Objectives

Key objectives of this chapter
- Basic Groovy Syntax
- Defining Functions
- Defining Classes
- Classes vs Scripts
- Defining Shared Libraries
- Using Shared Libraries

1.1 What is Groovy

- Groovy is an object oriented language
- It is based on Java platform.
- First released in 2007.
- It is distributed via the Apache License.

1.2 Groovy in Jenkins

- Groovy can be executed in Jenkins in various ways
  - Script Console (http://localhost:8080/script)
  - Execute system groovy script (Build step)
    - Runs in Jenkins' JVM.
    - Jenkins' system is available to the scripts
  - Execute groovy script (Build step)
    - Runs outside Jenkins' JVM.
    - Jenkins' system isn't available to scripts
  - Groovy PostBuild Plugin
  - Pipeline Job Command
1.3 Comments in Groovy

- Single-line comment
  - `//`
- Multi-line comment
  - `/*`
  - `*/`

1.4 Data Types

- Built-in types
  - byte
  - short
  - int
  - long
  - float
  - double
  - char
  - Boolean
  - String
- Object types
  - java.lang.Byte
  - java.lang.Short
  - java.lang.Long
1.5 Identifiers

- Identifiers are used to define variables, functions, or other user defined objects.
- Identifiers start with a letter, a dollar, or an underscore.
- They cannot start with a number.
- e.g.
  - `customer123name`, `_customer123Name`, `customer_name123`

1.6 Variables

- Case-sensitive
- Variable declarations
  - String `message = "Hello World";`
  - int `a = 5;`
- Printing variables
  - `println message;`
  - `println(message);`
  - `print message;`
  - `print(message);`
1.7 def

- def keyword can be used to define an identifier
- When using def, the actual type holder is **Object** (so you can assign any object to variables defined with def, and return any kind of object if a method is declared returning def)
- e.g.
  - def x = 5;

1.8 String Interpolation

- String interpolation requires double quotes.

```java
String name = "Bob";
String message = "Hi, ${name}";
```

1.9 Operators

- Arithmetic
  - +, -, *, /, %, ++, --
- Relational
  - ==, !=, <, <=, >, >=
- Logical
  - &&, ||, !
- Bitwise
  - &, |, ^, ~
- Assignment
Range
- `..`
- e.g. def range = 5..10
- println(range); //prints [5,6,7,8,9,10]
- println(range.get(2)); //prints 7

1.10 Ranges

Range examples
- 1..5 (inclusive range)
- 1..<5 (exclusive range)
- 5..1 (descending order)
- 'a'..'f' (character range)
- 'f'..'a' (descending order)

Methods
- contains(val)
- get(pos)
- size()
- subList(fromIndex, toIndex)
1.11 Conditional Statements

- **if**
  
  ```java
  if (a < 5) {
    println("A is less than 5");
  }
  ```

- **if / else**
  
  ```java
  if(a < 5) {
    println("A is less than 5");
  }
  else if(a == 5) {
    println("A is equal to 5");
  } else {
    println("A is greater than 5");
  }
  ```

- **switch**
  
  ```java
  switch(a) {
    case 1:
      println("A is 1");
      break;
    case 2:
      println("A is 2");
      break;
    default:
      println("Unknown");
      break;
  }
  ```
1.12 Loops

- for
  
  ```java
  for(int a = 0; a < 10; a++) {
      println(a);
  }
  ```

- for-in
  
  ```java
  String[] teams = ["Flames", "Maples", "Oilers" ];
  for(String team in teams) {
      println(team);
  }
  ```

- while
  
  ```java
  while(a < 5) {
      println(a);
      a++;
  }
  ```

- Additional keywords

  ◊ break
  
  ```java
  while(a < 5) {
      println(a);
      a++;
      if(a == 3) {
          break;
      }
  }
  ```

  ◊ continue
  
  ```java
  while(a < 5) {
      println(a);
      a++;
      if(a == 3) {
  ```
1.13 Lists

