**Module;**

A reusable block of code whose existence does not accidentally impact other code

**Common js modules:** an agreed upon standard for how code modules should be structured

**First-class functions:** everything you can do with other types you can do with functions. You can use functions like strings. Numbers etc.. them around set variables equal to them ut them in arrays and more.

**An expressions:** a block of code that results in a value. Functions expression are possible in js because functions are first class

-----------------------------------------------------------------------------------------------------------------------------------------

Name/value pair: a name which aps to a value

The name may be defined more than once but only can have one value in any given context. That value may be more name/value pairs.

Ex: address =’100 main st’ ---🡪 address= name 100main st 🡪 name value

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Object: a collection of name/value pairs the simplest definition when talking about js

Object literal: name/value pairs separated by commas and surrounded by curly braces. This is just a quick shorthand way to create javascript objects in code.

{

street:'main',

number:100,

Apartment:

{

floor: 3,

}

}

-----------------------------------------------------------------------------------------------------------------------------------------

Javascript aside: prototypal inheritance and function constructors.

Inheritance: one object gets access to the properties and methods of another object.

OBJ2

OBJ

Protype chain

Proto

{ }

Proto

{ }

Obj.prop1

Obj.prop2

Obj.prop3

Obj2.prop2

Function constructor: a normal function that is used to construct objects. The ‘this’ variable points a new empty object and that object is returned from the function automatically.

How do node moduels work.

Require is a function that you pass a path too

Module.export is what the require function returns

This works because your code is actually wrapped in a function that is given these things as function parameters

JSON

Javascript object notation a standard for structuring data that is inspired by javascript object literals.

Javascript engines are built to understand it.

{

street:'main',

number:100,

Apartment:

{

floor: 3,

}

}

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Web server checklist:

Better ways to organize our code into resuable pieces

Ways to deal with files

Ways to deal with dataases

The Ability to communicate over the internet

The ability to accept requests and send responses

A way to deal with work that takes a long time

Events and the event emitter:-

**Conceptual aside events:**

**Event:**  something that has happened in our app that we can respond to.

In node we actually talk about two different kinds of events

Coustom Events

System Events

C++ core --------------------------------------------------🡪> JS library

Libuv Event emmiter

Event listener:

The code that responds to an event. In javascripts case the listener will be a function

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Node js features:

* Asynchronous and event driven
* Very fast
* Single threaded but highly scalable
* No buffering

Uses of node js

* i/o bound applications
* data streaming applications
* data intensive real time applications
* JSON API is based applications
* SPA

**Single Thread flow:**

NODE JS thread pool

T2

T1

Event Queue

Request 1

Blocking-IO

Database

R-n

T-n

Event LOOP

Request 3

Request 2

Single thread flow: Request1--🡪 event queue🡪 event loop🡪 request1

Request2--🡪 event queue🡪 event loop🡪 request2

Event queue generate response

When blocking I/o dependency will come ..

---T2 will work on blocking I/o dependencies. T1 work on other thread pool in node js

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Know something:

Check nodejs version: node –version

Install node package: npm install <package name>

Uninstall npm package: npm uninstall<package name>

Search npm package : npm search<package name>

Update npm package: npm update <package name>

List of installed packages: npm ls

List of installed packaged globally: npm ls -g

------------------------------------------------------------------------------------------------------------------------------------------

1.callback functions

There are two types of blockings.. 1. Blocking 2. Non blocking

Node js mainly work on non-blocking concepts it means

**Npm install fs**

Non-blocking examples:

**var** fs = require(**'fs'**);  
  
fs.readFile(**'sample.txt'**,**function** (err, data) {  
 **if**(err) **return console**.error(err);  
 **console**.log(data.toString());  
});  
**console**.log(**'File reading started'**);

2.Directories

* **Create directory**

**var** fs = require(**'fs'**);  
  
**console**.log(**'create directory..'**);  
fs.mkdir(**'C:/Users/KRISHNA/Desktop/node/hacker\_dir'**,**function** (err) {  
 **if**(err){  
 **return console**.log(err);  
 }  
 **console**.log(**'directory created successfully'**);  
});

* Read directory

**var** fs = require(**'fs'**);  
  
**console**.log(**'read directory'**);  
fs.readdir(**'C:/Users/KRISHNA/Desktop/node'**,**function** (err,files) {  
 **if**(err){  
 **return console**.log(err);  
 }  
 files.forEach(**function** (file) {  
 **console**.log(file);  
 });  
});

* Remove directory

*var fs = require('fs');  
  
console.log('removing directory');  
  
fs.rmdir('C:/Users/KRISHNA/Desktop/node/hacker\_dir',function (err) {  
 if(err){  
 return console.log(err);  
 }  
 console.log('deleted successfully');  
});*

3.file system

* Open file & read file

fs.open(**'sample.txt'**, **'r+'**, **function** (err,fd) {

**if**(err){  
 **return console**.log(err);  
 }  
 **console**.log(**'FIle openend.....'**);  
 **console**.log(**'start reading file..'**);  
 fs.**read**(fd, buf, 0, buf.length, 0, **function** (err, bytes) {

**if**(err){  
 **console**.log(err);  
 }  
 **console**.log(**"bytes read:"**+ bytes);  
 *//print only read bytes to avoid junk* **if** (bytes>0){  
 **console**.log(buf.slice(0, bytes).toString());}  
 });  
});

* Reading file
* Writing file

**console**.log(**'writing file'**);  
fs.writeFile(**'sample.txt'**,**'Adding new content to the file hacker'**, **function** (err) {  
 **if**(err){  
 **return console**.log(err);  
 }  
 **console**.log(**"data written successfully"**);  
});

* Close file

fs.close(fd, **function** (err) {  
 **if**(err){  
 **console**.log(err);  
 }  
 **console**.log(**"file closed successfully.."**);  
});

* Delete file

console.log(**'Deleting existing file..'**);  
fs.unlink(**'sample.txt'**, **function** (err) {  
 **if**(err){  
 **return** console.log(err);  
 }  
 console.log(**"file deleted successfully"**);  
});

* Event
* Event emitter
  + Event listener
  + Emitter
* Difference between callback functions and events