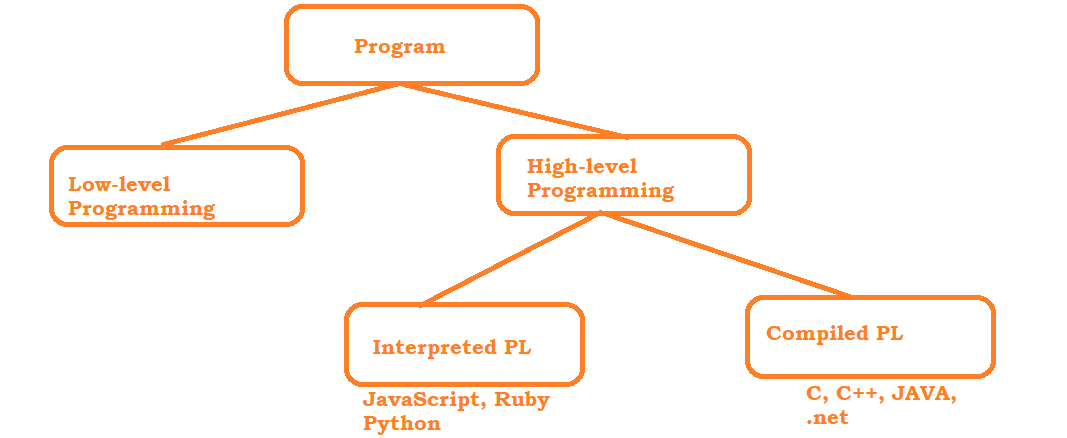
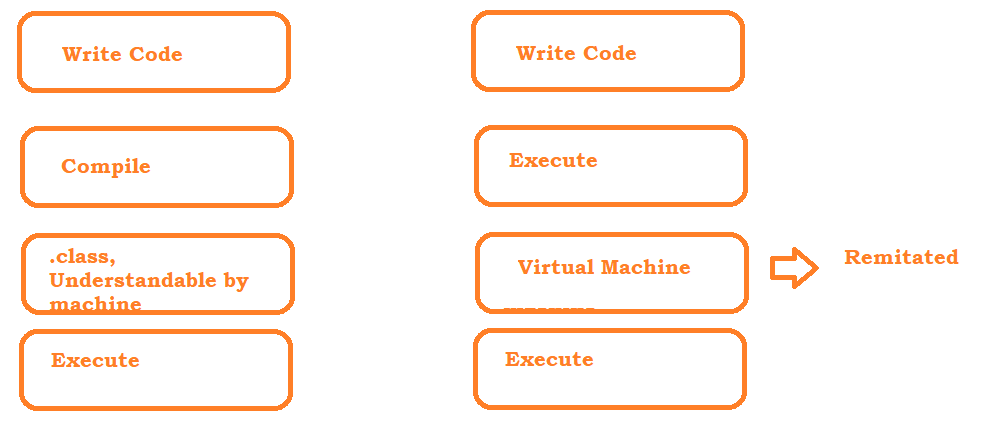
Java Script

Java Script is a high level, Object oriented and Interpreted programming language



# High level Difference :



# Components of Interpreter

# 

## Executing Java Script on Browser

Open browser -> press 12 -> go to console 🡪 execute javascript code

## Executing Java Script on Windows Machine

* Install nodejs on your machine
  + <https://nodejs.org/en/download/package-manager>
  + Open power shell and execute below commands

