

# Adv Web Dev Architecture

## Lecture 4

Amir Amintabar, PhD

# Outline

- Review
- 1- Server and server scripting
- 2- Node js
- 3- JS is asynchronous, single threaded, restaurant analogy
- 4- Node.js installation
  - Node.js debugging in VSC ( Visual Studio Code)
  - Node.js examples
    - Hello World!
- 5- Module (put your functions in module, just like including js external files), examples
- 6- using built in modules, examples
  - http
  - url
  - fs
- 7- Hosting Node js in shared hosting services, example

# Review

- Hoisting
- Functions with no return statement, return undefined !
- JS is asynchronous, single threaded, non-blocking!
- Difference between GET and POST
- In JS a function can return a function

- every team member should have a comprehensive understanding of the entire project, including the work done by their fellow teammates,

# Code of conduct

- Every team member should have a comprehensive understanding of the entire project, including the work done by their fellow teammates.



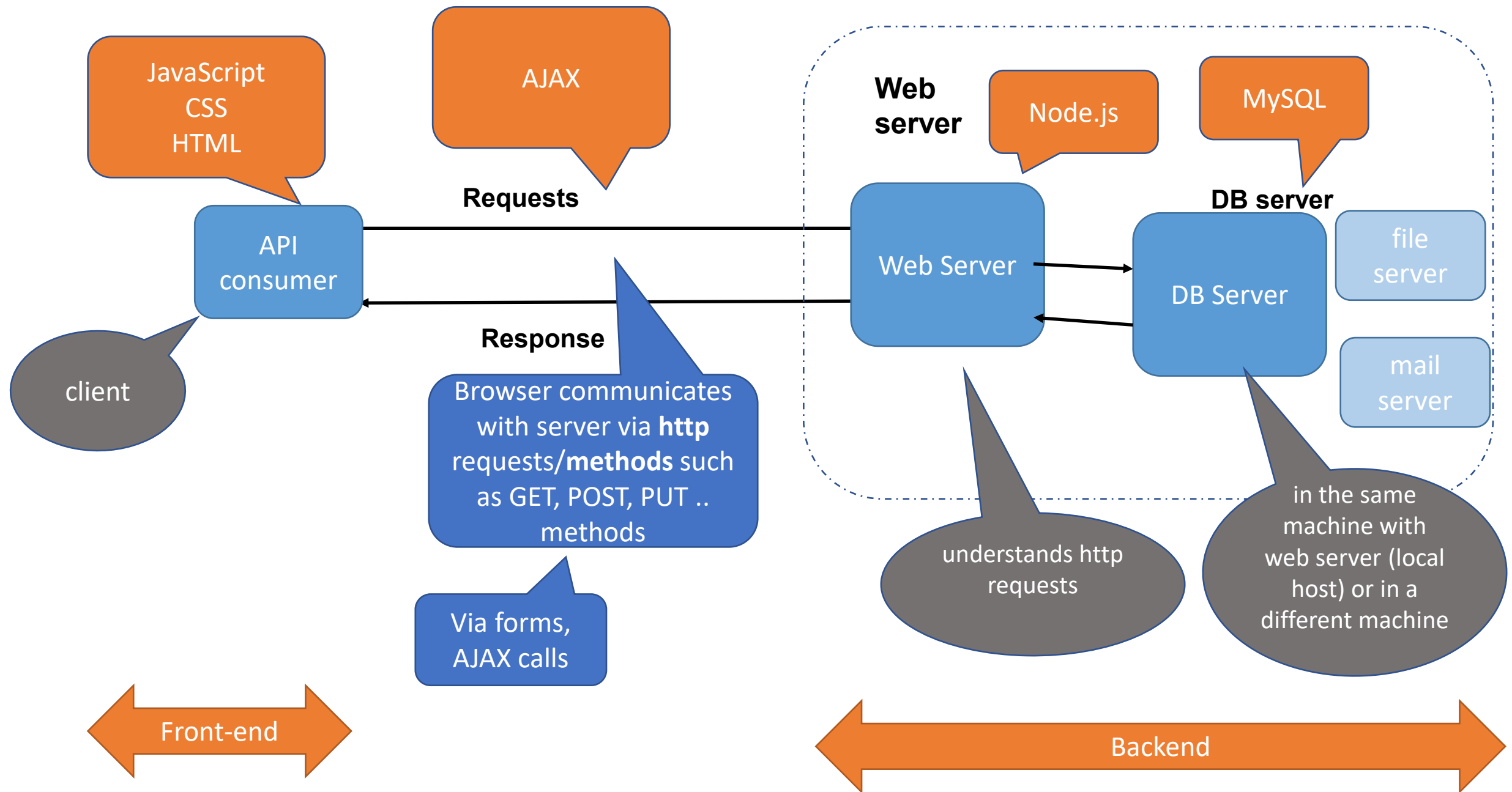
1

## Server and server scripting

# What is a server?

- A server is a computer ( or computers) running 24/7 and accessible via the internet
- When a client such as a browser or mobile app sends a requests via protocol such as http, ftp etc, the server looks for the application responsible to deal with that request, and returns the response using those protocols
- In this course, the request from clint to server are carried using http methods such as GET, POST etc.

# What is a server?





# What a server side scripting language can do?

- It can create a webserver to
- generate dynamic page content
- E.g. opening `wordpress.com/index.php` at browser
  - sends a request to the server,
  - The server looks for an app to run php files
  - If the php server is installed it runs that file and returns the result in the form of html
- Server side script can create, open, read, write, delete, and close **files** on the server
- Server side script can collect form data
- Server side script can add, delete, modify data in your **database**

In this course, we are going  
to use

2

node.js

for  
server side (backend) scripting

# What is Node.js?

- Node.js is an open source, free, server environment that uses JavaScript on the server side
- Node.js is an **open-source, cross-platform**, back-end JavaScript **runtime environment** that runs on the
- **Chrome V8** engine
- and executes JavaScript code **outside a web browser**

# Why nodeJS?

- **Node.js is non blocking!**
- Here is how PHP or ASP handles a file request:
  1. Sends the task to the computer's file system.
  2. Waits while the file system opens and reads the file.
  3. Returns the content to the client.
  4. Ready to handle the next request.
- Here is how Node.js handles a file request:
  1. Sends the task to the computer's file system.
  2. Ready to handle the next request.
  3. When the file system has opened and read the file, the server returns the content to the client.
- Node.js eliminates the waiting, and simply continues with the next request.
- Node.js runs single-threaded, non-blocking, asynchronously programming, which is very memory efficient.

