Groovy

Groovy - A dynamic language for the Java platform Read more here.

```
/*
  Set yourself up:

  1) Install SDKMAN - http://sdkman.io/
  2) Install Groovy: sdk install groovy
  3) Start the groovy console by typing: groovyConsole

*/

//  Single line comments start with two forward slashes
/*
Multi line comments look like this.
*/

// Hello World
println "Hello world!"

/*
  Variables:

  You can assign values to variables for later use
*/

def x = 1
println x

x = new java.util.Date()
println x

x = -3.1499392
println x

x = false
println x

x = "Groovy!"
println x

/*
  Collections and maps
*/

//Creating an empty list
def technologies = []

/*** Adding a elements to the list ***/

// As with Java
technologies.add("Grails")
```

```
// Left shift adds, and returns the list
technologies << "Groovy"

// Add multiple elements
technologies.addAll(["Gradle","Griffon"])

/*** Removing elements from the list ***/

// As with Java
technologies.remove("Griffon")

// Subtraction works also
technologies = technologies - 'Grails'

/*** Iterating Lists ***/

// Iterate over elements of a list
technologies.each { println "Technology: $it"}
technologies.eachWithIndex { it, i -> println "$i: $it"}

/*** Checking List contents ***/

//Evaluate if a list contains element(s) (boolean)
contained = technologies.contains( 'Groovy' )

// Or
contained = 'Groovy' in technologies

// Check for multiple contents
technologies.containsAll(['Groovy','Grails'])

/*** Sorting Lists ***/

// Sort a list (mutates original list)
technologies.sort()

// To sort without mutating original, you can do:
sortedTechnologies = technologies.sort( false )

/*** Manipulating Lists ***/e

//Replace all elements in the list
Collections.replaceAll(technologies, 'Gradle', 'gradle')

//Shuffle a list
Collections.shuffle(technologies, new Random())

//Clear a list
technologies.clear()

//Creating an empty map
def devMap = [:]
```

```
//Add values
devMap = ['name':'Roberto', 'framework':'Grails', 'language':'Groovy']
devMap.put('lastName','Perez')

//Iterate over elements of a map
devMap.each { println "$it.key: $it.value" }
devMap.eachWithIndex { it, i -> println "$i: $it"}

//Evaluate if a map contains a key
assert devMap.containsKey('name')

//Evaluate if a map contains a value
assert devMap.containsValue('Roberto')

//Get the keys of a map
println devMap.keySet()

//Get the values of a map
println devMap.values()

/*
  Groovy Beans

  GroovyBeans are JavaBeans but using a much simpler syntax

  When Groovy is compiled to bytecode, the following rules are used.

    * If the name is declared with an access modifier (public, private or
      protected) then a field is generated.

    * A name declared with no access modifier generates a private field with
      public getter and setter (i.e. a property).

    * If a property is declared final the private field is created final and no
      setter is generated.

    * You can declare a property and also declare your own getter or setter.

    * You can declare a property and a field of the same name, the property will
      use that field then.

    * If you want a private or protected property you have to provide your own
      getter and setter which must be declared private or protected.

    * If you access a property from within the class the property is defined in
      at compile time with implicit or explicit this (for example this.foo, or
      simply foo), Groovy will access the field directly instead of going though
      the getter and setter.

    * If you access a property that does not exist using the explicit or
      implicit foo, then Groovy will access the property through the meta class,
      which may fail at runtime.

*/
```

```
class Foo {
    // read only property
    final String name = "Roberto"

    // read only property with public getter and protected setter
    String language
    protected void setLanguage(String language) { this.language = language }

    // dynamically typed property
    def lastName
}

/*
  Logical Branching and Looping
*/

//Groovy supports the usual if - else syntax
def x = 3

if(x==1) {
    println "One"
} else if(x==2) {
    println "Two"
} else {
    println "X greater than Two"
}

//Groovy also supports the ternary operator:
def y = 10
def x = (y > 1) ? "worked" : "failed"
assert x == "worked"

//Groovy supports 'The Elvis Operator' too!
//Instead of using the ternary operator:

displayName = user.name ? user.name : 'Anonymous'

//We can write it:
displayName = user.name ?: 'Anonymous'

//For loop
//Iterate over a range