**# installs fnm (Fast Node Manager)**

* + winget install Schniz.fnm

**download and install Node.js**

* + fnm use --install-if-missing 20

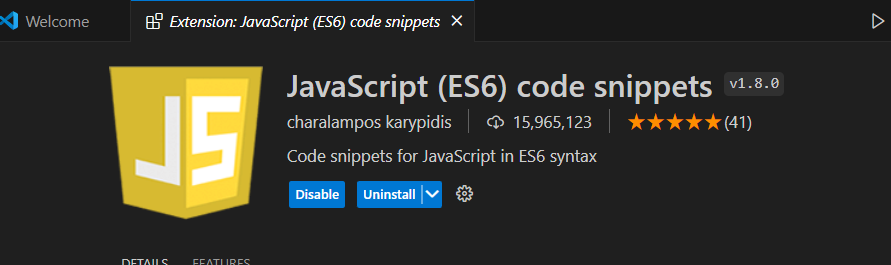
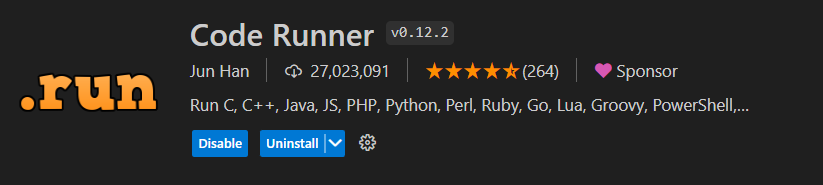
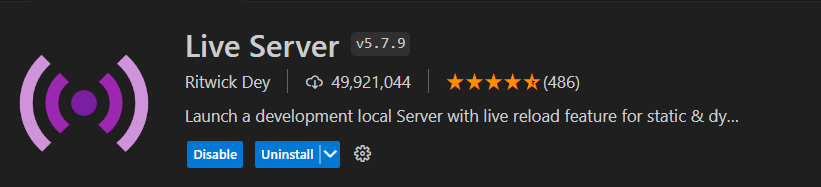
**# verifies the right Node.js version is in the environment**

* + node -v # should print `v20.14.0`

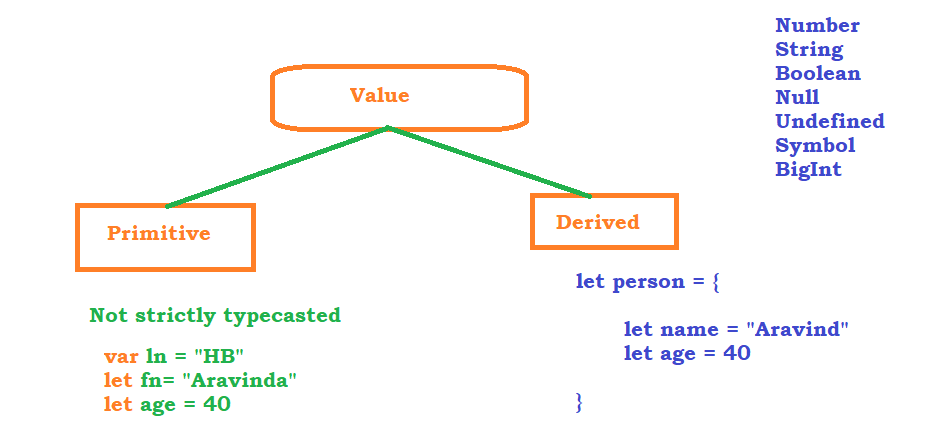
**# verifies the right NPM version is in the environment**

* + npm -v # should print `10.7.0`

Visual Studio Code Editor

* Download the software : <https://code.visualstudio.com/download>
* Follow the instructions OR installation steps
* Add Extensions :
  + 
  + 
  + 

# Data types and variables :



Operators in JS

# Arithmetic Operators

* Addition -> +
* Subtraction -> -
* Multiplication -> \*
* Division -> /
* Module -> %
* Exponential -> \*\*
* Increment -> ++

# Assignment Operators

* Assign = x=y
* Add and Assign += x+=y x = x+y
* Sub and Assign -= x-=y x = x-y
* Mul and Assign \*= x\*=y x = x\*y
* Div and Assign /= x/=y x = x/y
* Mod and Assign %= x%=y x = x%y
* Exp and Assign \*\*= x\*\*=y x = x\*\*y

# Comparison Operators

* Check equal value == x==y
* check equal value and type === x===y
* Check not equal value != x!=y
* check not equal value and type !== x!==y
* Greater than > x>y
* Less than < x<y
* Greater than equal >= x>=y
* Less than equal <= x<=y
* Ternary Operator ?: (x>y)?x:y

# Logical Operators

* Logical AND && x &&y
* Logical OR || x||y
* Logical NOT ! !x

# Type of Operators

* typeOf
* instanceOf

Conditional Statements

* if
* if, else
* if, elseif, .. else
* switch

Looping Statements

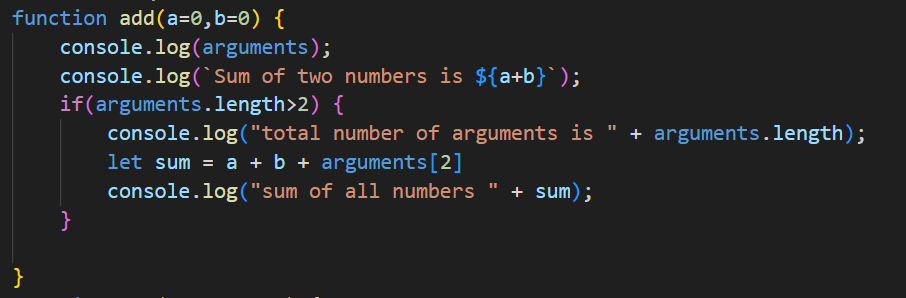
* for
* for each
* while
* do while
* for in
* for of