JS is  
asynchronous  
single threaded  
nonblocking

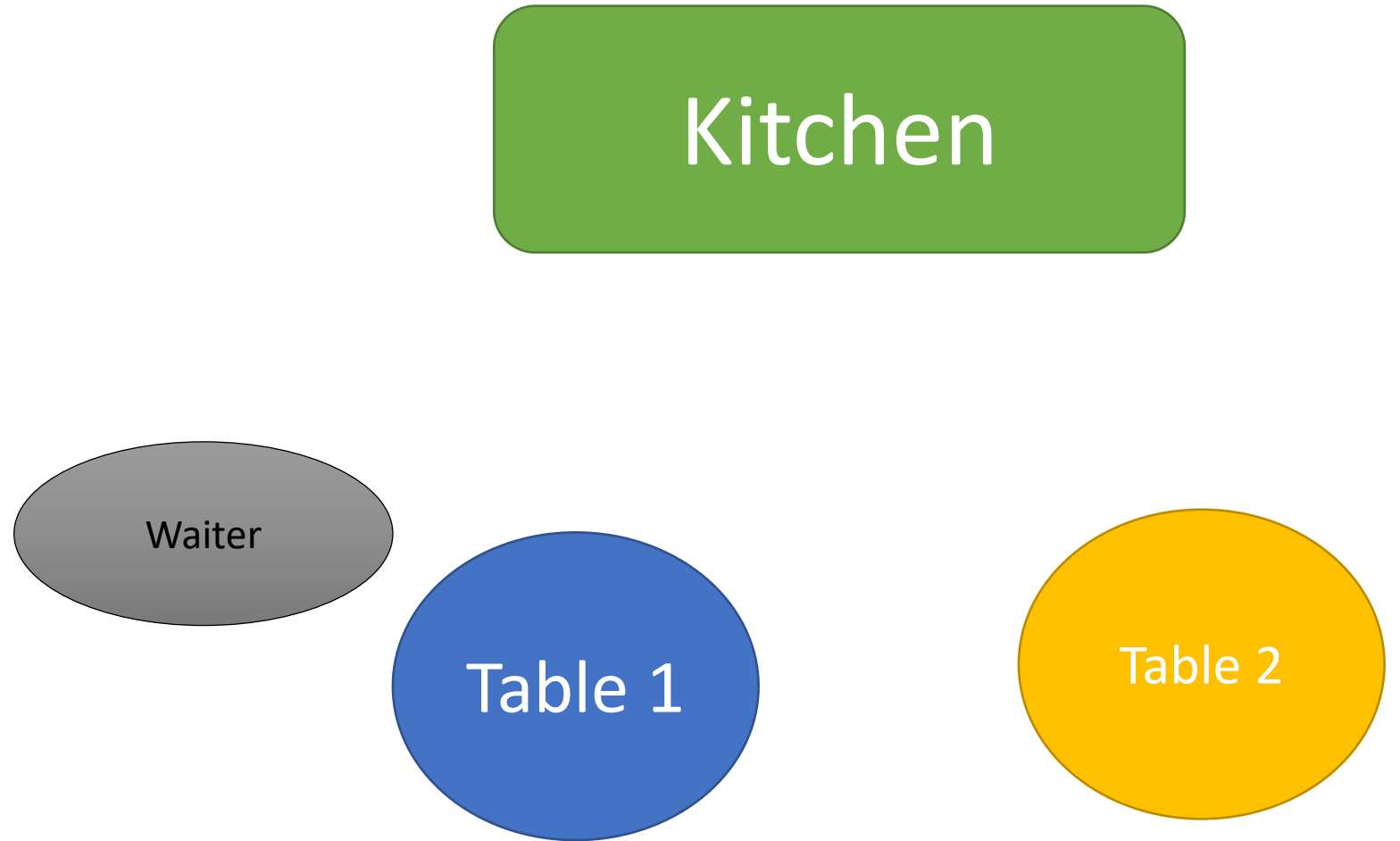
An orange oval with a thin black outline, containing the white number 3.