def x = 0
for (i in 0 .. 30) {
    x += i
}

//Iterate over a list
x = 0
for( i in [5,3,2,1] ) {
    x += i
}
```

```
//Iterate over an array
array = (0..20).toArray()
x = 0
for (i in array) {
    x += i
}

//Iterate over a map
def map = ['name':'Roberto', 'framework':'Grails', 'language':'Groovy']
x = 0
for ( e in map ) {
    x += e.value
}

/*
  Operators

  Operator Overloading for a list of the common operators that Groovy supports:
  http://www.groovy-lang.org/operators.html#Operator-Overloading

  Helpful groovy operators
*/
//Spread operator:  invoke an action on all items of an aggregate object.
def technologies = ['Groovy','Grails','Gradle']
technologies*.toUpperCase() // = to technologies.collect { it?.toUpperCase() }

//Safe navigation operator: used to avoid a NullPointerException.
def user = User.get(1)
def username = user?.username


/*
  Closures
  A Groovy Closure is like a "code block" or a method pointer. It is a piece of
  code that is defined and then executed at a later point.

  More info at: http://www.groovy-lang.org/closures.html
*/
//Example:
def clos = { println "Hello World!" }

println "Executing the Closure:"
clos()

//Passing parameters to a closure
def sum = { a, b -> println a+b }
sum(2,4)

//Closures may refer to variables not listed in their parameter list.
def x = 5
def multiplyBy = { num -> num * x }
println multiplyBy(10)

// If you have a Closure that takes a single argument, you may omit the
```

```
// parameter definition of the Closure
def clos = { print it }
clos( "hi" )

/*
  Groovy can memorize closure results [1][2][3]
*/
def cl = {a, b ->
    sleep(3000) // simulate some time consuming processing
    a + b
}

mem = cl.memoize()

def callClosure(a, b) {
    def start = System.currentTimeMillis()
    mem(a, b)
    println "Inputs(a = $a, b = $b) - took ${System.currentTimeMillis() - start} msecs."
}

callClosure(1, 2)
callClosure(1, 2)
callClosure(2, 3)
callClosure(2, 3)
callClosure(3, 4)
callClosure(3, 4)
callClosure(1, 2)
callClosure(2, 3)
callClosure(3, 4)

/*
  Expando

  The Expando class is a dynamic bean so we can add properties and we can add
  closures as methods to an instance of this class

  http://mrhaki.blogspot.mx/2009/10/groovy-goodness-expando-as-dynamic-bean.html
*/
  def user = new Expando(name:"Roberto")
  assert 'Roberto' == user.name

  user.lastName = 'Pérez'
  assert 'Pérez' == user.lastName

  user.showInfo = { out ->
      out << "Name: $name"
      out << ", Last name: $lastName"
  }

  def sw = new StringWriter()
  println user.showInfo(sw)


/*
```

```
  Metaprogramming (MOP)
*/

//Using ExpandoMetaClass to add behaviour
String.metaClass.testAdd = {
    println "we added this"
}

String x = "test"
x?.testAdd()

//Intercepting method calls
class Test implements GroovyInterceptable {
    def sum(Integer x, Integer y) { x + y }

    def invokeMethod(String name, args) {
        System.out.println "Invoke method $name with args: $args"
    }
}

def test = new Test()
test?.sum(2,3)
test?.multiply(2,3)

//Groovy supports propertyMissing for dealing with property resolution attempts.
class Foo {
   def propertyMissing(String name) { name }
}
def f = new Foo()

assertEquals "boo", f.boo

/*
  TypeChecked and CompileStatic
  Groovy, by nature, is and will always be a dynamic language but it supports
  typechecked and compilestatic

  More info: http://www.infoq.com/articles/new-groovy-20
*/
//TypeChecked
import groovy.transform.TypeChecked

void testMethod() {}

@TypeChecked
void test() {
    testMeethod()

    def name = "Roberto"

    println naameee

}
```

```
//Another example:
import groovy.transform.TypeChecked

@TypeChecked
Integer test() {
    Integer num = "1"

    Integer[] numbers = [1,2,3,4]

    Date date = numbers[1]

    return "Test"

}

//CompileStatic example:
import groovy.transform.CompileStatic

@CompileStatic
int sum(int x, int y) {
    x + y
}

assert sum(2,5) == 7
```

Further resources

Groovy documentation

Groovy web console

Join a Groovy user group

Books

- [Groovy Goodness] (https://leanpub.com/groovy-goodness-notebook)
- [Groovy in Action] (http://manning.com/koenig2/)
- [Programming Groovy 2: Dynamic Productivity for the Java Developer] (http://shop.oreilly.com/product/97819377853(

[1] http://roshandawrani.wordpress.com/2010/10/18/groovy-new-feature-closures-can-now-memorize-their-results/ [2] http://www.solutionsiq.com/resources/agileiq-blog/bid/72880/Programming-with-Groovy-Trampoline-and-Memoize [3] http://mrhaki.blogspot.mx/2011/05/groovy-goodness-cache-closure-results.html