Code Samples

Section 2: Groovy Operators

07 - Null-Safe Dereference Operator

```
@groovy.transform.ToString
class Job {
    String roleName
    int salary
}

@groovy.transform.ToString
class Person {
    String name
    Job job
}

@groovy.transform.ToString
class Department {
    String deptName = 'Platform Engineering'
    Set<Person> staff = [
        new Person(name: 'Matt', job: new Job(roleName: 'Developer', salary: 32000
)),
        new Person(name: 'Bhavin', job: new Job(roleName: 'Manager', salary: 38000
))
    ]
}

println new Department()
```

08 - Elvis Operator

```
def loggedInUser = 'Adam'  // null, '', ' '

def displayedUserName = loggedInUser ?: 'Guest'
```

09 - Spaceship Operator

```
'Agatha' <==> 'Molly'
```

```
'Molly' <==> 'Agatha'
```

```
'Molly' <==> 'Molly'
```

```
def l = ['black', 'white', 'blue', 'orange']
l.sort()
```

```
def l = ['black', 'white', 'blue', 'orange']
l.sort { a, b -> a <=> b }   // b <=> a
```

```
@groovy.transform.ToString
class FoodOrder {
    String name
    BigDecimal cost

    FoodOrder(name, cost) {
        this.name = name
        this.cost = cost
    }
}

def driveThruOrder = [
  new FoodOrder('Burger', 3.99),
  new FoodOrder('Fries', 1.85),
  new FoodOrder('Milkshake', 2.75)
]

driveThruOrder.sort()  // varies between runs
driveThruOrder.sort { a, b -> a.name <=> b.name }  // order by item name
driveThruOrder.sort { a, b -> a.cost <=> b.cost }  // order by item price
driveThruOrder.sort { a, b -> b.cost <=> a.cost }  // order by item price, most ex
pensive first
```

10 - Spread Operator

```
def fruits = ['Apples', 'Oranges', 'Pears']
def shoppingList = ['Milk, 'Cereal', *fruits]
def fruits = ['Apples', 'Oranges', 'Pears']

static void capitalizeThreeStrings(String a, String b, String c) {
    println 'A = ' + a
    println 'B = ' + b
    println 'C = ' + c
}

capitalizeThreeStrings(*fruits)
```

```
def fruits = ['apples', 'oranges', 'pears', 'oranges']

static void capitalizeThreeStrings(String... args) {
    args.each { println 'Element = ' + it }
}

capitalizeThreeStrings(*fruits)
```

```
def fruits = ['apples', 'oranges', 'pears', 'oranges']

fruits*.toUpperCase()  // new collection of all uppercased elements in original li
st
```

11 - Range Operator

```
1..5
```

```
(1..5).getClass()  // groovy.lang.IntRange
```

```
(1..5)[1]  // 2
```

```
(1..5).last()  // 5
```

```
(1..<5).each { println it }  // prints 1 to 4
```

```
enum Weekdays {
    MON, TUES, WEDS, THURS, FRI
}

(Weekdays.TUES..Weekdays.THURS).each { println it }
(Weekdays.TUES..<Weekdays.THURS).each { println it }
(Weekdays.TUES..Weekdays.THURS).getClass()  // groovy.lang.ObjectRange
```

```
(1.23).getClass()  // java.math.BigDecimal
```

```
(1.0..5.0).getClass()  // groovy.lang.ObjectRange
```

Section 3: Groovy Strings

13 - String Interpolation

```
def s = 'How are you?'
s.getClass()  // java.lang.String
```

```
def s = 'How are you? ' + 'Good?'
s.getClass()  // still a java.lang.String
```

```
def s = "How are you?"  // interpolation-ready string, but no interpolation needed
s.getClass()  // java.lang.String
```

```
def name = 'Matt'
def s = "How are you, $name?"  // interpolation being used, can also use ${name}
println 'String is: ' + s
s.getClass()  // org.codehaus.groovy.runtime.GStringImpl
```

