Class 4:

**Simple Data Types:**

Java Data Types

**Primitive type :**

1. byte (Wrapper Type : java.lang.Byte)
2. short
3. int
4. long
5. float
6. double
7. char
8. boolean

**Groovy Data Types:**

In groovy everything is object.

byte b = 10  
b.getClass().get.Name()​

Result : java.lang.Byte

So the variable b is instance of class “java.lang.Byte”; which is associated wrapper class for Byte.

In java we have to declare variable type before we can assign a value .

In groovy we have an option to explicitly declaring a variable using **def** Keyword; but this not mandatory.

**Numbers :**

// Groovy Number Defaults   
//------------------  
  
def number = 10   
println number.class  
  
def decimal = 5.50  
println decimal.class   
  
  
//:: Converting Data Types ::  
//---------------------------  
// Explicit Casting   
def myFloat = (float)1.0  
println myFloat.class   
//here it will be part of java.lang.float. Because we explicitly told the compiler that even though you wanna default it to BigDecimal but I want it to be float  
  
//Implicit - Coercion  
​//Rules for +, - , \*   
//-------------------------------------  
  
//If either operator is a float or double then result will be double   
//In Java if only floats are involved the result is a float  
  
  
//If either operand is a big decimal then the result will be BigDecimal  
//If either operand is a BigInteger the result is a BigInteger   
//If either operand is a Long the result is a Long  
//If either operand is a Integer result is an Integer in Java. To achieve same in groovy, we should use method intdiv (println 1.intdiv(2))   
  
  
//Integer Division   
def intDiv = 1/2  
println intDiv // this is much different than java where we would get 0   
println intDiv.getClass().getName() //this will be java.math.BigDecimal  
Println 1.intdiv(2) /// we can use method intdiv to get results in integer  
  
  
  
//GDK Methods : (Methods for number)  
assert 2 == 2.5.toInteger() //conversion  
assert 2 == 2.5 as Integer //enforced coercion  
assert 2 == (int) 2.5 //cast  
  
  
assert '5.50'.isNumber()  
assert 5 =='5'.toInteger()​​​​​​​​​​  
  
  
  
//times | upto | downto | step   
20.times {  
   print​ '-'  
}​​​  
  
1.upto(10) { num ->   
   ​println ​ num  
}  
  
  
10.downto(1) { num ->   
println num  
}   
  
  
0.step(1, 0.1) ​{ num ->   
println num  
} // we are stepping from 0 to 1 by value 0.1​​​​​​​​​​​​​​​​​​​​​​​​​​​​​​​​​​​​​​​​​​​​​​​​​​​​​​​​​​​​​

**Operator Overloading :**

Groovy supports operator overloading for a limited set of operators. Each supported operator corresponds to a particular method signature. If a type implements that method, then that operator is effectively overloaded for instances of that type.

@groovy.transform.ToString  
class Account {  
    BigDecimal balance = 0.0  
    String type //checking or saving   
  
    BigDecimal deposit(BigDecimal amount){  
        balance += amount  
    }  
  
    BigDecimal withdraw(BigDecimal amount){  
        balance -= amount  
    }   
  
    BigDecimal plus(Account account){  
        this.balance + account.balance  
    }  
}  
  
Account checking = new Account(type:"Checking")​​​​​​​​​​​​​​​​  
checking.deposit(100.00)  
  
​Account saving= new Account(type:"Saving")  
saving.deposit(500.00)  
  
println checking   
println saving  
  
BigDecimal total = checking + saving  
println total  
​

**Strings:**

//character definition  
char c = 'c'  
println c  
println c.class  
  
  
//string definition  
String str = 'this is string'  
println str  
println str.class​​​​​​​​​​​​​  
  
//by default condition will be string   
def c2 = 'c'  
​println c2.class​

We have two types of Strings in groovy

Plain strings and G Strings

​//plain string is instances of java.lang.String   
//G string (groovy string) is are instances of groovy.lang.String  
  
//string interpolation   
String name = "Rakesh"  
String msg = "Hello " + name + "..."​​​​​​​​​​​​​​​​​​​​​​​​​​​​​​​​​​​​  
println msg  
//Java way of doing   
  
String msg2 = "Hello ${name}"  
println msg2  
//groovy way of doing  
//so here we evalute our message at runtime. and this happens only if use double quotes  
  
  
//multiline string  
def msg3 = '''  
hi   
Rakesh  
yadav  
'''  
​def msg4 = """  
hi   
Rakesh  
yadav  
"""  
// here we can use expression as well; which will be evaluated at runtime  
  
  
  
  
  
//dollar slashy  
​def folder = $/C:\groovy\learning/$  
println folder​​​​​​​​​​​​​​​​​​​​​​​​​​​​​​​​​​​​​​​  
//result: unexpected char: '\' @ line 35, column 17.  
//if we replace single slash by double slash problem will be solved. but will use dollar slashy  
​