Groovy Programming language

1. Definition

- Apache Groovy is an object-oriented and Java syntax-compatible programming language built for the Java platform.
- This dynamic language has many features which are similar to Python, Ruby, Smalltalk, and Pero.
- Groovy source code gets compiled into Java Bytecode so it can run on any platform that has JRE installed.
- Groovy also performs a lot of tasks behind the scene that makes it more agile and dynamic.
- Groovy language can be used as a scripting language for the Java platform. It is almost like a super version of Java which offers Java's enterprise capabilities.
- It also offers many productivity features like DSL support, closures, and dynamic typing. Unlike some other languages, it is designed as a companion, not a replacement for Java.

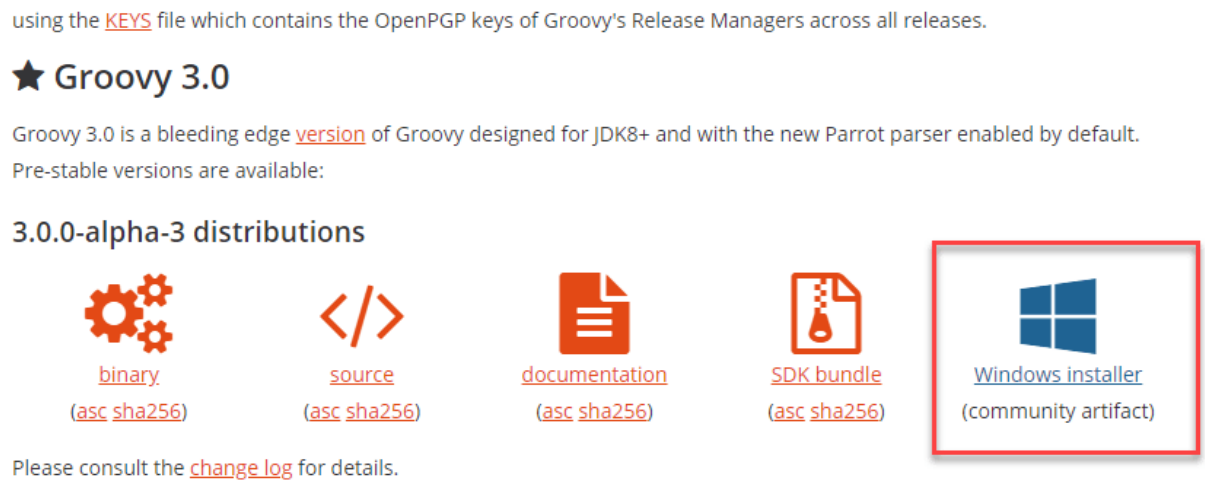
2. Features of Groovy

- Support for static and dynamic typing
- Concise, brief, direct syntax
- The relatively short learning curve
- Support for unit testing
- Native support for regular expressions
- The native syntax for lists and associative arrays
- Native support for markup languages like XML and HTML.
- Support for domain-specific languages

3. Installation of groovy on Linux

- **Ensure you have Java installed.**
  https://www.guru99.com/install-java.html

- **Go to http://groovy-lang.org/download.html and click installer.**



download the Groovy 2.0 binaries as described in the Installing Groovy on Windows recipe and perform the following steps to install Groovy on Linux and OS X:

➔ Create a new folder for the Groovy distribution:

```
sudo mkdir /home/user_name/groovy
```

➔ Move the unzipped Groovy folder into **/home/user_name/groovy** and create a symlink to the folder, without using the version number:

```
sudo mv groovy-version /home/user_name/groovy/

sudo ln -s /home/user_name/groovy/groovy-version current
```

➔ Finally, add Groovy to the path using this command:

```
sudo -H gedit /etc/profile.d/groovy.sh
```

➔ Then add the following line and save

```
export GROOVY_HOME= /home/user_name/groovy/current
export PATH=$GROOVY_HOME/bin:$PATH
```

➔ Your JAVA_HOME variable should be set as well. On OS X, the recommended way to set the variable is as follows:

```
export JAVA_HOME= java.home_path
```

➔ For searching the java. home path in Linux , can use one of the following commands:

```
java -XshowSettings:properties -version

OR

java -XshowSettings:properties -version 2>&1 > /dev/null | grep
'java.home'
```

➔ Example:-

```
export JAVA_HOME=/usr/lib/jvm/java-8-openjdk-amd64/jre
```

➔ To test if your installation is successful, type:

```
groovy -version
```

➔ As the first test for the groovy program Make one groovy script using the following command:

```
sudo -H gedit Hello_world.groovy
```

➔ And write

```
println "hello world..!"
```

➔ After write this save the file
➔ And for a run this file type

```
groovy Hello_world.groovy
```

➔ It will give you the following output

```
Hello world..!
```

➔ Installation of groovy in your Linux system is done!