Arrays

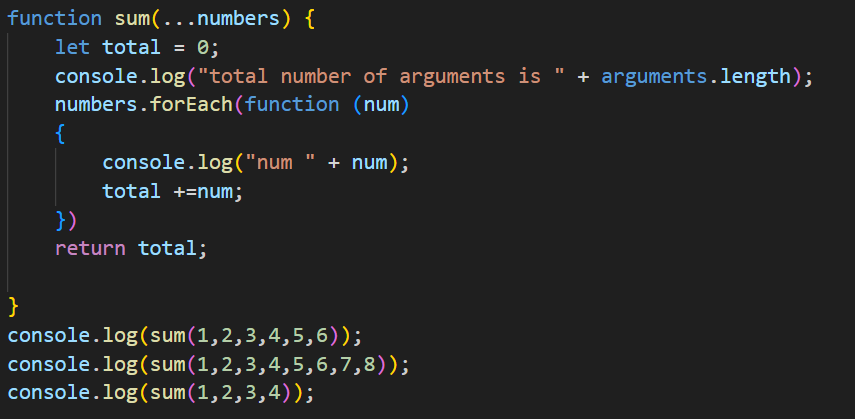
* length
* push
* pop
* shift
* unshift
* slice
* splice
* concat
* filter
* foreach
* find
* sort
* multidimentional arrays

Functions

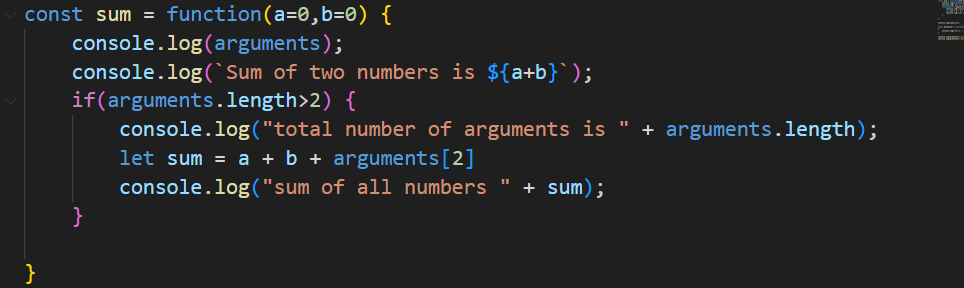
* Function with definitions



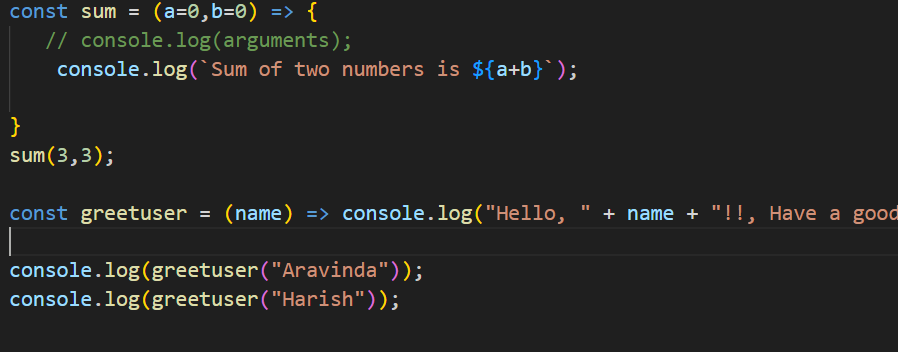
* Function with variable number of arguments



