**Using Properties:**

* First you want to create properties as:
  + Global
  + Project
  + Suite
  + Test Case
* Then you go into the test step and use the following format:
  + If Global:
    - $(GlobalPropertyName)
  + If anything, else:
    - $(#LevelOfProperty#NameOfProperty)

**DataSource Loop & Sink:**

* Create test steps
* Add in the function you want to create a loop of
* Then add a DataSource right after the function
* Create an excel sheet and starting at A1, input numbers with each column being a different variable
* Add the excel sheet as the input file, making sure the sheet name is specified and the data starts from A1
* Go to the variables and tell it to get the data from the DataSource, with the defined variable
* Create a DataSource Loop
* Double click the loop and define:
  + DataSource Step = Name of Data Source
  + Target Step (starting point) = The function
  + Make sure loop is at the end
* Create a DataSink to store the data
* Create variable in DataSink that pertains to information you want to store
* Go to the output you got, and then right click and store it into the DataSink variable
* Put that before the loop ends, with the input and output being from the excel sheet as well

**XPath Assertion:**

* Go to test case assertion
* Add in a New 🡪 Property Content 🡪 XPath Match
* Declare
* Use the following syntax at next fresh line:
  + “//ns1:[now add in the HTML path]
  + Use single slash “/” if you are looking at the exit child
    - That is the path DIRECTLY after previous path
  + Use double slash “//” if you want to look anywhere inside the child path

**XQuery:**

* Used to validate multiple data
* Getting query responses – like SQL getting information from database
* Go to test case 🡪 Assertion 🡪 Add new 🡪 Property Content 🡪 XQuery Match
* Declare the variable
* Use the following format:
  + <[Name of output you want]>
  + {
  + for $a (this is a variable we are declaring) in //ns1:[Path1]/ns1:[Exit Child]/text()
  + return $a
  + }
  + </[Name of output you want]>
* If you want to use an “if…then” statement you can use the following format:
  + <MultiplyResult>
  + {
  + for $a in //ns1:MultiplyResponse/ns1:MultiplyResult/text()
  + return
  + if ($a="14") then " AssertionPassed"
  + else "<fail>AssertionFailed</fail>"
  + }
  + </MultiplyResult>

**Groovy Scripts:**

* Define a variable with “def”
  + def i = 100
* Write an output with log.info
  + log.info “Text”
* Other acceptable Java scripts:
  + If-elseif-else
  + If-else
  + For (I, i<1, i++)
  + While
    - While (i<10)
    - Log.info i
    - I++
* Array:
  + def array = [5, 10, 15]
  + log.info arr[0] (you will get 5 here)
  + OR…….
  + Def array = new int[4]
    - This means we made a new array that has 4 values
* Special For loop:
  + For (def a : arr)
  + Log.info a
  + This means we defined a variable called “a”
  + It will then keep changing as it cycles through the max value of array values from “arr”
  + Then we display the “a” value each time with “log.info”
* Create properties and get them
  + testRunner.testCase.testSuite.project.setPropertyValue(“[Name for Property]”, “[Value you want for property]”)
  + get it via: getPropertyValue(“[Name for Property]”)
* Length:
  + S.length() = get number of characters
* Trim:
  + S.trim() = Take away spaces before and after the strings
* Upper/Lower case:
  + S.toUpperCase()
  + S.toLowerCase()
* Replace:
  + S.replace(“[What to Change]”, “[New]”)
  + Many words = double quotes
  + Single characters = single quotes
* Concatenation:
  + Helps you combine multiple strings
  + S.concat(“[What To Add on]”).concat(“[Next Thing to add]”)
* Find character at specific location:
  + Index starts at 0
  + S.charAt(#)
* Find part of a string
  + S.substring(Where to Start, Where to end + 1)
  + Make sure you do +1 at the end because that is how the method is called – they take the number and then minus one
* Ignoring case:
  + S.equalsIgnoreCase()
* Compare to:
  + S.compareTo(s1) == 0
  + If they are the same, the answer will come out to 0
  + Can also use compareToIgnoreCase
* Contains:
  + Checks if a part of the string contains anything
  + S.contains(“Test”)
  + If you want to do lower case:
    - S.toLowerCase().contains(“test”)
* Split methods:
  + S.split(“ “)
  + This means we will create a split everything whatever is in the quotes occurs
  + In this case, we will cause a break after each space
  + The index starts at 0
  + We want to define new one for the splits usually so:
  + Def s1 = s.split(“ “)
* Displaying all the splits:
  + For (def i=0; i<1; i++) {
  + Println s1[i]

**Automation:**

* Create a test case
* Start with a smoke test, that checks for each of the different quest listed in WSDL file
* Create an excel sheet with following columns:
  + Step ID = Step1…Step2
  + Step Name = Name of request
  + Status = Y/N if you are going to run it
  + Data = The data that will be passed through the request
* Add in a groovy script and make it the first step
* Make sure to import:
  + Import java.io.\*
  + Import jxl.\*
* Def f = new File(“[Location”)
* Def wk = Workbook.getWorkbook(f)
* Def ws = wk.getSheet(“Sheet1”)
* R = ws.getRows()
  + //For the rows, the index starts at 0, which in our case is the header so we get data starting at 1, which is why we start at 1 below
* For (def I =1; i<r; i++) {
  + Cell c1 = ws.getCell(2,i)
    - //The 2 is for the 3rd column, again because the first column index is at 0
  + If (c1.getContents().equalsIgnoreCase(“Y”)) {
    - Cell c2 = ws.getCell(3,i)
    - testRunner.testCase.testSuite.setPropertyValue(“testData”, c2.getContents())
    - Cell c3 = ws.getCell(1,i)
    - testRunner.runTestStepByName(c3.getcontents())
      * //runTestStepByName lets you look at whatever is listed, which in our case is the contents of c3, and then running it based on that name
* Make sure you go to the individual request and tell it to get the data from the test suite

**Command Prompt Automation:**

* Open up CMD
* Go to folder where SoapUI is installed – in the bin folder
* Type:
  + Testrunner.bat -s[NameOfSuite] “[Location of Project]”

**WebService Mocking:**

* This is the work you do before the functionality is working
* You set up the framework so when the services come online, you are ready

**Debugging:**

* Process to find out where the issue is occurring in the code
* When you go to the Test Suite, you then move over and click on debugging
* This allows you to stop steps and figure out the properties for each and analyze what is causing the issues
* You can also make it conditional, so it only causes a break if something specific occurs

**Setup Script:**

* This is used to make a script occur BEFORE the project/suite/cases start
* Such as setting an environment, so the test steps only occur in like the QA or Mock environments

**Conditional Execution:**

* testRunner.runTestStepByName(“[Test-Name]”)
* This allows you to create an “if-then” statement and then use the above line to run the test case only if a certain situation occurs

**SoapUI Functions:**

* Get = retrieve data from server
* Post = add new data
* Put = update data
* Delete = get rid of data
* Patch = Only update SUBSET of data
  + Head data = only beginning part of data
  + Options = figure out what functions work with the specific API
  + Trace = Detail all the method being sent to the server