**JavaScript Engine**

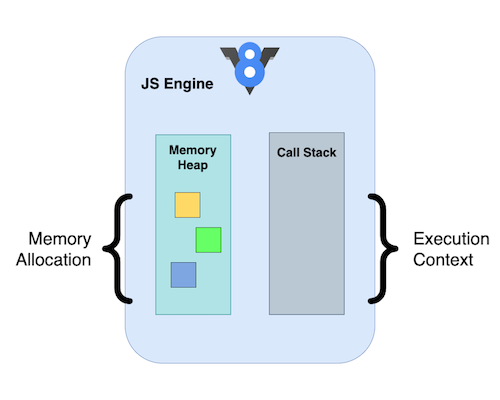
The JavaScript engine is the part where execution of the code happen.

some of the popular JavaScript engines:

* Google V8 JS engine used in the Google Chrome browser and in Node.js.
* Mozilla SpiderMonkey used in the Mozilla Firefox browser.
* Apple Nitro and JavaScript Core used in the Safari browser.
* Microsoft Chakra and CharkraCore, used in the Microsoft Edge browser.

The JavaScript engine converts your JavaScript code into binary machine language that understood by the hardware in computer and can be executed.

The two main components of an engine are the heap memory (memory allocation) and the call stack (execution context).



Java script memories source: <https://javascript.plainenglish.io/how-does-javascript-actually-work-under-the-hood-567f9f17b855>

**Heap Memory**

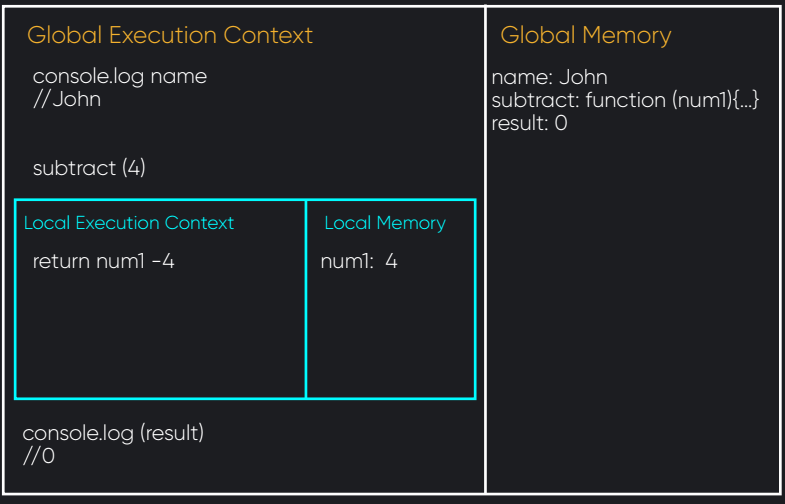
where the memory allocation for the variables and function declarations happens.

**Call Stack**

which is responsible for executing the code. When the code is parsed from top to bottom, it gets pushed onto the call stack. As a stack follows the **LIFO (last in first out)** order, any code pushed onto the stack will execute and then will be popped off. This process will continue till all blocks of the code are executed.

Heap memory

Stack execution



**Callback Queue (Task Queue)**

The callbacks and results from the different Web APIs will get pushed onto the callback or task queue. All the callbacks/results will get queued and later be moved onto the call stack by the event loop.

The callback queue for the V8 engine has a Microtask queue and a Macrotask queue.

The high priority callbacks like, promises, asynchronous function calls are classified as microtasks. Other callbacks are less riority like setTimeout, setInterval, DOM Events, etc., that are considered macro tasks.

The results and callbacks are moved onto their respective queues once they get processed by the WebAPIs.

**So what is node? What can you do with it?**

NodeJS is a javascript runtime built in Chrome’s V8 engine. Node allows you to execute Javascript code outside of the browser, in a computing environment (such as a server or local development environment) rather then a browser environment.

**There are 5 main things people use node for.**

* Package management through [npm](https://www.npmjs.com/), [bower](https://bower.io/), [jspm](http://jspm.io/), etc.
* Development tooling e.g with comjuction of visual studio code or other tools.
* Creation of back-end web applications.
* Command line tools like [rimraf](https://github.com/isaacs/rimraf).
* [Desktop applications](http://electron.atom.io/).
* There are quite other special use cases, like developing AI and [neural networks](http://synaptic.juancazala.com/#/) apps, or [chat bots](https://docs.botframework.com/en-us/node/builder/overview/), or really anything .

**What is NPM?**

npm stands for Node Package Manager. It's a library and registry for JavaScript software packages. npm also has command-line tools to help into install the different packages and manage their dependencies. npm is free and relied on by over 11 million developers worldwide

**Installing npm**

npm is installed with Node.js

we have to install Node.js from web site: [https://nodejs.org](https://nodejs.org/) to get npm installed on the computer.