* Function with Expressions OR Anonymous functions



* Arrow Functions



Strings

ss

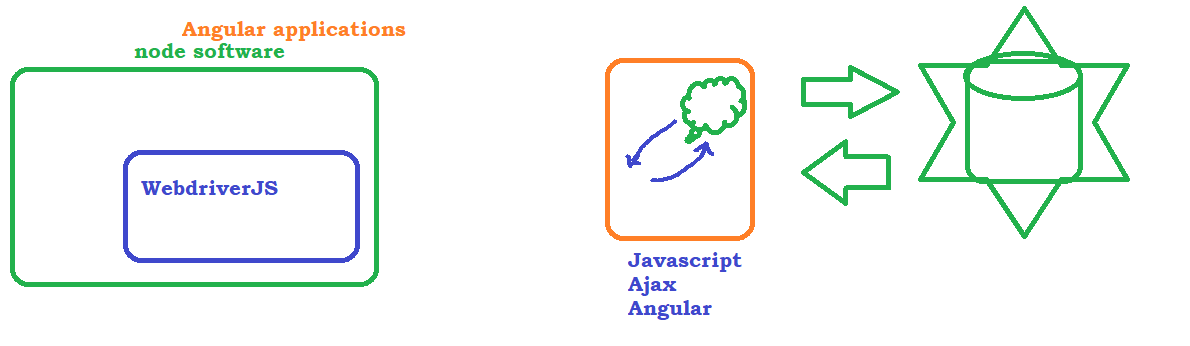
OOPS

Maps

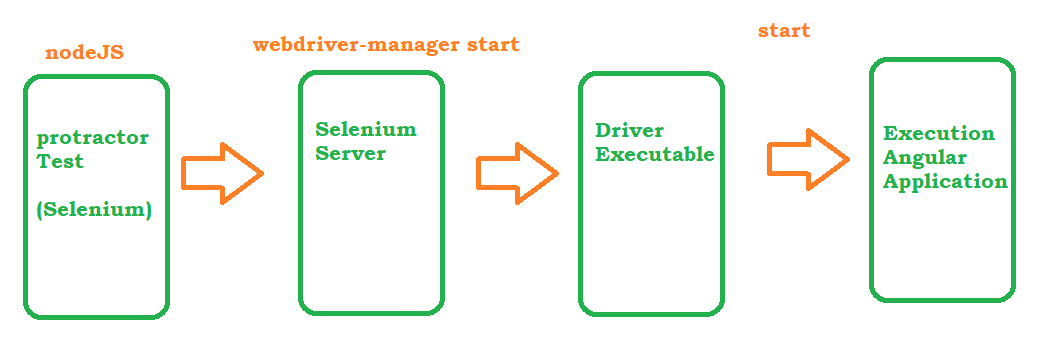
Set

Jasmine

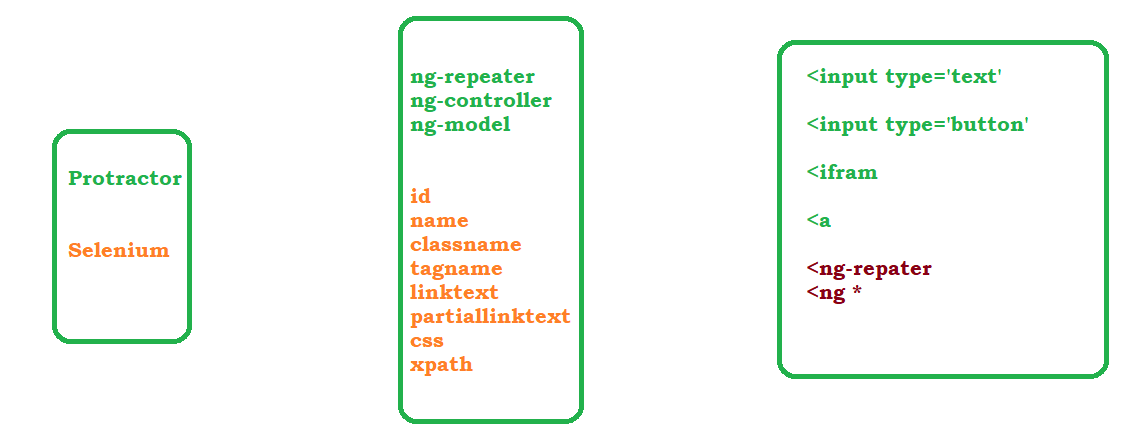
Protractor



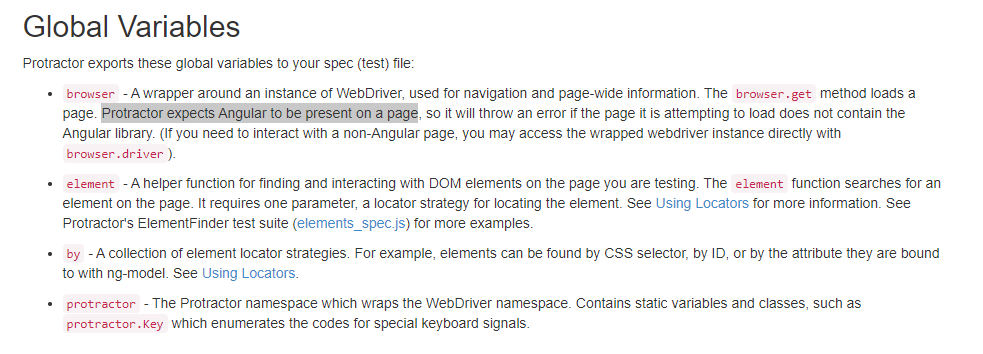
# How Protractor works



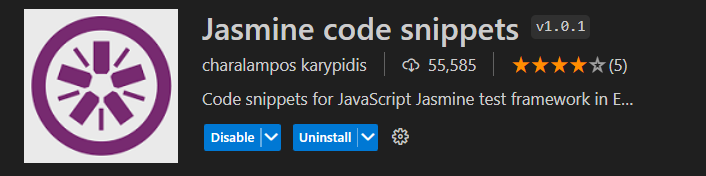
# High level diagram

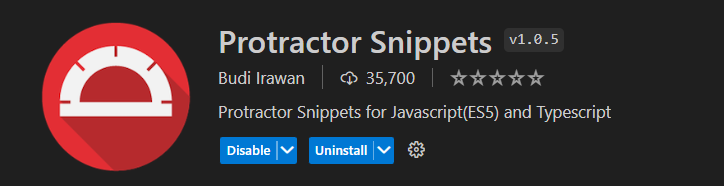


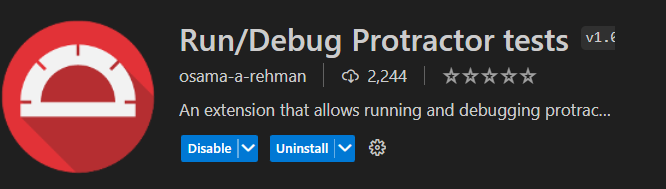
# [Global Variables](https://www.protractortest.org/#/api-overview)



# Plugins







Asynchronous nature of Java Script

Control will go to next step / next line for execution even though current step is waiting for any resource or even though current step is not completed

We can achieve synchronous nature in Java script by resolving promises.

**To resolve the promises** we have to use call back function

Promises => is nothing but the state of the step or the result of the step

State or Result :

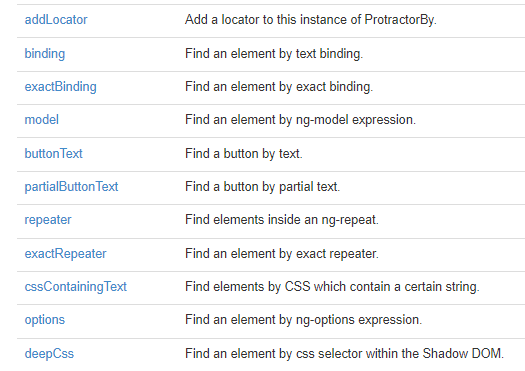
* Pending : still executing
* Resolved : pass
* Rejected : fail

**Almost all the protractor functions promises are resolved by default , i.e, control will go to next step only when the current step is resolved or rejected.**

**But there are some functions like, getText(), getAttribute(), Sleep() – promises are not handled, in these functions you have to handle the promise**