14 - Heredocs

```
def emailText =
'''
Hi there!

Thanks for signing up, you're awesome!

Have a great day!

The Groovy Team
'''

emailText.getClass()  // java.lang.String
```

```
def name = 'Alex'

def emailText =
"""
Hi there, $name!

Thanks for signing up, you're awesome!

Have a great day!

The Groovy Team
"""

println 'Email:'
println emailText
emailText.getClass()  // org.codehaus.groovy.runtime.GStringImpl
```

15 - Pattern Operator in Regular Expressions

```
def re = 'S.*'  // Sugar, Sweet, Syrup

re.getClass()  // java.lang.String
```

```
def re = ~'S.*'  // Sugar, Sweet, Syrup

re.getClass()  // java.util.regex.Pattern
```

```
def re = ~'S.*'  // Sugar, Sweet, Syrup

def matcher = re.matcher('Sweet')

matcher.getClass()  // java.util.regex.Matcher
matcher.matches()  // true
```

```
def re = ~'l.*'  // Sugar, Sweet, Syrup

def matcher = re.matcher('Sweet')

matcher.matches()  // false
```

16 - Slashy Regular Expressions

```
def re = ~/S.*/  // Sugar, Sweet, Syrup

~'\d'  // doesn't compile since slash isn't escaped
~'\\d'  // compiles now but clunky
~/\d/  // forward slashes means you don't need to do the escaping now - nicer!
```

17 - Find Operator in Regular Expressions

```
def re = ~/S.*/
'Sugar' =~ /S.*/  // java.util.regex.Matcher

('Sugar' =~ /S.*/).matches()  // true
('Sugar' =~ /m.*/).matches()  // false
```

```
def matcher = 'Sugar' =~ /S.*/

matcher.matches()  // true
```

18 - Match Operator in Regular Expressions

```
def matches = 'Sugar' ==~ /S.*/  // true
def matches = 'Wednesday' ==~ /S.*/  // false
def matches = 'Sugar' ==~ /Sugar/  // true
```

19 - Capture Groups in Regular Expressions

```
def s = 'Friday is my favourite day!'

s =~ /(.*) is my favourite day!/  // true
s =~ (/(.*) is my favourite day!/)[0]  // list of input string, and first captured
 group (Friday)
s =~ (/(.*) is my favourite day!/)[0][1]  // Friday (the actual capture group)
```

```
def s = 'Monday is my favourite day!'

def dayOfWeek = (s =~ (/(.*) is my favourite day!/)[0][1]

println 'Day of week: ' + dayOfWeek
```

```
def s = 'Monday is my favourite day!'

def captures = (s =~ /(.*) is my favourite day!/)[0]
def dayOfWeek = captures[1]  // Monday
def whoseDay = captures[2]  // my

println 'Day of week: ' + dayOfWeek
println 'Whose day: ' + whoseDay
```

```
def s = /Sunday is Dave's favourite day!/

def captures = (s =~ /(.*) is my favourite day!/)[0]
def dayOfWeek = captures[1]  // Monday
def whoseDay = captures[2]  // my

println "$whoseDay best day of the week is $dayOfWeek"
```

Section 4: Collections in Groovy

21 - Creating Lists and Sets

```
def l = [1, 3, 5, 7]
l.getClass()  // java.util.ArrayList
```

```
def l = [1, 3, 5, 7] as LinkedList
l.getClass()  // java.util.LinkedList
```

```
def l = [1, 3, 5, 7] as Set
l.getClass()  // java.util.LinkedHashSet
```

```
def l = [1, 3, 5, 7] as HashSet
l.getClass()  // java.util.HashSet
```

```
def m = [a: 1, b: 5, f: 17]
m.getClass()  // java.util.LinkedHashMap
```

23 - Composing Collections of Different Types

```
def food = [fruites: ['apple', 'banana', 'ornage'], vegetables: ['potato', 'green
beans']]
food.getClass()  // java.util.LinkedHashMap
```

24 - Accessing Elements of a List

```
def l = ['a', 'b', 'c', 'e', 'f']
l.get(0)  // a
l[0]  // a
l[1..3]  // [b, c, e]
l[1..3].getClass()  // java.util.ArrayList
```

