What is a run time environment

Think of a run time environment as anything lying between your program and the operating system it runs on/

Its main job is to provide services for your executing program. These services can vary greatly in scope from providing runtime libraries and memory management to managing the execution of your code and abstracting interactions with your operating system.

Static Libraries and Dynamic Libraries

The complexity and functionality of the runtime environment depend on the programming language. In some programming languages the runtime environment is minimal and primarily focused on low-level tasks like memory management and input/output. In others it might include advanced features garbage collection, exception handling and cross-platform compatibility

---

JavaScript Runtime

An environment which provides all the necessary components to use and run javascript. There are many javascript runtime environments

- Different Browser Runtimes
- Node.Js Runtime
- Deno Runtime
- Bun Runtime

Let's talk about a browser's runtime

- JavaScript Engine
- Web APIs
- Callback queue
- Micro-task queue

Engine

Think of it as the javascript compiler! So javascript is compiled?
Let's talk about compiler pipelines

it is a program which executes javascript code

Every browser has its own JavaScript engine

- chrome : V8 engine
- Firefox : spider monkey
- explorer :

Web APIs

Talk about Sandbox architecture

- WEB API provide extra functionality to the engine that is not part of the language itself.
- DOM
- TIMER
- FETCH
- console.log

Callback queue (later)

Micro-task queue (later)

JavaScript Execution

When the engine is executing javascript code. It on the side has what is called an execution context of the running code.

An execution context has
- variable environment : variables - function declarations
- scope accessible to the code
- this keyword
- we can add (global object) in a special context

so you can say that an execution context is simply

- **abstract context that contains information about the environment of the currently executing code.**
- **it is an abstract concept in javascript that defines the environment in which javascript code is executed.**
- **it is a device used to track the run time evaluation of code.

Types of execution context

- Global execution context
  - created for any piece of code that isn't a function
- Function execution context
  - when a function is invoked
- Eval execution context

This whole execution context stuff has two phases

- Creation of the execution context
- Actually executing the code

```
var x = 10;

function foo(a){

    var b = 20;

    function bar(){
        var c = 30;
        console.log(a + b + c):
    }

    bar();
}

foo(40)
```

Let's spend some time (a lot of time) analyzing this code.

This

I mentioned earlier in the execution context something called the this keyword.

What is this? (funny right)

Definition It is a special variable that exist for every execution context. It most of the times refers to the caller of the function (Don't make it a rule)

- It is not a static value because I said it related to the execution context

Examine

```
const obj = {
        name: 'John',
        age: 30,
        print: function() {
                console.log(this.name, this.age);
        }
};

obj.print(); // John 30

const obj2 = {
        name: 'Jane',
        age: 25,
};

obj2.f = obj.print; // since obj.print is a function, it can be assigned to
obj2.f and it is by reference

obj2.f(); // ?
```

2.

```
function seeThis(){
        console.log(this);
```

```
}

seeThis();
```

```
'use strict'

function seeThis(){
        console.log(this);
}

seeThis();
```

3.

```
'use strict';


function callMe(cb){
        cb();
}


const obj = {
        name: 'John',
        age: 25,
        greet: function(){
                console.log(this.name + ' says Hello');
        }
}


callMe(obj.greet); // what is the result?
```

4. Is there is a solution to this.

```
callMe(obj.greet.bind(obj))
```

```
const obj2 = {
        name: "Mohamed",
        age: 30
}

obj.greet.bind(obj2)()

obj.greet.call(obj2, arg1, arg2, ..)

obj.greet.apply(obj2, [arg1, arg2, ...])

// let's try them
```

Asynchronous JavaScript

Say you have a GUI program with a lot of buttons
most of button do simple computations but one of the buttons load a 1 gb file into memory

if you pressed that button what will happen
the GUI will freeze until your operation is finished

that is not acceptable right the operation might take a minute and that is a terrible user experience what might you do in this case

you can make another process/thread do the work of the loading file into memory and the main thread on yours remain looking after the GUI

so now what you have two processes

one process is looking after the GUI
one process is invoked when i clicked the button and it is loading the big file into memory

that is one solution. There is another solution is to use non-blocking IO operations

forget about all of this now and let's talk

what happens during loading data from disk to memory. Do you think the processor will be too busy handling this request let's look close

let's start what you are thinking

Here is your processor here is the main data bus now the processor want to fetch data from the disk so it will instruct the disk directly the disk will does its thing and send data to the processor then the processor will send it to the memory and back and forth until the whole file is loaded. so if you have a single core all your life will be stopped for all this time

so it's not efficient what about introducing small specialized computer that manages this operation and then informs the processor when its finished that is the DMA

it's job is allowing peripherals to communicate with memory without involving the processor.

how it works

the processor instructs the DMA controller to move data from disk to memory the dma will handle the data transfer independently after the transfer is complete it generate an interrupt signal

so when I call load function the processor isn't doing any work so why my program blocks that is because this is called blocking io

you make a system call (looks like a function call but its kinda like a call to the operating system to do something)

read() you block the operating system goes until the he has the file then returns to you

we will then go for another thing which is the non-blocking io when we call read() the os will return immediately and we call poll the resource to see if is finished or not later or it will notify us when one of the resources is finished

that is basically what asynchronous is under the hood

Basically,

- JavaScript is synchronous blocking single-threaded language

so we need new pieces from outside javascript itself. so where will we go?
the runtime
runtime environments provide javascript with functions and APIs that allow us to register

functions that shouldn't be executed synhronously and instead called when something happened like a file is read

now you can let your code do several things at once

```
setTimeout(() => {
        console.log("hello")
}, 1000);

setInterval(() => {
        console.log("hello");
}, 1000);
```

you do know what a callback is right. a callback is simply a function passed to another function as argument which calls it

---

More

- How function constructors work
- Hositing being totally wrong
- LLVM introduction

Resources

- understanding javascript the weird parts
- JavaScript Internals
- This key word
- Event loop visualized
- Execultion Contexts

Practise

https://github.com/nmadd/AC3.3/tree/master/lessons/javascript-advanced