

[MrHAKI's Groovy Goodness Snippets](#)

## Mailing list for users

- [user@groovy.codehaus.org](mailto:user@groovy.codehaus.org)

## Books

# Bootstrapping Groovy

[Prashanth Babu](#) / [@P7h](#)

Thanks

Q n A