# Identifying the Elements in Protractor :

Refere : <https://www.protractortest.org/#/api?view=ProtractorBy>

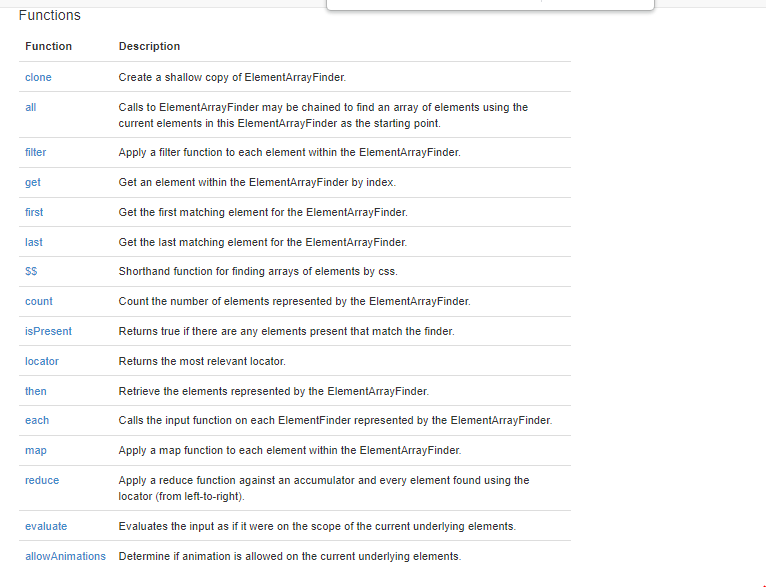


# beforeEach

beforeEach is a block of code which will be executed before executing a test. if we have any common code which needs to be executed before executing the test then we can go with beforeEach block

# Playing with more than one element

# Working with Chain locators



# Validations in Protractor

expect()....

Capturing Screen shots in protractor

* Go to npm repository : <https://www.npmjs.com>
* Search for : protractor-jasmine2-screenshot-reporter
  + <https://www.npmjs.com/package/protractor-jasmine2-screenshot-reporter>
* Install the plugin using npm i protractor-jasmine2-screenshot-reporter
* Verify the package is installed under node\_modules
* Update conf.js as per the instructions
* Execute the test
* Verify the html report and screenshot

Allure Reports

* Go to npm repository : <https://www.npmjs.com>
* Search for : protractor- jasmine-allure-reporter
  + <https://www.npmjs.com/package/jasmine-allure-reporter>
* Install the package using npm i jasmine-allure-reporter
* Update the conf.js as per the instructions
* Execute the tests and make sure xml results are captured
* To generate HTML output from allure results
  + install allure command line tool : npm i allure-commandline
  + exeute allure serve <<location of xml file>>

HTML Reports using HTML Reporter

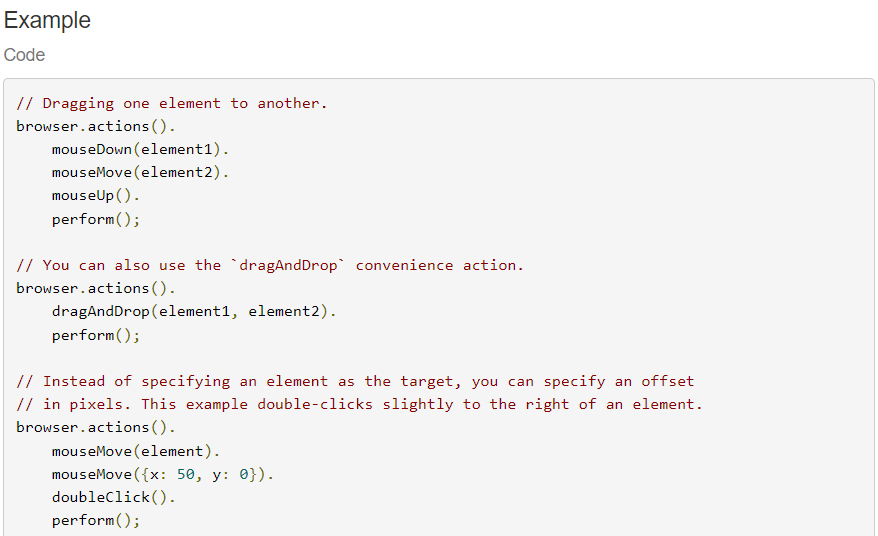
* Go to npm repository : <https://www.npmjs.com>
* Search for : protractor-html-reporter-2
  + <https://www.npmjs.com/package/protractor-html-reporter-2>
* Install the package using npm i protractor-html-reporter-2
* Update the conf.js as per the instructions
* Execute the tests and make sure xml results are captured