3

# The restaurant analogy!

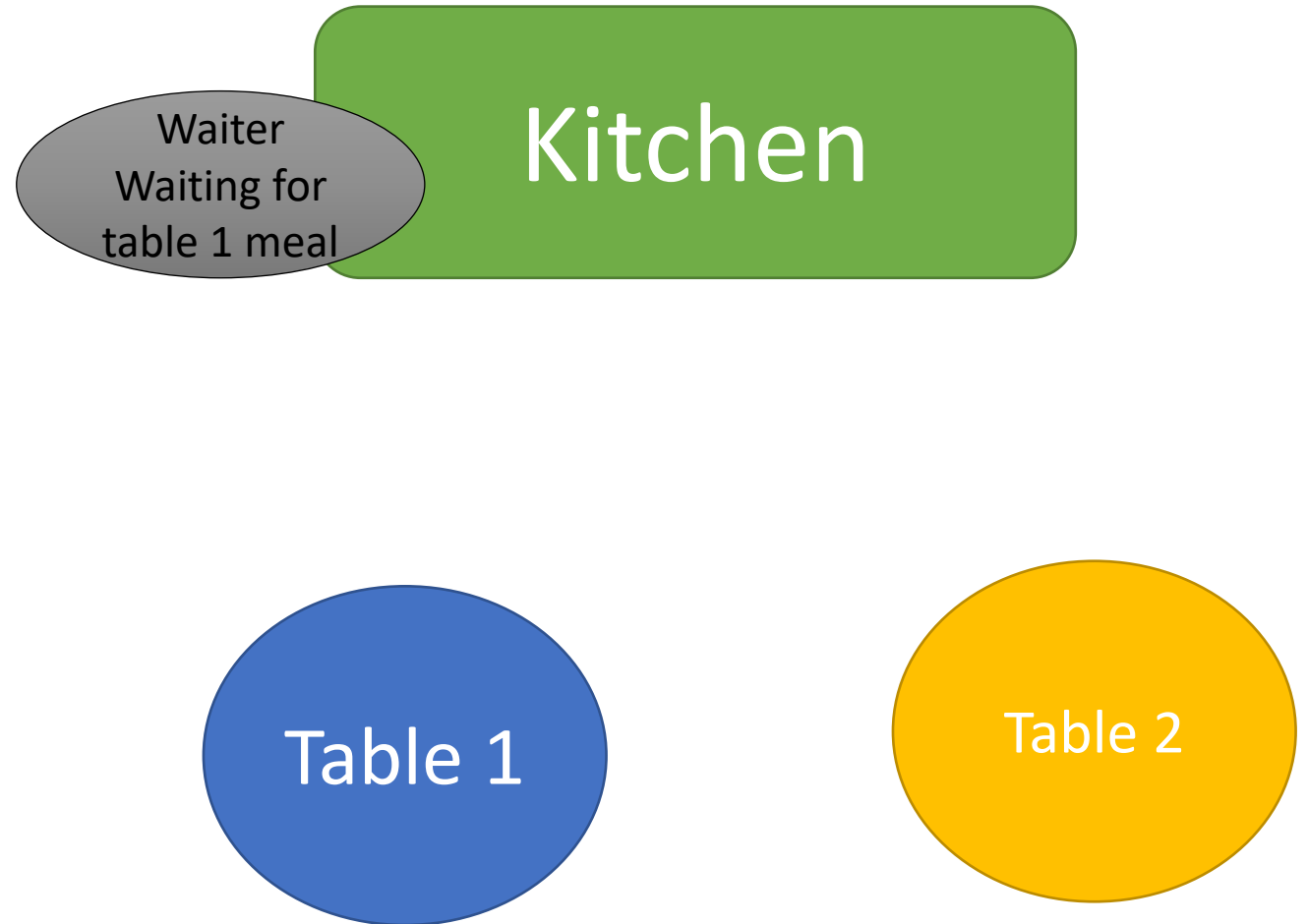
# Example 1

- Same waiter can serve multiple tables



## Example 2

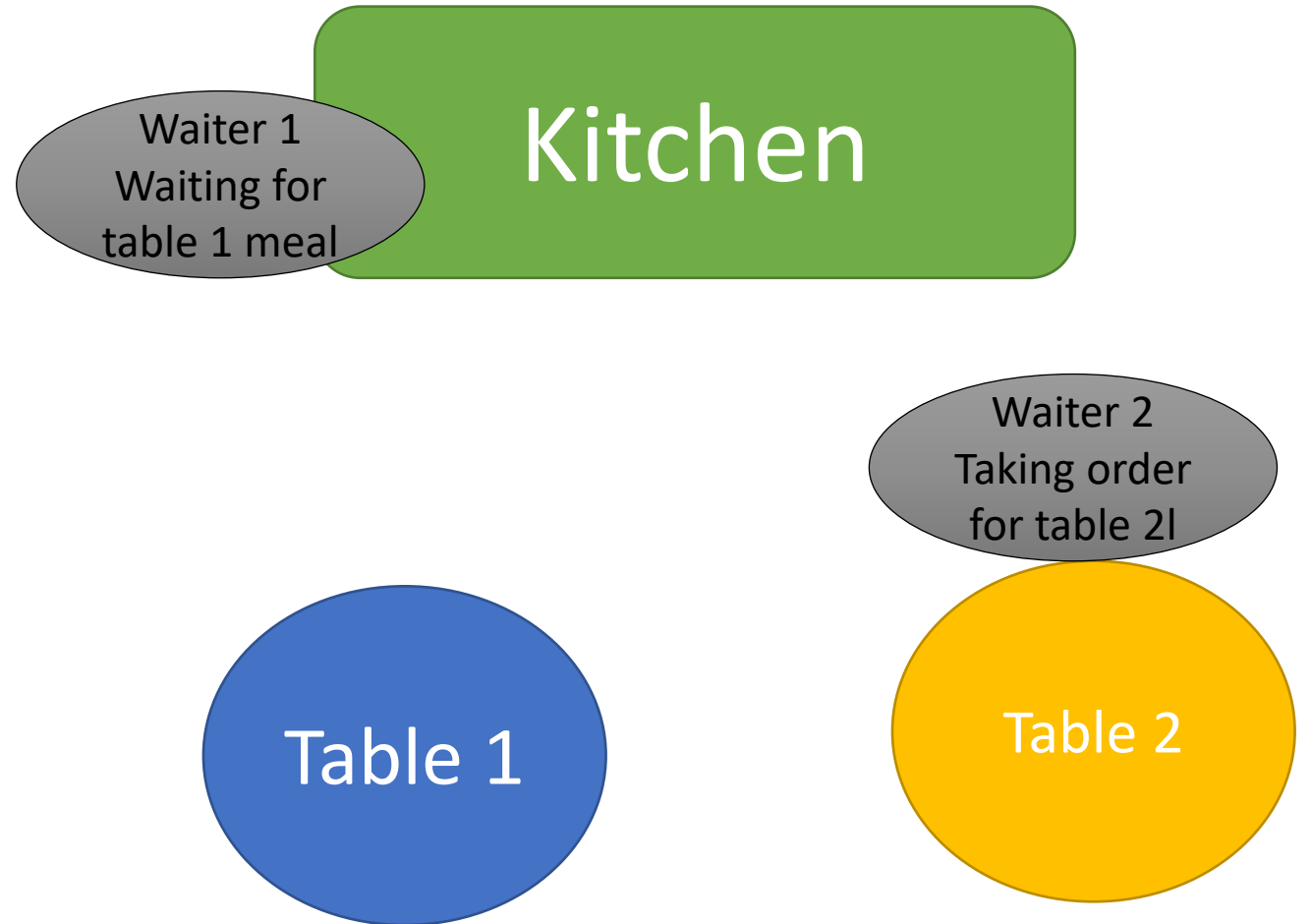
- Same waiter can serve multiple tables, but every time waits for the chef to prepare the meal then (and only then) takes another order.





# Example 3

- Multiple waiters
- Each dedicated to a single table only!

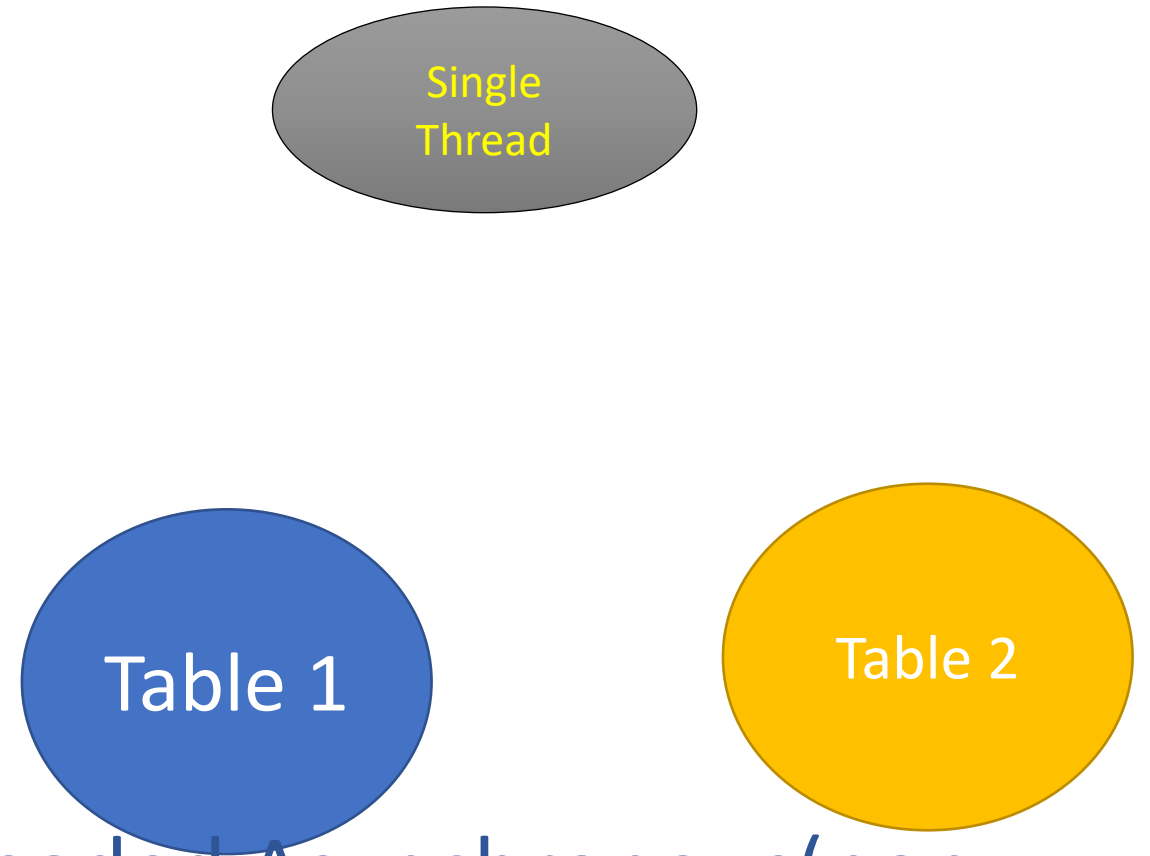


- Example 1: single threaded non-blocking ( asynchronous)
- Example 2: single threaded blocking (synchronous)
- Example 3: multi-threaded