- List examples
  - [] // an empty list
  - [1, 2, 3, 4, 5]
  - [1, 2, [3, 4], 5] // a nested list
  - ['apple', 'banana', 'cherry'] // a list of strings
  - [1, 'apple', 2, 'banana'] // heterogeneous list

- Methods
  - list.add(val);
  - list.remove(val);
  - list.contains(val);
  - list.get(3);
  - list.isEmpty();
  - list.minus(anotherList);
  - list.plus(anotherList);
  - list.pop();
  - list.reverse();
  - list.size();
  - list.sort();
1.14 Maps

- Dictionary / table / hash
- Unordered collection of object references
- key / value pair
- e.g.
  - [:] // an empty map
  - teams = ['Calgary': 'Flames', 'Toronto': 'Maples']

- Processing all items in a map

```java
for(team in teams) {
    println(team);
}
```

- Methods
  - map.containsKey(key);
  - map.get(key);
  - map.put(key, value);
  - map.size();

1.15 Exception Handling

- try .. catch .. finally

```java
try {
    def arr = [1,2,3];
    def item = arr[7];
    println(item);
}
```
catch(ArrayIndexOutOfBoundsException ex) {
    println(ex.getMessage());
}
catch(Exception ex) {
    println("Some other exception");
}
finally {
    println("always executed");
}

1.16 Methods

- A method is defined with a return type or with the def keyword.
  - e.g.
    def sum(def a, def b) {
        return a + b;
    }

    def result = sum(5,6);
    - Default parameters
      def sum(def a = 5, def b = 6) {
        return a + b;
      }

      def result1 = sum(5);
      def result2 = sum();
    - Return keyword and semi-colon are optional, but recommended
      def sum(def a, def b) {
        a + b
      }
1.17 Closures

- e.g.
  ```groovy
def closure = {println "Hello world"};
closure.call();
```
- Parameters in closure
  ```groovy
def closure = {param -> println "Hello ${param}"};
closure.call("World");
```
- Multiple parameters in closure
  ```groovy
def sumClosure = {num1, num2 -> return num1 + num2};
closure.call(5,6);
```

1.18 this Keyword

- Used for accessing instance-level variable.
  ```groovy
class Customer {
    String name;
    public void SetName(String value) {
      this.name = value;
    }
    public String GetName() {
      return this.name;
    }
}
```

1.19 Classes

- A Groovy class is a collection of data and the methods that operate on that data
- A class declares the state (data) and the behavior (methods) of objects defined by that class.
• Getter and setter methods are used to implement encapsulation

• e.g.

```java
class Customer {
    String name;
    public void SetName(String value) {
        this.name = value;
    }
    public String GetName() {
        return this.name;
    }
}
```

• Instance creation

def cus1 = new Customer();

### 1.20 Static Methods

• Class level methods

• e.g.

```java
class MyClass {
    static def MyMethod() {
        println("Static method");
    }
}
```

### 1.21 Inheritance

• `extends` keyword is used to inherit the properties of a class.

```java
class Person {
    String name;
    public Person() {
```
this.name = "";
}
public void SetName(String value) {
    this.name = value;
}
public String GetName() {
    return this.name;
}
}

public class Student extends Person {
    public Student() {
        super();
    }
}

## 1.22 Abstract Classes

- Abstract classes represent generic concepts
- They cannot be instantiated
- They must be sub-classed
- e.g.

```java
abstract class Person {
    public String name;
    abstract void DisplayDetails();
}
```

```java
public class Customer extends Person {
    public Customer() {
        super();
    }
}
```
void DisplayDetails() {
    println "Details...";
}

1.23 Interfaces

- Defines a contract that a class needs to conform to.

```java
interface Vehicle {
    void Start();
}

class Car implements Vehicle {
    void Start() {
        println("Car.Start()");
    }
}
```

