Groovy

1. Basic syntax

```
class Example {
    static void main(String[] args) {
        // 1. Semicolons is not require
        // 2. Identifier using 'def'

        def x = 1
        println('Hello World')
    }
}
```

2. Data Types

Built-in Data Types

Groovy offers a wide variety of built-in data types. Following is a list of data types which are defined in Groovy –

- **byte** – This is used to represent a byte value. An example is 2.
- **short** – This is used to represent a short number. An example is 10.
- **int** – This is used to represent whole numbers. An example is 1234.
- **long** – This is used to represent a long number. An example is 10000090.
- **float** – This is used to represent 32-bit floating point numbers. An example is 12.34.
- **double** – This is used to represent 64-bit floating point numbers which are longer decimal number representations which may be required at times. An example is 12.3456565.
- **char** – This defines a single character literal. An example is 'a'.
- **Boolean** – This represents a Boolean value which can either be true or false.
- **String** – These are text literals which are represented in **the form** of chain of characters. For example "Hello World".

Bound values

The following table shows the maximum allowed values for the numerical and decimal literals.

byte	-128 to 127
short	-32,768 to 32,767
int	-2,147,483,648 to 2,147,483,647
long	-9,223,372,036,854,775,808 to +9,223,372,036,854,775,807
float	1.40129846432481707e-45 to 3.40282346638528860e+38
double	4.94065645841246544e-324d to 1.79769313486231570e+308d

3. Variables

Variables in Groovy can be defined in two ways – using the **native syntax** for the data type or the next is **by using the def keyword**.

```
class Example {
   static void main(String[] args) {
      // Defining a variable in lowercase
      int x = 5;

      // Defining a variable in uppercase
      int X = 6;

      // Defining a variable with the underscore in it's name
      def _Name = "Joe";

      println(x);
      println(X);
      println(_Name);
   }
}
```

4. Operator & Control Flow

```
// Range
def range = 5..10;
println(range);
println(range.get(2)); // Return 7


static void control(){
    int count = 0;
    while(count<5) {
        println(count);
        count++;
    }
    for(int i = 0;i<5;i++) {
        println(i);
    }

    int[] array = [0,1,2,3];
    for(int i in array) {
        println(i);
    }

    def employee = ["Ken" : 21, "John" : 25, "Sally" : 22];

    for(emp in employee) {
        println(emp);
    }

}
```

5. Methods

```
static def DisplayName() {
    println("This is how methods work in groovy");
    println("This is an example of a simple method");
}
// Default parameter
def someMethod(parameter1, parameter2 = 0, parameter3 = 0) {
    // Method code goes here
}
```

6. Groovy - Optionals

Groovy is an "**optionally**" typed language, and that distinction is an important one when understanding the fundamentals of the language. When compared to Java, which is a "**strongly**"

typed language, whereby the compiler knows all of the types for every variable and can understand and honor contracts at compile time. This means that method calls are able to be determined at compile time.

Optional typing can be a powerful utility during development, but can lead to problems in maintainability during the later stages of development when the code becomes too vast and complex.

To get a handle on how you can utilize optional typing in Groovy without getting your codebase into an unmaintainable mess, it is best to embrace the philosophy of “duck typing” in your applications.

If we re-write the above code using duck typing, it would look like the one given below. The variable names are given names which resemble more often than not the type they represent which makes the code more understandable.

7. Strings

```
    static void main(String[] args) {
        String sample = "Hello world";
        println(sample[4]); // Print the 5 character in the string

        //Print the 1st character in the string starting from the back
        println(sample[-1]);
        println(sample[1..2]);//Prints a string starting from Index 1 to 2
        println(sample[4..2]);//Prints a string starting from Index 4 back to 2
    }
```

8. Date

```
class Example {
   static void main(String[] args) {
      Date date = new Date();

      // display time and date using toString()
      System.out.println(date.toString());
   }
}
```

9. Exception

```
      try {
         def arr = new int[3];
         arr[5] = 5;
      }catch(ArrayIndexOutOfBoundsException ex) {
         println("Catching the Array out of Bounds exception");
      }catch(Exception ex) {
         println("Catching the exception");
      }

      println("Let's move on after the exception");
```

10. Traits

Traits are a structural construct of the language which allow –

- Composition of behaviors.
- Runtime implementation of interfaces.
- Compatibility with static type checking/compilation

They can be seen as interfaces carrying both default implementations and state. A trait is defined using the trait keyword.

```
trait Marks {
    int currentMarks
    void displayMarks(){
        println("My mark is " + currentMarks)
    }
}

class Student implements Marks {
    int StudentID
    int Marks1;


}

class TraitMain {
    static void main(String[] args) {
        Student student = new Student()
        student.currentMarks = 10
        student.displayMarks()
    }
}
```

11. Closure

A closure is a short anonymous block of code. It just normally spans a few lines of code. A method can even take the block of code as a parameter. They are anonymous in nature.

Following is an example of a simple closure and what it looks like.

```
static void main(String[] args) {
    def test = {params -> println "Hello ${params}"}
    test.call("Duy") // Hello Duy
}
```

12. Difference between java and groovy

1. No semicolons
2. Return keyword is optional
3. Def and type, two way to declares new variables
4. Optional method paramater type
5. Public by default (class and method)
6. Parenthesis sometimes is optional
7. Getter and setter is automatically created
8. Using **with** to operate common operation on the same bean
9. == in java is "is" in groovy, meanwhile == in groovy is equal in java
10. Switch is allow multiple type
11. Import aliasing
12. Check null can be replace with "?"