25 - Using Groovy Truthiness with Collections

```
def l = ['a', 'b', 'c', 'e', 'f']
l.size()  // 5
l.length  // not legal - this array method isn't implemented
l.isEmpty()  // false
```

```
def l = ['a', 'b', 'c', 'e', 'f']
boolean hasElements = l  // true
```

```
def l = []
boolean hasElements = l  // false
```

26 - Creating and Accessing Composite Collections

```
def m = [
    boys: ['Harry', 'Bill'],
    girls: ['Wendy', 'Sofia']
]

m['girls']  // Wendy, Sofia
m['boys']  // Harry, Bill

m['boys'][1]  // Bill
m['girls'][0]  // Wendy
```

27 - Processing Lists and Sets

```
def l = [1, 2, 3, 4, 5]

l.each { element -> println element }
l.each { element -> l.remove(element) }  // illegal - cannot modify while traversi
ng the list
```

28 - Processing Lists and Sets by Index

```
def l = [1, 2, 3, 4, 5]

l.eachWithIndex { el, idx ->
    println 'Current element = ' + el + ', Iteration # = ' + idx
}
```

```
def l = [1, 2, 3, 4, 5] as Set

l.eachWithIndex { el ->    // maintains insertion order even though it's a set (si
nce it's a linked set)
    println 'Current element = ' + el + '
}
```

29 - Processing Maps by Key and Value

```
def m = [
    'Monday': ['Record courses', 'Do exercise'],
    'Friday': ['Relax', 'Spend time with family', 'Walk dog']
]

m.each { k, v ->
    println "$k = $v"
}

for (entry in m) {
    println 'Key = ' + entry.key
    println 'Value = ' + entry.value
}
```

30 - Filtering Collections

```
def numbers = [1, 2, 3, 4, 5]
numbers.findAll { e ->
    e % 2 == 1
} // 1, 3, 5
numbers.findAll { e -> e % 2 == 1 }  // all on one line
numbers.findAll { it % 2 == 1 }  // using it
numbers.findAll { it % 2 == 1 }.getClass()  // java.util.ArrayList
```

```
def names = ['Tom', 'Dick', 'Harry']
names.findAll { it.startsWith('T') }  // Tom
```

```
class Person {
    String name
    int age
}

def namesAndAges = [
    new Person(name: 'Robert', age: 50),
    new Person(name: 'Simon', age: 45),
    new Person(name: 'Suzie', age: 32)
]
namesAndAges.findAll { it.age >= 40 }.each { println 'Name: ' + it.name }  // prin
ts Robert and Simon
```

31 - Finding the Matching Element in a Collection

```
class Person {
    String name
    int age
}

def namesAndAges = [
    new Person(name: 'Robert', age: 50),
    new Person(name: 'Simon', age: 45),
    new Person(name: 'Suzie', age: 32)
]
namesAndAges.find { it.age >= 40 }.name  // just the first match, Robert
```

```
class Person {
    String name
    int age
}

def namesAndAges = [
    new Person(name: 'Simon', age: 45),
    new Person(name: 'Robert', age: 50),
    new Person(name: 'Suzie', age: 32)
]
namesAndAges.find { it.age >= 40 }.name  // just the first match, Simon this time
```

32 - Testing Elements in a Collection

```
class Person {
    String name
    int age
}

def namesAndAges = [
    new Person(name: 'Simon', age: 45),
    new Person(name: 'Robert', age: 50),
    new Person(name: 'Suzie', age: 32)
]
namesAndAges.every { it.age >= 40 }  // false
```

```
class Person {
    String name
    int age
}

def namesAndAges = [
    new Person(name: 'Simon', age: 45),
    new Person(name: 'Robert', age: 50),
    new Person(name: 'Suzie', age: 65)
]
namesAndAges.every { it.age >= 40 }  // true
```

```
class Person {
    String name
    int age
}

def namesAndAges = [
    new Person(name: 'Simon', age: 45),
    new Person(name: 'Robert', age: 50),
    new Person(name: 'Suzie', age: 65)
]
namesAndAges.any { it.age >= 40 }  // true
```

```
class Person {
    String name
    int age
}

def namesAndAges = [
    new Person(name: 'Simon', age: 45),
    new Person(name: 'Robert', age: 50),
    new Person(name: 'Suzie', age: 32)
]
namesAndAges.any { it.age >= 40 }  // still true
```

```
class Person {
    String name
    int age
}

def namesAndAges = [
    new Person(name: 'Simon', age: 39),
    new Person(name: 'Robert', age: 30),
    new Person(name: 'Suzie', age: 32)
]
namesAndAges.any { it.age >= 40 }  // false
```