4. Groovy Hello-world Example:

- If we want to print hello world in java we would write like following
  Public class Hello_world{

```
class HelloWorld {

public static void main(String args[]) {

System.out.println("Hello world…!");

}

}
```

The above code is valid in both Java and Groovy as Groovy is a superset of Java. But the advantage with Groovy is that we can do we away with class creation, public method creation, etc and achieve the same output with a single line code as follows:

```
println (“Hello World…!”)

OR

println (‘Hello World…!’)

OR

println ‘Hello world’
```

- There is no need for semicolons
- There is no need for parenthesis
- System.out.println is reduced to println

5. Variables

a. Define variable

In groovy we can define variables in the following ways

1.using def

2.using the data type

3.only variable name

```
//1. define variable
def  name1 = "Tom1" //using def
String name2 = "Tom2" // using data type
name3 ="Tom3" //using only variable name
name4 ='Tom'
//name5 = Tom // won't work
println name1+" "+name2+" "+name3+" "+name4

//output:
/*Tom1 Tom2 Tom3 Tom*/

//Example of define a variable in groovy
def name = "Tom"
println  "my name is ${name}"
//will replace the value of name and print
println  'my name is ${name}'
//will print the statement without replacing
println "my name1 is "+name
println 'my name2 is '+name
//will replace the name value and print

//output:
/*my name is Tom
my name is ${name}
my name1 is Tom
my name2 is Tom*/
```

b. Naming:

1. Valid name
2. Letters,digit,underscore _
3. Groovy is a case sensitive
4. Groovy is a dynamically typed lang
5. Multiple assignments is possible

```
//2.naming convention

//Valid name of variable is
//Letters,digit,underscore _
def A = 10
def a1 = 20
def _var = "Variable"
println A+" "+a1+" "+_var

//output:
/*10 20 Variable*/

//grovy is a case sensitive lang
def x =10
def X =20
println x
println X
//output:
/*10
20*/

//Groovy is a dynamically typed lang
def name_1 ="Tom"
name_1 =10
println name_1
//output:
/*10*/
```

c. Multiple assignments

```
//3.Multipe assignment
def(a,b,c)=[30,40,50]
println a
println b
println c
//output:
/*30
40
50*/

def(String d,int e,double f)=[30,40,50]
println d
println e
println f
//output:
/*30
40
50.0*/

//will give error
def(String g,int h,double i)=[30,40]
println g
println h
println i
//output:
/*Caught: org.codehaus.groovy.runtime.typehandling.GroovyCastException: Cannot cast object
'null' with class 'null' to class 'double'. Try 'java.lang.Double' instead
org.codehaus.groovy.runtime.typehandling.GroovyCastException: Cannot cast object 'null' with
class 'null' to class 'double'. Try 'java.lang.Double' instead
        at groovy_variable.run(groovy_variable.groovy:76)*/
```

d. In Groovy can create multiline strings. by enclosing the String in triple quotes.

e. You can still variable types like byte, short, int, long, etc with Groovy. But you cannot dynamically change the variable type as you have explicitly declared it.

```
//4.multipe string
def str = """Groovy
at
Guru99"""
println str
//output:
/*Groovy
at
Guru99*/

//5.Can't change dynamically variable data type
int int_x = 104
int_x = "Guru99"
println int_x
//output:
/*Caught: org.codehaus.groovy.runtime.typehandling.GroovyCastException:
Cannot cast object 'Guru99' with class 'java.lang.String' to class 'int'
org.codehaus.groovy.runtime.typehandling.GroovyCastException: Cannot
cast object 'Guru99' with class 'java.lang.String' to class 'int'
        at groovy_variable.run(groovy_variable.groovy:114)*/
```

