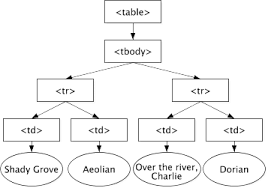
|  |  |
| --- | --- |
| Java | JavaScript |
| Ham | HamSter |
| Designed for scalability | Designed for flexibility |
| Strong and static | Loose and Dynamic |
| OOP | Multi-paradigm (functional) |
| Compiled | Just In time (Compiles as it executes) |
| Access Modifiers | No access modifiers |
| Multi-Threaded | Single-Threaded |
| Backend | Mostly frontend |
| General programming for making applications | Scripting language (designed to tell the browser what to do) |
| Generics | No generics |
|  |  |
|  |  |
|  |  |

* JavaScript features
  + Primitives
    - String
    - Boolean
    - Number
    - Undefined
    - Null
    - Symbol (es6)
  + Type coercion
    - Process by which JS will change the type of an object/primitive to perform an operation
      * String – Boolean – number
  + Truthy Falsy
    - The inherit Boolean value of any object/primitive
    - Falsy Values (imply some kind of non existence)
      * Null
      * Undefined
      * 0
      * “”
      * False
      * NaN
  + Scopes in JS
    - Global scope
      * Not using a scoping modifier x=100;
      * Attaches a variable to the global object
      * Avoid using if possible
    - Function scope
      * Var keyword
      * You should not ever use var
    - Block scope
      * Let or const keyword
        + Let means the variable can change
        + Const is like final in Java
        + Let variables are not hoisted
  + Hoisting “feature”
    - Perform a function executes in JS it will allocate memory for all variables declared in the function.
    - You can ‘use’ a variable before the code creates it because the value has been defaulted to undefined.
  + Objects
    - Traditional JS does not have classes
    - Objects can be created via the literal syntax
      * Let adam = {propery:value,…..}
    - Objects are just colletions of key value pairs
    - You can edit the properties of an object at any time
      * Add or remove properties
    - Method vs function semantics
      * Function is a standalone piece of code
      * Method is a function attached to an object
  + Functions
    - Functions are objects
    - Functions are special objects that have code attached to them that can be invoked via ()
    - Functions are first class citizens (technical term for saying functions are objects)
    - Types of functions
      * Callback function
        + A function passed into another function as a parameter
      * Higher order function
        + A function that accepts a callback function
      * Anonymous functions
        + These are functions that are never given a name or stored in a variable
      * Two ways to create functions
        + Function keyword
        + Arrow notation
  + Arrays
    - A linear collection of objects in JS
    - Arrays are indexed and maintain the order of insertion
    - Arrays can hold anything
      * Methods attached to arrays
        + Push(element) adds to an array
        + Pop() removes from an array
        + Filter(callback) return elements that meet a certain condition
        + Map (callback) return a new array where each element has been modified in some way
        + forEach(callback) call the callback function on each element in the array
  + JavaScript Object vs JavaScript Object Notation (JSON)
    - JSO is a programming object
      * Easy to work with and get the properties
    - JSON is a formatted string
      * The way to pass information around on the internet
      * Functions are not put into JSON
    - JSON object that is in web browsers
      * JSON.parse() – turns a JSON into a JSO
      * JSON.stringify() – turns a JSO into a JSON
  + AJAX (Asynchronous JavaScript And XML)
    - In the early days of the internet whenever you made an http request you always had to reload the entire page.
      * This is not a great user experience
      * Less efficient
    - AJAX is what allows JS to tell the browser’s http messenger to make an http request and not have to reload the page
    - Allows us to make http calls in the background
    - You can tell JS what to do when you get the information back instead of reloading the page with the new information
  + Making an HTTP call
    - Use fetch
      * Fetch (url,config)
        + You get back a promise which will eventually hold a HttpResponse
        + Promise is an object in JS that will eventually have a value
    - Async await
      * Syntactic sugar for doing .then()
      * Async in front of a function signature
        + This enables you to use the await keyword within it
        + Conveniently documents the function as executing code that does not immediately give a response
      * Await
        + Tells JS to wait until the promise object has a value before continuing execution
    - .then()
      * The promise object has a method called .then()
        + It accepts a callback function to execute once the promise is resolved
  + HTTP more advanced
    - HTTP responses come back in two parts
      * First you get back the header
      * You get the body and download it

DOM Manipulation

* DOM (Document Object Model)
* A tree representation of elements in the html page
* 
* JS was invented so that we could programmatically update and edit the DOM
* Ways to add JS to your html page
  + Script tag
    - Write your JS within the script tag
  + In line JS
    - Write JS directly into your html elements
    - Generally not a great idea
  + External File
    - Have your script tag reference an external .js file
* Getting elements in JS
  + Document.getElementById(“elementid”)
    - By tag
    - By name
* Responding to events in JS
  + <element onclick=”function()”>something </element>
  + Add an event listener
    - Element.addEventListener(“event”, callback, capturing?)
* Template literal
  + Use the JS `` backticks to interpolate our string values
    - `hello I am ${person.name}`
  + I use this a lot for swapping inner html to create dynamic webpages
* Alternate to swapping innerHtml to edit the DOM
  + You can elements and there are functions for appending and removing elements/nodes

Testing our webpages

* Usually testing a webpage is an end to end test
  + Your entire backend and front end has to work for this test to succeed.
  + True for anything you test on the webpage that uses your API
* Selenium
  + Is a browser automation software
  + Selenium is NOT A TESTING SOFTWARE
    - Often used for testing
  + The main core of selenium is the webdriver
    - You need a webdriver for each different browser
  + Allows us to program actions that a human user could perform
    - Type this into a field
    - Click on this button
    - Etc….
* Jasmine
  + JUnit for JavaScript
  + Jasmine test syntax/structure
    - Describe (“suite description”, ()=>{})
    - It (‘should do something’, ()=>{}) each it is a spec/test
    - Expect(actual value)
    - ToBe (expected value)
* Protractor
  + Is a library of tools used to test Angular webpages
    - It can also be used on other web pages
  + Protractor is a combination of
    - Selenium
    - Jasmine
    - Angular specific libraries
* Setting up protractor
  + Start your selenium server
  + Create a node project
  + Add the protractor dependency
  + Write a main file that has configuration details of where to find the selenium server and where to find your specs
  + Write your specs
* Page object Model
  + A JS class that has fields which are the interactable elements on the webpage
  + Makes writing out tests easier

Node.js

* Node.js
  + Node is a runtime environment of JavaScript on the server side
  + This allows you to create backend applications/APIs written in JS
* Structure of a node application
  + Every node application has a main.js file (technically can be named anything)
    - It is the entry point of the application (like a main method in Java)
  + Package.json
    - This is essentially node’s version of a pom.xml
    - It keeps track of dependencies of your application
* It also has meta information like the name and author of the application