33 - Collecting Elements to a List

```
def l = [1, 2, 3, 4, 5]
l.collect { it * 2 }  // 2, 4, 6, 8, 10
```

```
def l = ['Matt', 'Alan', 'Gavin']
l.collect { it.toUpperCase() }  // MATT, ALAN, GAVIN
```

```
class Person {
    String name
    int age
}

def l = [
    new Person(name: 'Matt', age: 25),
    new Person(name: 'Alan', age: 32),
    new Person(name: 'Gavin', age: 34)
]
l.collect { it.name.toUpperCase() }  // MATT, ALAN, GAVIN
```

34 - Collecting Entries to a Map

```
class Person {
    String name
    int age
}

def l = [
    new Person(name: 'Matt', age: 25),
    new Person(name: 'Alan', age: 32),
    new Person(name: 'Gavin', age: 34)
]
l.collect { it.name.toUpperCase() }.getClass()  // java.util.ArrayList
```

```
class Person {
    String name
    int age
}

def l = [
    new Person(name: 'Matt', age: 25),
    new Person(name: 'Alan', age: 32),
    new Person(name: 'Gavin', age: 34)
]
l.collectEntries { [(it.age): it.name.toUpperCase()] }  // map keyed by age
```

35 - Creating Aggregate Functions with Inject

```
def l = [1, 2, 3, 4, 5]
l.sum()  // 15
l.min()  // 1
l.max()  // 5
```

```
def l = [1, 2, 3, 4, 5]
l.inject(0) { sum, e -> sum + e }  // returns sum
l.inject(0) { max, e -> Math.ax(max, e) }  // returns max
```

Section 5: Groovy Language Features

37 - Default Imports

```
def number = 1.23
number.getClass()  // java.math.BigDecimal
```

```
def number = 123
number.getClass()  // java.lang.Integer
```

```
def number = 123G  // can use lowercase g too as suffix
number.getClass()  // java.math.BigInteger
```

38 - Main Methods in Groovy Applications

```
public static void main(String[] args) {
    // code for app
}
```

```
public static void main(String[] args) {
    println 'Running'
}
```

```
public static void main(args) {  // don't need to specify types
    println 'Running'
}
```

```
static def main(args) {  // don't need to specify void and/or public either
    println 'Running'
}
```

```
def main(args) {  // this won't work - must always be static
    println 'Running'
}
```

39 - Creating Classes and Instances in Groovy

```
class Person {

}

public static void main(String[] args) {
    Person p = new Person
}
```

40 - Adding Methods and State to Groovy Classes

```
class Person {

}

public static void main(String[] args) {
    Person p = new Person
    p.greet()  // groovy.lang.MissingMethodException
}
```

```
class Person {
    String name

    Person(String name) {
        this.name = name
    }

    void greet() {
        println "Hello, I'm $name!"
    }
}

public static void main(String[] args) {
    Person p = new Person('Matt')
    p.greet()
}
```

```
class Person {
    String name

    Person(name) {  // we've dropped the type - legal in Groovy!
        this.name = name
    }

    def greet() {  // we're using def instead of the return type (weak typing)
        println "Hello, I'm $name!"
    }
}

public static void main(String[] args) {
    Person p = new Person('Matt')
    p.greet()
}
```