6. Data Type

Groovy lang have the following data type

1. Byte
2. Short
3. Integer
4. Long
5. Double
6. Float
7. Char
8. String

```
//Data type
//byte
byte a =10
println a
println Byte.MIN_VALUE
println Byte.MAX_VALUE
//output
/*10
-128
127*/

//short
short b =10
println b
println Short.MIN_VALUE
println Short.MAX_VALUE
//output
/*10
-32768
32767*/

//integer
int c =10
println c
println Integer.MIN_VALUE
println Integer.MAX_VALUE
//output
/*10  -2147483648 2147483647*/
```

```
//long
long d =10
println d
println Long.MIN_VALUE
println Long.MAX_VALUE
//output
/*10
-9223372036854775808
9223372036854775807*/

//double
double f =10
println f
println Double.MIN_VALUE
println Double.MAX_VALUE
//output
/*10.0
4.9E-324
1.7976931348623157E308*/

 //Boolean
 boolean h =true  println h
 //output /*true*/

//Extra
def j =10
println j
println j.getClass().getName()
//output /*10java.lang.integer*/
```

```
//float
float e = 10
println e
println Float.MIN_VALUE
println Float.MAX_VALUE
//output
/*10.0
1.4E-45
3.4028235E38*/


//char
char g ='a'
println g
//output
/*a*/

//String
String i ="Tom"
println i
//output
/*Tom*/

//Extra
def k =(byte)10
println k
println k.getClass().getName()
//output /*10 java.lang.Byte*/
```

7. Groovy Operator

Groovy has the following five types of operators

- Arithmetic operators: Add (+), Subtract (-), Multiplication (*), Division(/)
- Relational operators: equal to (==), Not equal to (!=), Less than (<) Less than or equal to (<=), Greater than (>), Greater than or equal to (>=)
- Logical operators: And (&&), Or(||), Not(!)

- Bitwise operators: And(&), Or(|), (^), Xor or exclusive-or operator
- Assignment operators: Negation operator (~)

For detailed information refer the following link

http://docs.groovy-lang.org/latest/html/documentation/core-operators.html

8. Groovy Conditional statement

Groovy support following two types of conditional statement

a. if,else
b. switch

1.if,else

```
//Conditional statement
//if else
def x =10
if(x==10){
println "x is equal to 10"
}
else if(x>10){
println "x is greater then 10"
}
else {
println "x is less then 10"
}

//output
/*
x is equal to 10
*/
```

2. switch case

The switch statement in Groovy is backward compatible with Java code; so you can fall through cases sharing the same code for multiple matches.

One difference though is that the Groovy switch statement can handle any kind of switch value and different kinds of matching can be performed.

```
//switch
def y = 1.23   def result = ""
switch ( y ) {
case "foo":
      result = "found foo"
      // lets fall through
case "bar":
      result += "bar"
case [4, 5, 6, 'inList']:
      result = "list"
      break
case 12..30:
      result = "range"
case Integer:
      result = "integer"
      break
case Number:
      result = "number"
       break
case ~/fo*/: // toString() representation of x matches the pattern?
     result = "foo regex"
      break
case { it < 0 }: // or { x < 0 }
      result = "negative"
      break
default:
      result = "default"}
println result
```

The switch supports the following kinds of comparisons:

- Class case values match if the switch value is an instance of the class
- Regular expression case values match if the toString() representation of the switch value matches the regex

- Collection case values match if the switch value is contained in the collection. This also includes ranges (since they are Lists)
- Closure case values match if the calling the closure returns a result that is true according to the Groovy truth
- If none of the above are used then the case value matches if the case value equals the switch value

> When using a closure case value, the default it parameter is actually the switch value (in our example, variable x).

