Groovy Session

Running Groovy on the Command Line

```
groovy -e 'println "Hello, ${args[0]}. ${args[1]}"' Buddy
'Have a nice day!'
```

Using groovyConsole

```
%GROOVY_HOME%/bin/groovyConsole
```

Default Imports

java.lang.*, java.util.*, java.io.*, java.net.*, java.math.BigDecimal, java.math.BigInteger, groovy.lang.* , groovy.util.*

Ways to Loop

```
0.upto(2) { print "$it " }
3.times { print "$it " }
0.step(10, 2) { print "$it " }
```

A Quick Look at the GDK

```
println "git help".execute().text
println "groovy -v".execute().getClass().name
```

Safe Navigation Operator

```
//if (str != null) { return str.reverse() }
str?.reverse()
```

Exception Handling

```
def openFile(fileName){
	new FileInputStream(fileName)
}

try{
	openFile("nonexistentfile" )
}catch(FileNotFoundException ex)
{// Do whatever you like about this exception here
	println "Oops: " + ex
}
```

Groovy as Lightweight Java

- String objects are given special importance
- return is almost optional
- ; is almost optional
- Methods and classes are public by default.
- ?. operator
- ?: elvis operator
- Dynamic typing using def keyword
- Groovy automatically treats all primitives(char, int, float, double, byte, short etc) as objects. 1.class.name, 10.4.class.name, 'a'.class.name
- Easy syntax for List and Map e.g. [1,2,3] is a list and [name:'bhagwat', male:true] is a Map
- Constructors using named parameters (Map)
- use **this** within static methods to refer to the Class object
- Getters and Setters automatically created
- property like access of Getter methods //str.getClass().getName()   // str.class.name
- optional parenthesis in method calls which take at least one argument
- optional typing in method signature
- default arguments/Optional Parameters
- Implementing Interfaces

```
interface SayHello{
 String hello();
}

void callMe(SayHello s){
 println s.hello();
}

class MyClass implements SayHello{
  String hello(){return "Hello"}
}

//callMe(new MyClass())

callMe({"Me Hello"} as SayHello)
```

Groovy boolean Evaluation

you can use an object reference where a boolean expression is expected.

null is false

Boolean > true or false
numeric value > 0=false otherwise true
enumeration iterator > have elements then true
String > empty=false

Operator Overloading

http://groovy.codehaus.org/Operator+Overloading

```
class MyClass{
  int x
//  MyClass(int){}
  MyClass plus(MyClass other){
          return new MyClass(x:this.x+other.x)
 }
}


MyClass m=new MyClass(x:12)
MyClass n=new MyClass(x:13)

//MyClass p=m+n
MyClass p=m.plus(n)

println p.x
```

each operator in groovy has a standard mapping to java methods.
e.g.

```
List fruits=["apple", "banana"]
fruits<<"mango"     // fruits.add("mango")

StringBuffer sb=new StringBuffer()
sb<<"Hello"             // sb.append("Hello")
```

Support of Java 5 Language Features

Autoboxing, for-each, enum, Varargs, Annotation, Static

import, Generics

enum CoffeeSize { SHORT, SMALL, MEDIUM, LARGE, MUG }
def foo1(int a, int... b)
import static Math.random;
import groovy.lang.ExpandoMetaClass as EMC

Gotchas : ranging from minor annoyances to surprises if you're not expecting them

- last statement result is returned in a method if you are not using return explicitly
- Groovy's == Is Equal to Java's equals // use is( ) in Groovy for reference comparison
- No Compile-Time Type Checking //Integer val = 4; val = 'hello'; val.blah();
- def, in, it should not be used as variable name
- No Inner Classes
- No Code Block
- Missing semicolon sometimes leads to problem

```
 class Semi{
def val = 3
{println "Instance Initializer called..."}
}
println new Semi()
```

- Different Syntax for Creating Primitive Arrays

```
int[] arr = new int[] {1, 2, 3, 4, 5};  //  java way
int[] arr = [1, 2, 3, 4, 5]   //  groovy way
```

Working with Strings

```
'a' and "a" are both strings
String user="John"
String x="a"+"b"
String z='Hello dear "Bhagwat" '
String k="Hello dear 'Bhagwat' "
def multiLine1='''
Hello
I am spread over
multiple lines '''
```

```
def multiLine2="""
Another one
which is spread over multiple lines
"""
```

GString

```
println "hello dear $user"
println "Hello the no. of chars in user : ${user.size()}"
```

String Methods

```
str = "It's a rainy day in Seattle"
str -= "rainy "
str += "today"
println "<>" * 20
str.split()
str.tokenize()
```

Closure

Named or anonymous code block that can be used as methods or passed as an argument to other methods/clousers

```
def x={println it}  // x(10)
def y={var->  println var}  //y(10)
def z={String name->  println name}  // z("Hello")
def sum={int x, int y-> return (x+y)}  // sum(10, 20)
def square={num -> num*num}  // square(10)

[1,2,3,4].each(square)
```

Compiling Groovy files

```
groovyc    // groovy compiler
groovyc -j Country.groovy World.java  // joint compiler
groovy     // for running class files
```

Working with Files

http://www.groovyexamples.org/2010/05/03/list-all-files-in-a-directory/

```
new File("foo.txt").text
new File("foo.txt").bytes
```

```
new File("foo.txt").readLines()
new File("foo.txt").text="abc"
new File("foo.txt").append("abc")
new File("foo.txt").eachLine { line -> println(line) }

dir.eachFile
dir.eachDir
dir.eachFileRecurse
```