JavaScript(nodejs) is  
Single threaded  
Asynchronous(non blocking) just like the restaurant in  
Example 1 ( one waiter serves everyone)

# Non-blocking

- Same waiter can server multiple tables (don't not get blocked by one table)

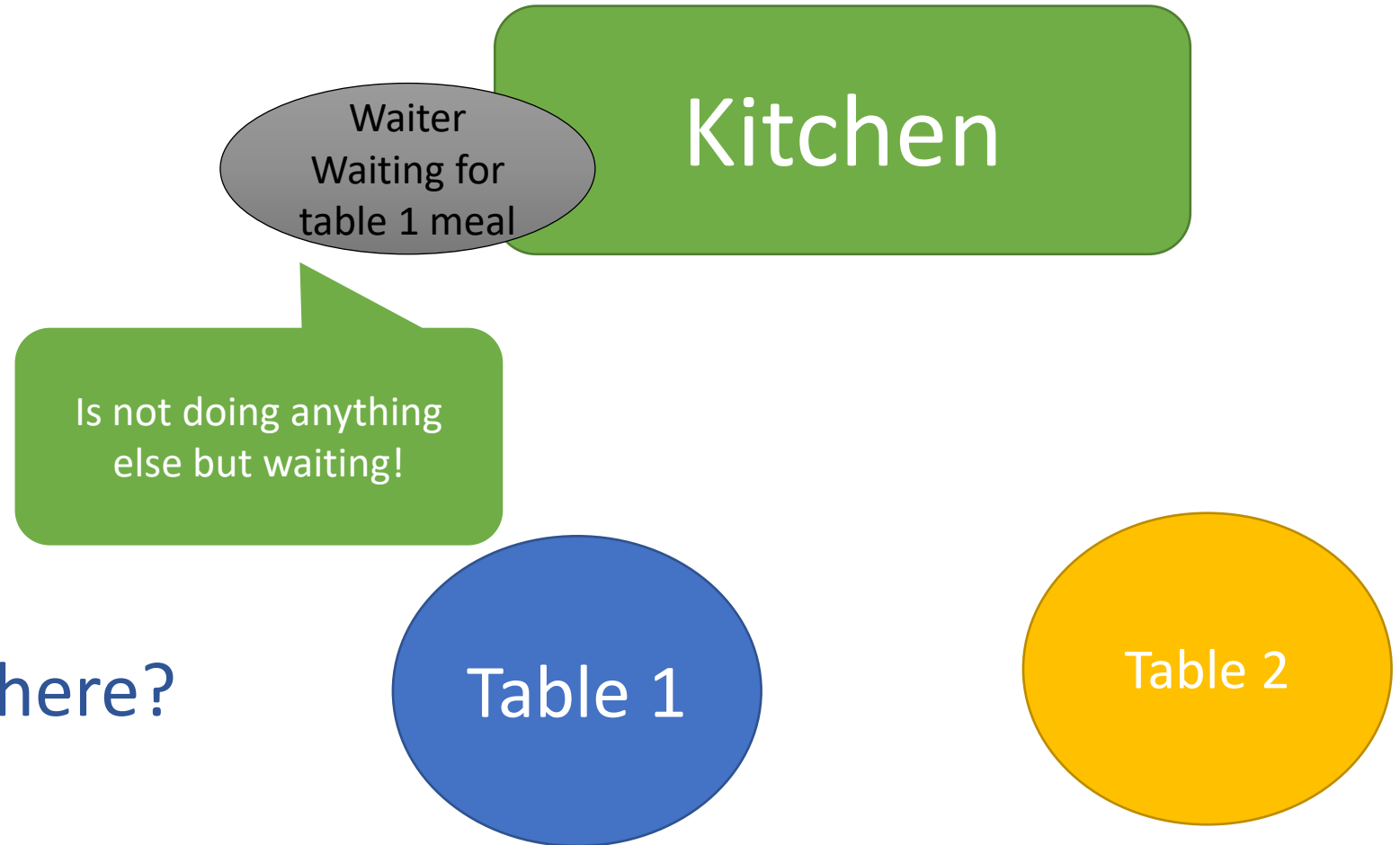


JavaScript(nodejs) is single threaded Asynchronous(non blocking) architecture

Example 1

# Blocking or synchronous (ASP. Net by nature )

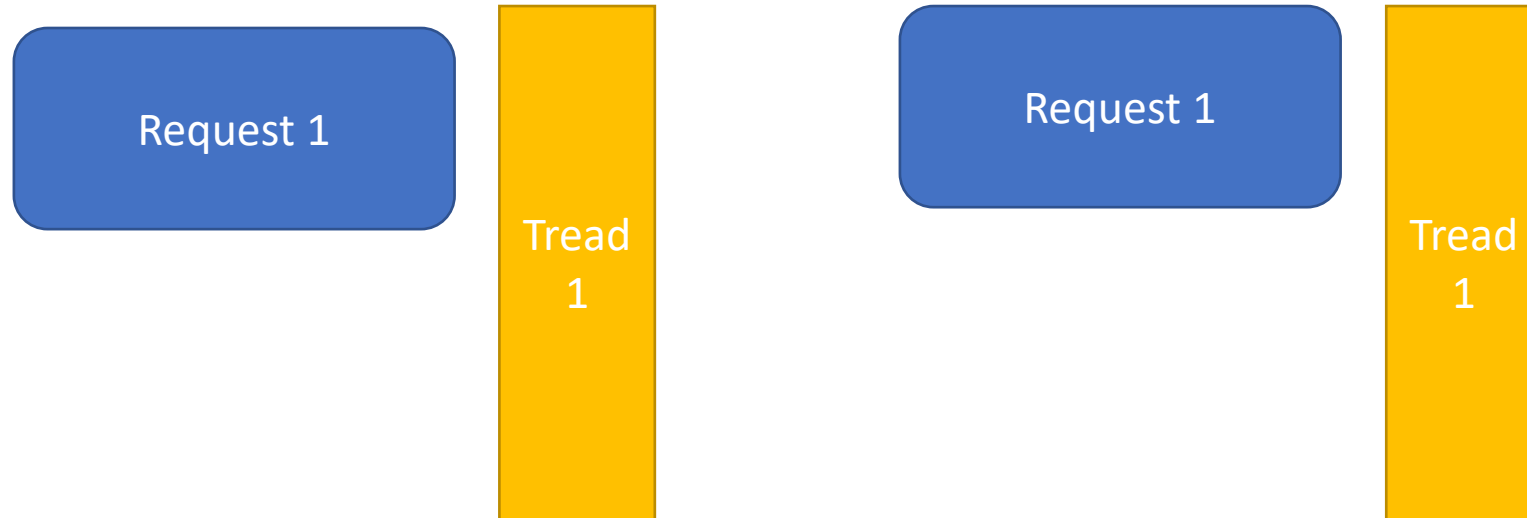
- Same waiter can server multiple tables, but every time waits for chief to prepare the meal then takes another order.



Q:What's the problem here?