41 - Using Inheritence in Groovy

```
class Person {
    String name

    Person(name) {  // we've dropped the type - legal in Groovy!
        this.name = name
    }

    def greet() {  // we're using def instead of the return type (weak typing)
        println "Hello, I'm $name!"
    }
}

class Emnployee extends Person {
    int salary

    def reportForWork() {
        "Here sir, ready and willing for the long day ahead!"
    }
}

public static void main(String[] args) {
    Person p = new Employee('Matt')
    p.greet()
}
```

```
class Person {
    String name

    Person(name) {  // we've dropped the type - legal in Groovy!
        this.name = name
    }

    def greet() {  // we're using def instead of the return type (weak typing)
        println "Hello, I'm $name!"
    }
}

class Emnployee extends Person {
    int salary

    Employee(String name) {
        super(name)
    }

    def reportForWork() {
        "Here sir, ready and willing for the long day ahead!"
    }
}

public static void main(String[] args) {
    Employee e = new Employee('Matt')
    e.greet()
    println e.reportForWork()
}
```

42 - Overriding Methods in Groovy

```
class Person {
    String name

    Person(name) {  // we've dropped the type - legal in Groovy!
        this.name = name
    }

    def greet() {
        println "Hello, I'm $name!"
    }
}

class Emnployee extends Person {
    int salary

    Employee(String name, int salary) {
        super(name)
        this.salary = salary
    }

    def reportForWork() {
        "Here sir, ready and willing for the long day ahead!"
    }

    def greet() {
        println "Hello, I'm $name, I earn $salary"
    }
}

public static void main(String[] args) {
    Employee e = new Employee('Matt', 25000)
    e.greet()
    println e.reportForWork()
}
```

43 - POGOs and Groovy Property Generation

```
class Person {
    String name
}
```

44 - Operator Overloading

```
String s = 'a'
+s  // groovy.lang.MissingMethodException for String.positive()
```

```
class Greeting {
    String message
}

Greeting g = new Greeting(message: 'Hello')
+g  // groovy.lang.MissingMethodException for Greeting.positive()
```

```
@groovy.transform.ToString
class Greeting {
    String message

    Greeting positive() {
        return new Greeting(message: this.message.toUpperCase())
    }
}

Greeting g = new Greeting(message: 'Hello')
+g  // Greeting with HELLO in it
```

45 - String Equality in Groovy

```
String s1 = 'Hello'
String s2 = 'Hi'

s1.equals(s2)  // logical equality in Java
s1 == s2  // reference equality in Java, but logical equality in Groovy!
```

46 - Returning Multiple Values from a Method

```
@groovy.transform.ToString(includeNames = true)
class BoxDimensions {
    int x, y, z
}

static BoxDimensions calculate() {
    // do some calculation...
    // ...then return the dimensions
    new BoxDimensions(x: 10, y: 12, z: 30)
}

public static void main(String[] args) {
    BoxDimensions dimensions = calculate()

    // calculation 1
    int area = dimensions.x * dimnensions.y

    // calculation 2
    // calculation 3
    //       :
}
```

```
static def calculate() {  // use weak typing
    // do some calculation...
    // ...then return the dimensions
    [10, 12, 30]
}

public static void main(String[] args) {
    def (x, y, z) = calculate()

    // calculation 1
    int area = x * y

    // calculation 2
    // calculation 3
    //       :
}
```

47 - Autogenerating Equals and HashCode with Groovy AST Transformations

```
class Person {
    String name
    int age
}

public static void main(String[] args) {
    new Person()  // just prints the default 'address' string representation of th
e object
}
```

```
import groovy.transform.ToString

@ToString   // can also use fully qualified classname to avoid import statement
class Person {
    String name
    int age
}

public static void main(String[] args) {
    new Person()  // has nice string representation now
}
```

```
import groovy.transform.ToString
import groovy.transform.EqualsAndHashCode

@ToString
@EqualsAndHashCode
class Person {
    String name
    int age
}

public static void main(String[] args) {
    new Person()  // has hashCode and equals methods on it too now autogenerated b
y AST
}
```

48 - Named Constructors

```
class Person {
    String name
    int age
}

static void main(String[] args) {
    new Person(name: 'Daisy', age: 5)  // using named constructors
}
```