1.24 Generics

- Generalized classes
- Generic for Collections

```java
def list = new ArrayList<String>();
list.add("A");
```
- Generalized Classes

```java
public class MyClass<T> {
    private T localVariable;
    public T getVariable() {
        return this.localVariable;
    }
    public void set(T value) {
```

---
Canada
821A Bloor Street West
Toronto, Ontario, M6G 1M1
1 866 206 4644
getinfo@webagesolutions.com

United States
436 YorkRoad, Suite 1
Jenkintown, PA, 19046
1 877 517 6540
getinfousa@webagesolutions.com
1.25 Jenkins Script Console

- Manage Jenkins > Script Console
- Allows execution of adhoc scripts.
- Sample script:

```java
def printFile(location) {
    pub = new File(location)
    if (pub.exists()){
        println "Location \${location}"
        pub.eachLine{line-> println line}
    } else{
        println "$\{location\} does not exist"
    }
}
```

`printFile("C:/Windows/System32/drivers/etc/hosts")`

1.26 Extending with Shared Libraries

- Parts of Pipelines can be shared between various projects to reduce redundancies and keep code DRY (Don't Repeat Yourself).
- A Shared Library is defined with a name, a source code retrieval method such as by SCM, and optionally a default version.
- Version can be anything understood by the SCM, e.g. branches, tags, and commit hashes.
Library can be loaded implicitly or explicitly

1.27 Directory Structure

```
(root)
+- src          # Groovy source files
 |  +- org
 |   |  +- foo
 |   |     |  +- Bar.groovy # for org.foo.Bar class
+- vars
 |  +- foo.groovy # for global 'foo' variable
 |  +- foo.txt   # help for 'foo' variable
+- resources   # resource files (external libraries only)
 |  +- org
 |     |  +- foo
 |     |     |  +- bar.json # static helper data for org.foo.Bar
```

Can be defined at various levels

- `src` directory should look like standard Java source directory structure. This directory is added to the classpath when executing Pipelines.
- The `vars` directory hosts scripts that define global variables accessible from Pipeline. The basename of each `*.groovy` file should be a Groovy identifier, conventionally camelCased. The matching `*.txt` can contain documentation.
- The `resources` directory allows usage from an external library to load associated non-Groovy files.

1.28 Sample Groovy Code

```groovy
package com.abcinc;

def checkout() {
    node {
```
stage 'Checkout'
git url: 'C:\Software\repos\SimpleGreeting.git'

1.29 Defining Shared Libraries

- Shared libraries can be defined at various levels:
  - Global Shared Libraries
    - Manage Jenkins > Configure System > Global Pipeline Libraries
  - Folder-level Shared Libraries
  - Automatic Shared Libraries
    - e.g. GitHub Organization Folder
1.30 Using Shared Libraries

- Shared libraries can be utilized in various places
  - Pipeline
  - Execute system Groovy script
  - Execute Groovy script
  - Execute PostBuild script

- Loading libraries explicitly
  ```groovy
  @Library('my-shared-lib')
  @Library('my-shared-lib@master')
  @Library(['my-shared-lib', 'another-shared-lib'])
  
  Conventionally the annotation goes on an import statement
  ```
  ```groovy
  @Library('my-shared-lib')
  import com.abcinc.utils;
  ```

1.31 Same Shared Library Usage Code

```groovy
@Library('my-shared-lib')
import com.abcinc.utils;

def u = new utils();
u.checkout();
```
1.32 Defining Global Variables

// vars/acme.groovy
def setName(value) {
    name = value;
}
def getName() {
    return name;
}

// src/com/abcinc/sample.groovy
def myFunction() {
    name = "Bob";
}

1.33 Summary

- Groovy's syntax is similar to Java
- Semi-colon is optional in end of statement
- `return` keyword is optional in a method
- `def` keyword can be used to declare variables, as method return type, and for method input parameters.
- Groovy is used to define shared libraries
- In Jenkins shared libraries can be defined at various levels
- Shared libraries can be loaded implicitly or explicitly using @Library annotation.