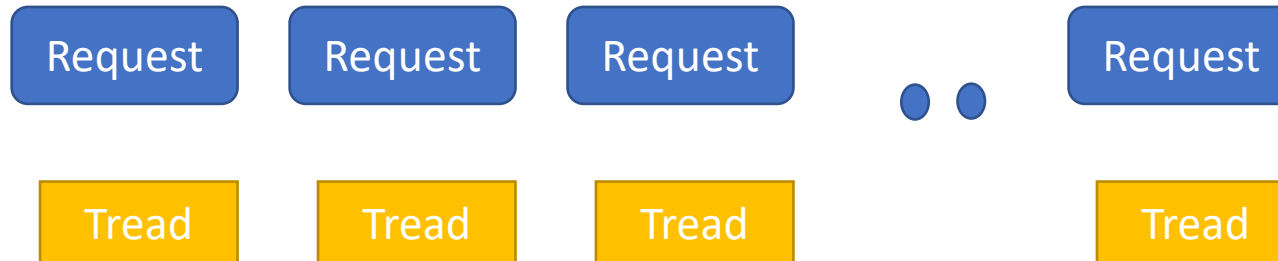
We are not utilizing our resources (the waiter) efficiently

# What if there are multiple requests for synchronous systems?



# Imaging 100 concurrent requests for a synchronous system.

Then the system has to create 100 threads! ( 100 waiters!!)



Lets get started with

node.js

Installation  
and  
Example code

4

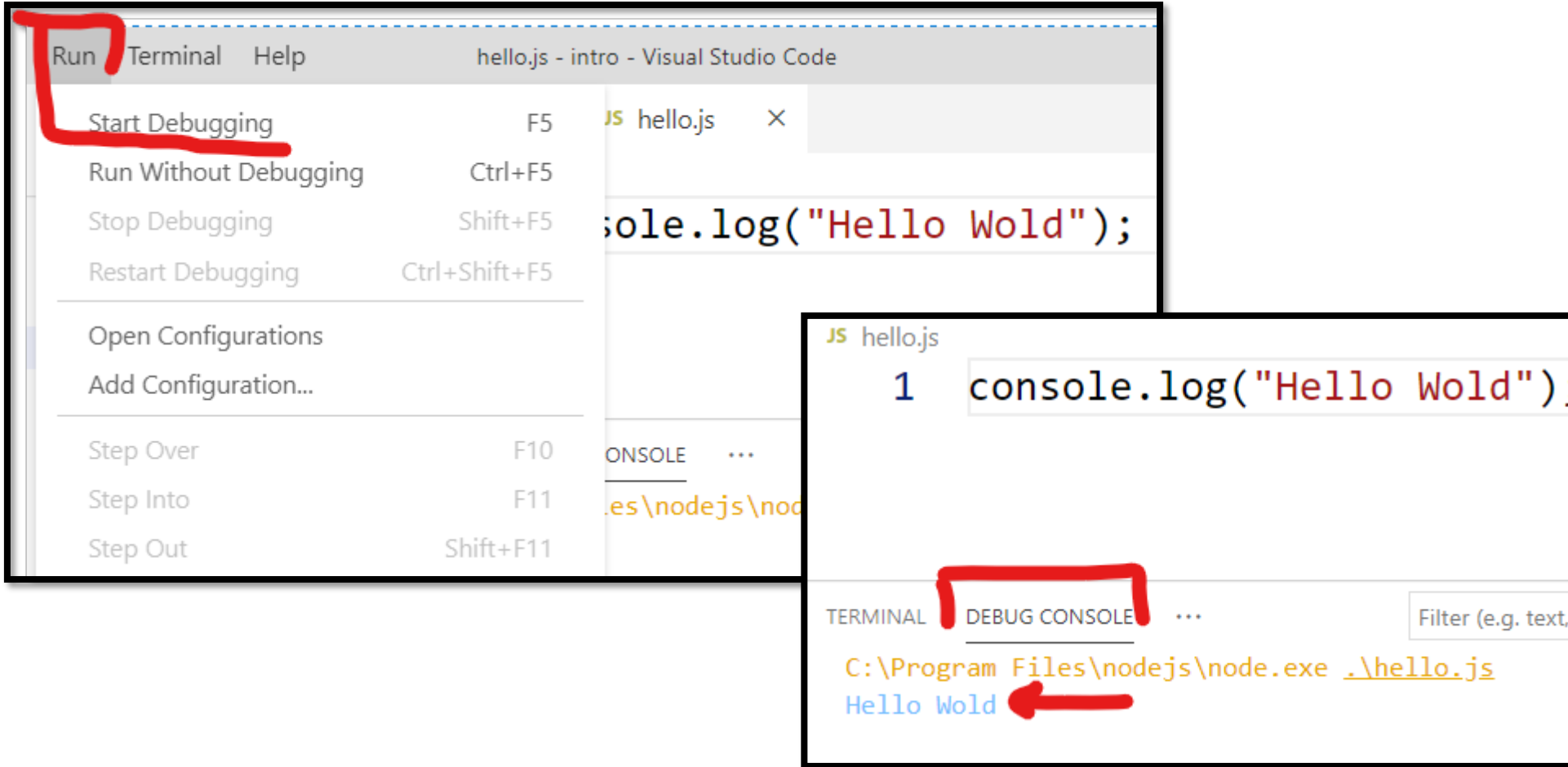


# Getting started

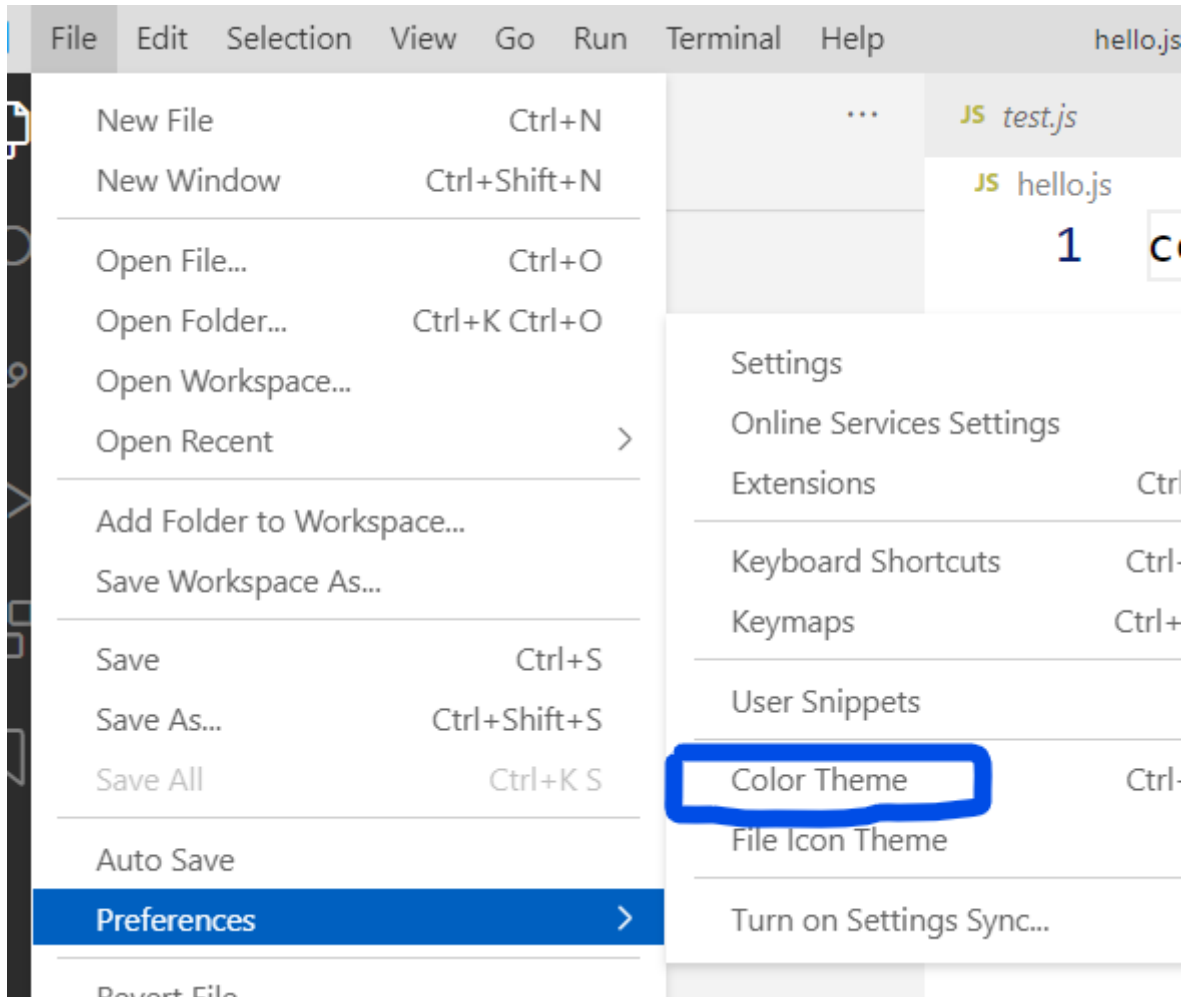
- Follow the instructions at ...
- **...NodeJs-Install-and-Debug.pdf**
- To **install** nodejs on your machine locally and to use Visual Studio Code to **debug**
- Using Nodejs you can easily develop a web server at your pc
- Then you send requests from your browser or the html files stored in your machine to the nodejs server you have developed