Explicit waits – only for non angular apps

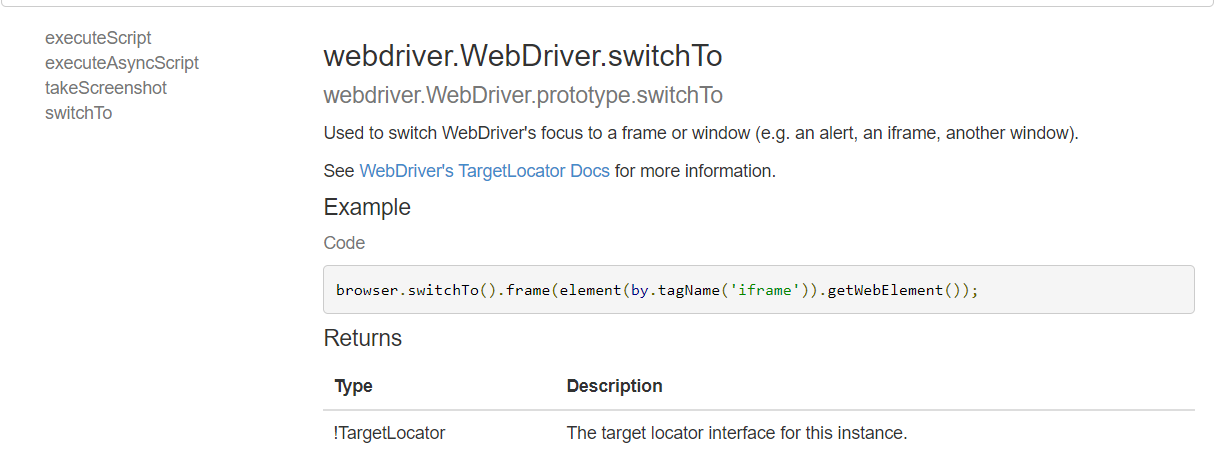
<https://www.protractortest.org/#/api?view=ProtractorExpectedConditions>



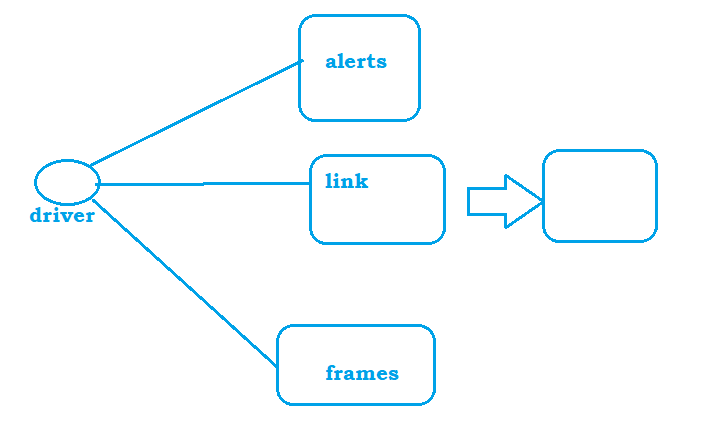
Actions in Protractor



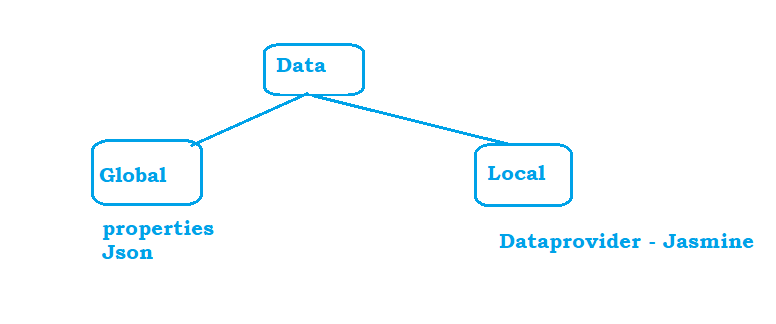
Handling Frames



SwitchTo in Protractor



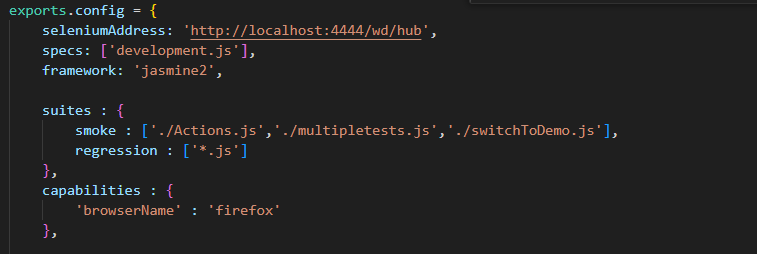
Data driven testing



Using jasmine data-provider

1. download and install jasmine-data-provider from the npm library
2. https://www.npmjs.com/package/jasmine-data-provider
3. npm i jasmine-data-provider
4. Follow the instructions to use the data provider

Batch Execution



protractor .\conf.js --suite=regression