9. Groovy looping Structure

Groovy support following looping structure

1.**for loop**

a. classic for
b. Enhanced classic Java-style for loop
c. Multi-assignment in combination with for loop

```
//loops
//1.for loop
//a.classic for
for(int i =1;i<=5;i++)
{println i}
//output
/*1 2 3 4 5*/

//b.Enhanced classic java-style for loop
def facts = []
def count = 5
for (int fact = 1, i = 1; i <= count; i++, fact *= i) {
   println fact}
//output
/*1 2 6 24 120*/

//c.Multi-assignment in combination with for loop
def list = []
for (def (String u, int v) = ['abc', 1]; v < 5; u++, v++) {
    list = "$u $v"}
println list
//output
/*abf 4*/
```

2. **for in loop**

a. iterate over a range
b. iterate over a list
c. iterate over an array
d. iterate over a map
e. iterate overvalues in a map
f. iterate over the characters in a string

Groovy also supports the Java colon variation with colons: for (char c : text) {}

```
//2.for in loop
//a.iterate over a range
for(i in 0..9)
{println i}
//output
 /*0 1 2 3 4 5 6 7 8 9*/

//c. iterate over an array
for(i in (0..4).toArray())
{println i}
//output
/*0 1 2 3 4*/


 //e.iterate over the characters in a string
 def text = "abc"
def l = []
def str=[]
for (c in text) {
    l.add(c)}
println l
//output
/*[a, b, c]*/
```

```
//b.iterate over a list
for(i in [0,1,2,3,4,5])
{println i}
//output
/*0 1 2 3 4 5*/


//d.iterate over a map
for(i in
[key1:"val1",key2:"val2
"]){
println i.key
println i.value}
//output
/*Key1 val1
key2 val2*/
```

3. while loop

```
//3.while loop
def A= 1
while(A<10){
println A
A++

}
//output
/*
1
2
3
4
5
6
7
8
9
*/
```

4.do/while loop

```
//4.do/while loop
def abc =0
do {
   abc++
} while(abc < 10)
println abc
//output
/*
1
2
3
4
5
6
7
8
9
10
*/
```

10. Strings

Groovy has the following type of strings

a. Single quoted '....'
b. Double quoted "...."
c. Triple single-quoted '''....'''
d. Triple double-quoted """...."""""
e. Slashy /..../
f. Dollar slashy $/..../$

```
//Strings
def name1 ="Tom" println name1
println "my name is "+name1
println "my name is".concat(name1)
println"my name is $name1"
println "my name is ${name1}"

def TripleQuotes = """this is groovy script
of Strings""" println TripleQuotes

def name2 ="Sam"
println name2.length()
println name2[1] println name2[-1]
println name2.indexOf('a')
println name2[0..2]  println name2[2..0]
println name2[0,2]
println name2.substring(2)
println name2.subSequence(0,1)

def str ="This is groovy script"
println str.split(" ") println str-("groovy ")
println str.replace("script","program")
println str.toLowerCase()
println str.toUpperCase()
println str.toList() println"groovy "* 3

println "Abc".equals("abc")
println "Abc".equalsIgnoreCase("abc")

def s3 ="My name is \"Justin\""
println s3
def s4 =/my name is "justin"/
println s4

def name3 = "Justin"
def s1 = / a blue ske $name3/
def s2 = $/ a green tree $name3/$
println s1 println s2

//output/*Tom
my name is Tom |my name isTom
my name is Tom |my name is Tom
this is groovy script
of Strings
3 | a |m |1 |Sam |maS|Sm|m|S
[This, is, groovy, script]
This is script
This is groovy program
this is groovy script
THIS IS GROOVY SCRIPT
[T, h, i, s,  , i, s,  , g, r, o, o, v, y,  , s,
c, r, i, p, t]
groovy groovy groovy
false
true
 a blue ske Justin
 a green tree Justin
My name is "Justin"
my name is "justin"*/
```

11.Methods

In the groovy method, we will consider the following parameter to understand

a. Method creation
b. Method parameter
c. Return type
d. Instance methods

```
//Methods
def naming(){
println "My name is Tom"}
naming()

def sum(int a ,int b){
println a+b}
sum(5,5)

def sub(int a,int b=5){
println a-b}
sub(10)

def mul(int a ,int b){
def c= a*b
return c}
def d = mul(2,3)
println d

class Test{
public static void main(String args){
Test test= new Test()
test.demo() }
def demo(){
println "this is a demo method"
}}
```

```
//output
/*My name is
Tom
10
5
6*/
```

12. Jenkins pipeline job with groovy

https://www.eficode.com/blog/jenkins-groovy-tutorial#:~:text=Jenkinsfile%20created%20by%20the%20classic,script%20in%20Jenkins%20script%20console.