# Example 1 : Hello world!

- Just like any desktop programming language. Here console.log acts like System.out.println() in Java

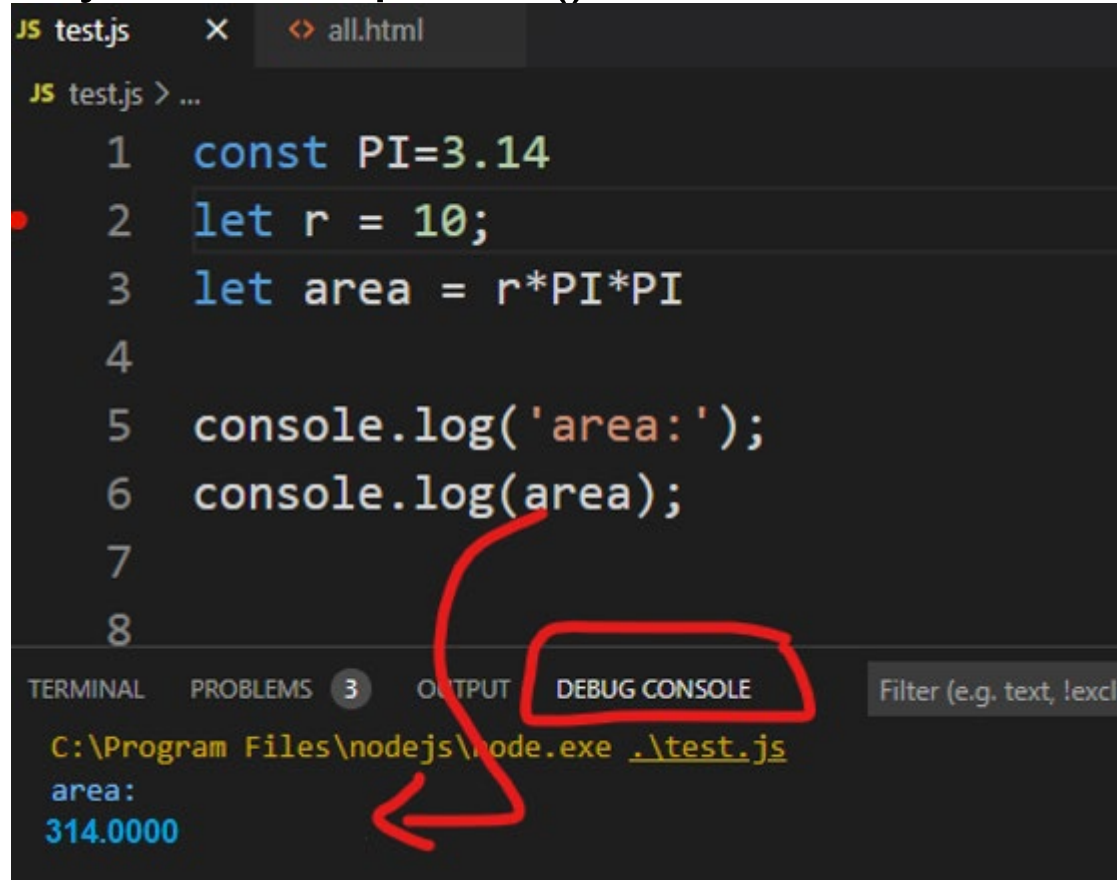


# You can change the look and feel of your VSC (Visual Studio Code)



## Example 2 : area calculation

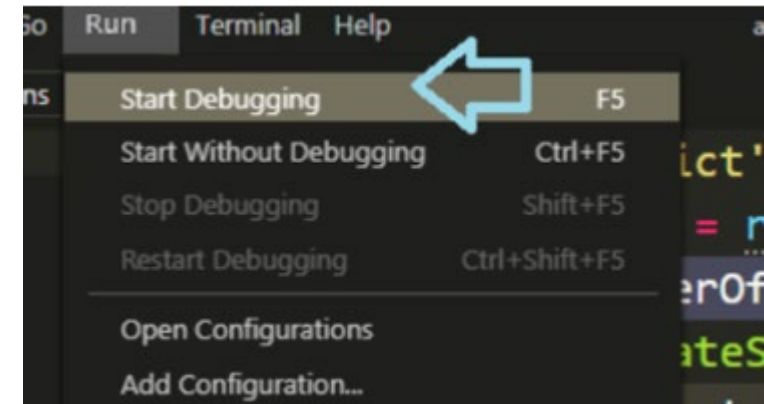
- Just like any desktop programming language. Here console.log acts like System.out.println() in Java



```
JS test.js  x  all.html
JS test.js > ...
1  const PI=3.14
2  let r = 10;
3  let area = r*PI*PI
4
5  console.log('area:');
6  console.log(area);
7
8

TERMINAL  PROBLEMS  3  OUTPUT  DEBUG CONSOLE  Filter (e.g. text, !excl
C:\Program Files\nodejs\node.exe .\test.js
area:
314.0000
```

- Note the break point on the line 2
- Bad function identifier/name !Do we really calculate the area of circle like this? Or is it just circumference



Follow the steps on "...-NodeJs-Install-and-Debug.pdf" to learn how to debug your server side nodejs code in VSC

