What is Groovy?

Groovy is an optionally typed, dynamic language for the Java platform with many features that are inspired by languages like Python, Ruby, and Smalltalk, making them available to Java developers using a Java-like syntax.

Unlike other alternative languages,it's designed as a companion to , not a replacement for, Java.

Groovy is often referred to as a scripting language, and it works very well for script- ing. It's a mistake to label Groovy purely in those terms, though.
It can be precompiled into Java bytecode, integrated into Java applications, power web applications,add an extra degree of control within build files, and be the basis of whole applications on its own.

Groovy, obviously, is too flexible to be pigeonholed.

SEAMLESS INTEGRATION

The integration aspect of Groovy: it runs inside the JVM and makes use of Java's libraries (together called the Java Runtime Environment, or JRE). Groovy is only a new way of creating ordinary Java classes— from a runtime perspective, Groovy is Java with an additional JAR file as a dependency.

Consequently, calling Java from Groovy is a nonissue. When developing in Groovy, you end up doing this all the time without noticing. Every Groovy type is a subtype of `java.lang.Object` . Every Groovy object is an instance of a type in the normal way.

A Groovy date is a `java.util.Date` .You can call all methods on it that you know are available for a Date , and you can pass it as an argument to any method that expects a Date .

SYNTAX ALIGNMENT

Lets compare these two snippets :

```
import java.util.*;
Date today = new Date();
```

```
def date = new Date()
```

```
require 'date'
today = Date.new
```

```
import java.util._
var today = new Date
```

```
(import '(java.util Date))
(def today (new Date))
(def today (Date.))
```

```
import java.util.*
var date=Date()
```

They are similar the only difference is the optional typing which you can actually use the explicit type if you want another difference is the import statement you don't need it in groovy because groovy import java.util.* by default.

FEATURE-RICH LANGUAGE

Groovy has two main enhancements over and above those of Java:

- language features
- libraries specific to Groovy, and additions to the existing Java standard classes (known as the Groovy Development Kit, or GDK)

here are a few examples :

LISTING A FILE : CLOSURES AND I/O ADDITIONS

in groovy

```
def number = 0
new File('data.txt').eachLine { line ->
    number++
    println "$number: $line"
}
```

in java

```
final int[] number = {0};
try {
    Files.lines(Paths.get(filePath)).forEach(line -> {
        number[0]++;
        System.out.println(number[0] + ":" + line);
    });
} catch (IOException e) {
    e.printStackTrace();
}
```

PRINTING A LIST : COLLECTION LITERALS AND SIMPLIFIED PROPERTY ACCESS

in groovy

```
def classes = [String, List, File]

for (clazz in classes) {
    println clazz.package.name
}
```

in java

```
import java.io.File;
import java.util.List;

Class[] classes={String.class,List.class,File.class};
for(Class clazz : classes){
    System.out.println(clazz.getPackage().getName());
}
```

XML HANDLING THE GROOVY WAY : GPATH WITH DYNAMIC PROPERTIES

customers.xml

```
<?xml version="1.0" ?>
<customers>
    <corporate>
        <customer name="Bill Gates"
        <customer name="Tim Cook"
        <customer name="Larry Ellison"
    </corporate>
    <consumer>
        <customer name="John Doe" />
        <customer name="Jane Doe" />
    </consumer>
</customers>
```

in groovy

```
def customers = new XmlSlurper().parse(new File('customers.xml'))
for (customer in customers.corporate.customer) {
    println "${customer.@name} works for ${customer.@company}"
}
```

in java

```
try {
    DocumentBuilderFactory factory = DocumentBuilderFactory.newInstance();
    DocumentBuilder documentBuilder = factory.newDocumentBuilder();
    Document document = documentBuilder.parse(new File("customers.xml"));
    NodeList customers = document.getElementsByTagName("customer");
    for (int i = 0; i < customers.getLength(); i++) {
        Node item = customers.item(i);
        System.out.println(item.getAttributes().getNamedItem("name")
                + " works for "
                + (item.getAttributes().getNamedItem("company")));
    }
} catch (ParserConfigurationException | IOException | SAXException e) {
    e.printStackTrace();
}
```

Running Groovy

Table 1. Commands to execute Groovy

Command	What it does
`groovy`	Starts the processor that executes Groovy scripts. Single-line Groovy scripts can be specified as command-line arguments.
`groovysh`	Starts the groovysh command-line shell, used to execute Groovy code inter- actively. By entering statements or whole scripts line by line into the shell, code is executed on the fly.
`groovyConsole`	Starts a graphical interface that's used to execute Groovy code interactively; moreover, groovyConsole loads and runs Groovy script files.

Compiling and running Groovy

```
groovyc -d classes HelloWorld.groovy
```

the `groovyc` compiler outputs Java class files to a directory named `classes` , which you told it to do with the `-d` flag. If the directory specified with `-d` doesn't exist, it's created. When you're running the compiler, the name of each gener- ated class file is printed to the console.

For each script, Groovy generates a class that extends `groovy.lang.Script` , which contains a main method so that Java can execute it. The name of the compiled class matches the name of

the script being compiled. More classes may be generated, depending on the script code.

Running a compiled Groovy script with Java

Running a compiled Groovy program is identical to running a compiled Java pro- gram, with the added requirement of having the embeddable `groovy-all-*.jar` file in your JVM 's classpath, which will ensure that all of Groovy's third-party dependencies will be resolved automatically at runtime. Make sure you add the directory in which your compiled program resides to the classpath, too. You then run the program in the same way you'd run any other Java program, with the `java` command.

```
java –cp %GROOVY_HOME%/embeddable/groovy-all-2.4.0.jar;classes HelloWorld
```