**NPM Usage**

npm can manage packages that are local [dependencies](https://en.wikipedia.org/wiki/Coupling_(computer_programming)) of a particular project, as well as globally-installed JavaScript tools. When used as a dependency manager for a local project, npm can install, in one command, all the dependencies of a project through the package.json file. Json file is important in this process and will be described bellow.

**What is the file `package.json`?**

All npm packages contain a file, usually in the project root, called package.json - this file holds various metadata relevant to the project. This file is used to give information to npm that allows it to identify the project as well as handle the project's dependencies. It can also contain other metadata such as a project description, the version of the project in a particular distribution, license information, even configuration data - all of which can be vital to both npm and to the end users of the package. The package.json file is normally located at the root directory of a Node.js project.

e.g. of package.json

{

"name" : "underscore",

"description" : "JavaScript's functional programming helper library.",

"homepage" : "http://documentcloud.github.com/underscore/",

"keywords" : ["util", "functional", "server", "client", "browser"],

"author" : "Jeremy Ashkenas <jeremy@documentcloud.org>",

"contributors" : [],

"dependencies" : [],

"repository" : {"type": "git", "url": "git://github.com/documentcloud/underscore.git"},

"main" : "underscore.js",

"version" : "1.1.6"

}

 there are fields for the description and keywords of your projects. This allows people who find your project understand what it is in just a few words. The author, contributors, homepage and repository fields can all be used to credit the people who contributed to the project, show how to contact the author/maintainer, and give links for additional references.

The file listed in the main field is the main entry point for the library; when someone runs require(<library name>), require resolves this call to require(<package.json:main>).

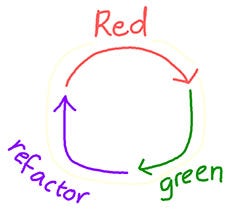
Finally, the dependencies field is used to list all the dependencies of your project that are available on npm. When someone installs your project through npm, all the dependencies listed will be installed as well. Additionally, if someone runs npm install in the root directory of your project, it will install all the dependencies to ./node\_modules.

It is also possible to add a devDependencies field to your package.json - these are dependencies not required for normal operation, but required/recommended if you want to patch or modify the project.

**TDD Test Driven Development**

The test driven development is by tests*'drive' the* development process.

How it is work? Start by writing a failing test, after that improve the code to make the test pass, then refactor. The repeated cycle is called red, green, refactor



Red, green, refractor cycle . source : https://northcoders.com/company/blog/the-benefits-of-test-driven-development-tdd

RED: The red phase is the starting point of the cycle, where you define expectations for the piece of code being tested, and let the test fail.

GREEN: Implement the necessary logic to make your test pass

REFACTOR: During the refactor phase, consider how you could optimise your code, without adding any additional functionality.

**Benefits of the TDD Test Driven Development**

* Using TDD you can dictate what you expect the result of the code from the ground up. thinking about what you expect from scenario to scenario is forced from the begning, so it is likely to cover all the bases.
* It helps you to develop the logic in your coding By early tests with the simplest functionality first, then use first tests to guide your logic as you build up functionality. By this a problem can be break down into smaller and more manageable parts.
* Exherts very high test-coverage, test-coverage refers to the proportion of your code tested. A high test coverage means your code works can be trusted because as large set of tests already done.
* High quality of the code ,as your code is written to pass the tests in place, and refactoring at the end of each test; you ensure your code is clean and optimised without any extra pieces of .
* Bugs are early Prevented ,as you are adding to the functionality of the code while you go along, each stage of the code is tested working meanwhile your progress.
* Adding functionality to the code is easy, with a high test coverage, you topupa highluy tested piece of code. So you can rely on the code that y already have, additional tests can be conducted to add functionality bit by bit.
* The code is understood easily, the benefit of testing in general (and using descriptive test statements!), is the increased readability of the code. Developers, testers and non-technical colleagues working on or around the codebase can identify exactly what the piece of code does.

**a higher order function.**

 a function which takes another function as an argument or returns a function is known as a higher order function.

examples of .map() and .filter(). Both of them take a function as an argument. They're both higher order functions.

**higher-order functions useful?**

Higher-order functions are :

* making your code brief with les declarations of variables and functions,
* the aove feature also save memory that used by variables and functions.
* flexible
* and maintainable in the same time it is still relevant and useful.

e.g. of Higher-order functions

const num = [1, 2, 3, 4, 5];

num.forEach((num) => console.log(num));

When applied a Higher-order functions forEach() to the array called "num" At the beginning of forEach(), there is an anonymous function passes as arument, which process each element of the array - one at a time.

The Higher-Order Function forEach() applies a function to each element of an array.