5

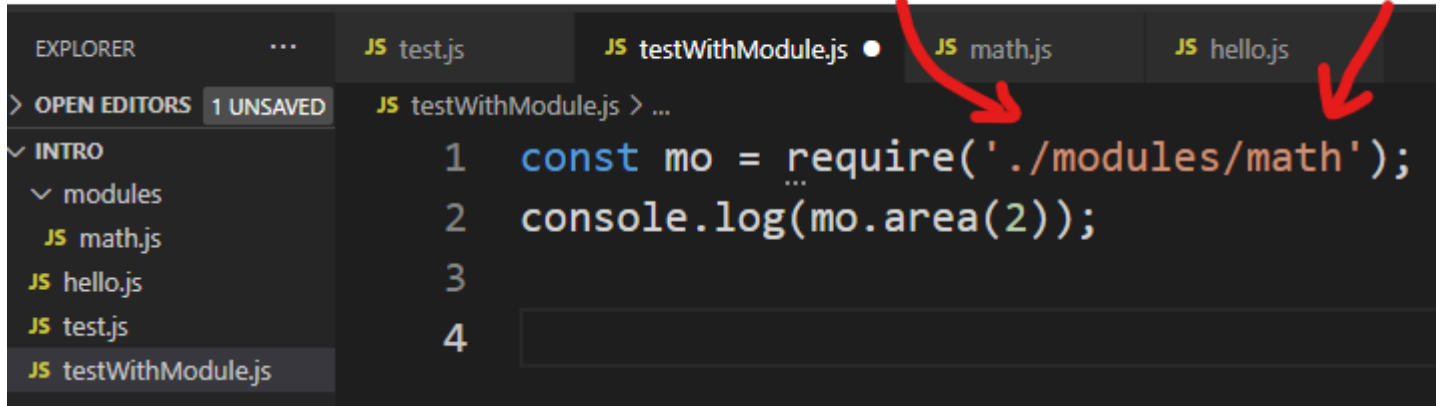
Put your functions  
in  
**Modules**

# What is a Module in Node.js?

- Consider modules to be the same as JavaScript libraries.
- A set of functions you want to include in your application.

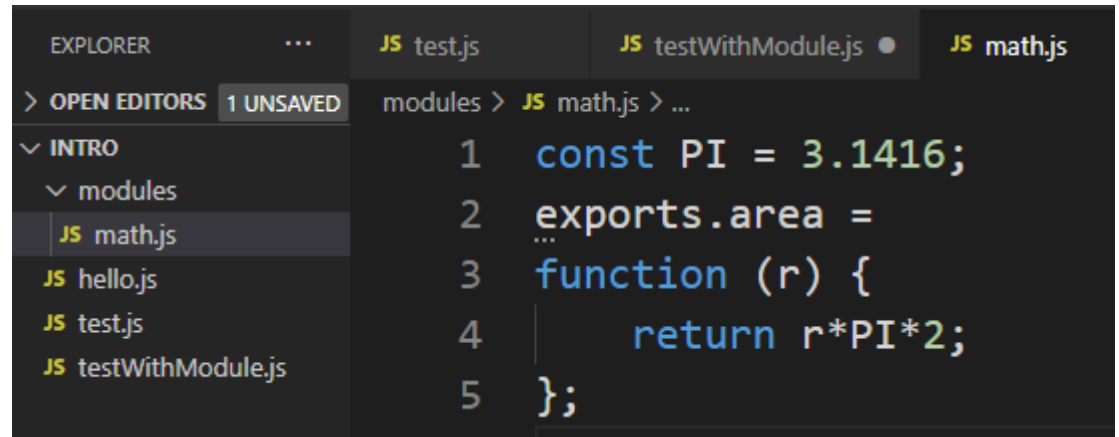
# Example 3, area calculation, using module

- 1. Our module math.js is located at modules/math.js
- Notice that we use ./ to locate the folder of our module
- That means that the folder is located in the same folder as the main Node.js file.
- 2. Notice we don't need to include the .js when we import the module (its optional)



This screenshot shows the VS Code interface with the Explorer on the left and the Editor on the right. The Explorer shows a project structure with an 'INTRO' folder and a 'modules' subfolder containing 'math.js', 'hello.js', 'test.js', and 'testWithModule.js'. The Editor has four tabs: 'test.js', 'testWithModule.js' (active), 'math.js', and 'hello.js'. Red arrows point from the numbers '1' and '2' in the text to the 'math.js' tab and the 'require' function call in the code, respectively.

```
1 const mo = require('./modules/math');
2 console.log(mo.area(2));
3
4
```



This screenshot shows the VS Code interface with the Explorer on the left and the Editor on the right. The Explorer shows the same project structure as the previous screenshot, but with 'math.js' selected in the Explorer. The Editor has three tabs: 'test.js', 'testWithModule.js', and 'math.js' (active). The code in 'math.js' defines a constant PI and an exported area function.

```
1 const PI = 3.1416;
2 exports.area =
3 function (r) {
4     return r*PI*2;
5 };
```

More examples using  
nodejs  
**built-in modules**



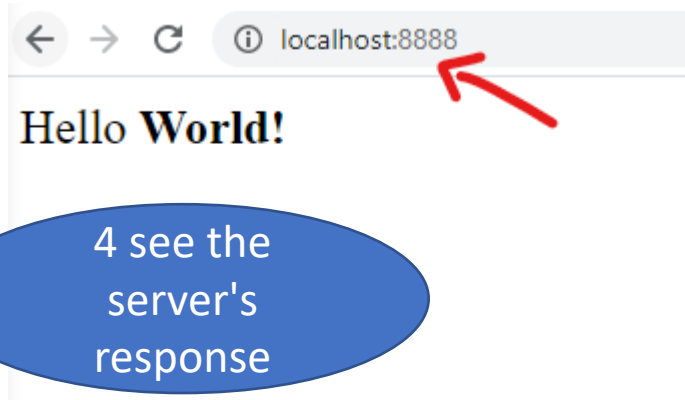
# Example 4 : a simple http server

- We are using the nodejs built-in module 'http'
- The server receives a request from the client ( the browser )

2 enter the url

3 the breakpoint hits.  
Press > to continue

4 see the server's response



A screenshot of the Visual Studio Code editor. The 'server.js' file is open, showing the following code:

```
1 const http = require('http');
2 http.createServer(function (req, res) {
3   res.writeHead(200, {'Content-Type': 'text/html'});
4   res.end('Hello <b>World!</b>');
5 }).listen(8888);
```

A blue oval callout at the top right says '1 run the server from Run menu'. A red arrow points from the 'console.log' statement on line 10 to the debug console at the bottom. The debug console shows the output: 'C:\Program Files\nodejs\node.exe .\server.js' and 'Server is running and listening ...'. A blue oval callout at the bottom left of the code editor says '3 the breakpoint hits. Press > to continue'.

## Example 4 : a simple http server ...

```
const http = require('http');

http.createServer(function (req, res) {
  console.log("The server recived a request ");
  res.writeHead(200, { "Content-Type": "text/html", "Access-Control-Allow-Origin": "*" });
  res.end('Hello <b>World!</b>');
}).listen(8888);
```

- 1 at your local browser type `http://localhost:8888/` and observe the result
- 2 replace `'text/html'` with just `'text'` and see what happens
- 3 now change this code to return a random number between 0 to 10 to client every time
- 4 what does `"Access-Control-Allow-Origin"` do? do we need it now?

## Example 4, returning server's current date (put functions in module)

```
let http = require('http');
```

```
let dt = require('./myModule');
```

```
http.createServer(function (req, res) {  
  res.writeHead(200, {'Content-Type': 'text/html'});  
  res.write("Now: " + dt.myDateTime());  
  res.end();  
}).listen(8888);
```

myModule.js

```
exports.area =  
function () {  
  return Date();  
};
```

# In-class activity

- Replace the 'text/html' with just 'text' and observe the response that the client receives

```
server.js  X  <> ajax.html
JS server.js > ...
1
2  const http = require('http');
3
4  http.createServer(function (req, res) {
5    console.log("Hi");
6    res.writeHead(200, {'Content-Type': 'text/html'});
7    res.end('Hello <b>World!</b>');
8  }).listen(8888);
9
10 console.log('Server is running and listening ...');
11
12
```

Example 5, using 'url' built-in module  
to extract the query strings in the url

## Using url built-in module to parse parameters in a string

```
let url = require('url');  
let adr =  
'http://localhost:8888/default.htm?name=John&age=23';  
let q = url.parse(adr, true);  
  
console.log(q.host); //returns 'localhost:8888'  
console.log(q.pathname); //returns '/default.htm'  
console.log(q.search); //returns '?name=John&age=23'  
  
let qdata = q.query; //returns an object: { name: John,  
age: 23 }  
console.log(qdata.name); //returns 'John'
```

# Example 5: http server with parameters

```
JS server.js
JS server.js > http.createServer() callback
1
2 let http = require('http');
3 let url = require('url');
4 http.createServer(function (req, res) {
5   let q = url.parse(req.url, true);
6   console.log(q.query); //returns the object form of '?name=John'
7   res.writeHead(200, {"Content-Type": "text/html"});
8   res.end('Hello ' + q.query["name"]);
9 }).listen(8888);
10
11 // at your browser address bar, try
12 //http://localhost:8888/?name=John
13
```

TERMINAL PROBLEMS OUTPUT DEBUG CONSOLE

C:\Program Files\nodejs\node.exe .\server.js

> {name: 'John'}

- Note! I did not need "Access-Control-Allow-Origin": "\*"
- Why?

# 'fs', the file system built-in module

- We are going to be using another built-in module, 'fs', file system
- allows us to work with the file system on our computer or the server.
-



## Example 6: Creating a file server in node js

- Create a Node.js file that opens the requested file and returns the content of the file to the client. If the file does not exist, throw a 404 error!

# Example 6: Creating a file server in node js

```
const http = require('http');
const url = require('url');
const fs = require('fs');
http.createServer(function (req, res) {
  const q = url.parse(req.url, true);
  const filename = "." + q.pathname;
  fs.readFile(filename, function (err, data) {
    if (err) {
      res.writeHead(404, { 'Content-Type': 'text/html' });
      return res.end(q.pathname + " 404 Not Found!");
    }
    res.writeHead(200, { 'Content-Type': 'text/html' });
    res.write(data);
    return res.end();
  });
}).listen(8888);

// at your browser address bar, try
//http://localhost:8888/file.txt
```

4xx are status message codes meaning client error.

That means client request was not valid, either due to authentication or the client asked for a file which did not exist

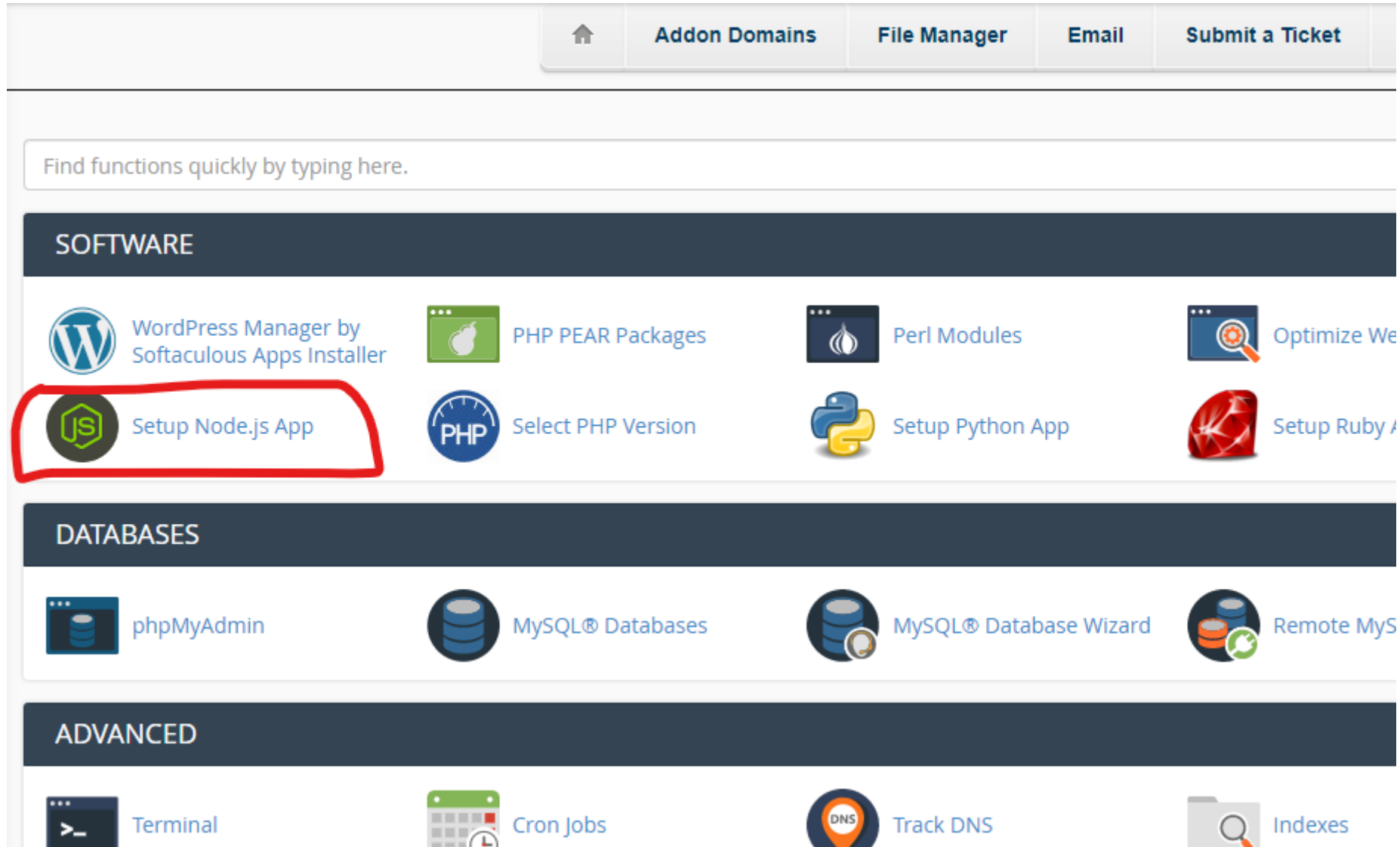
404: not found

HTTP Status Messages:

[https://www.w3schools.com/tags/ref\\_httpmessages.asp](https://www.w3schools.com/tags/ref_httpmessages.asp)

# Setting up node js app on shared hosting with cPanel

- Refer to the lecture videos



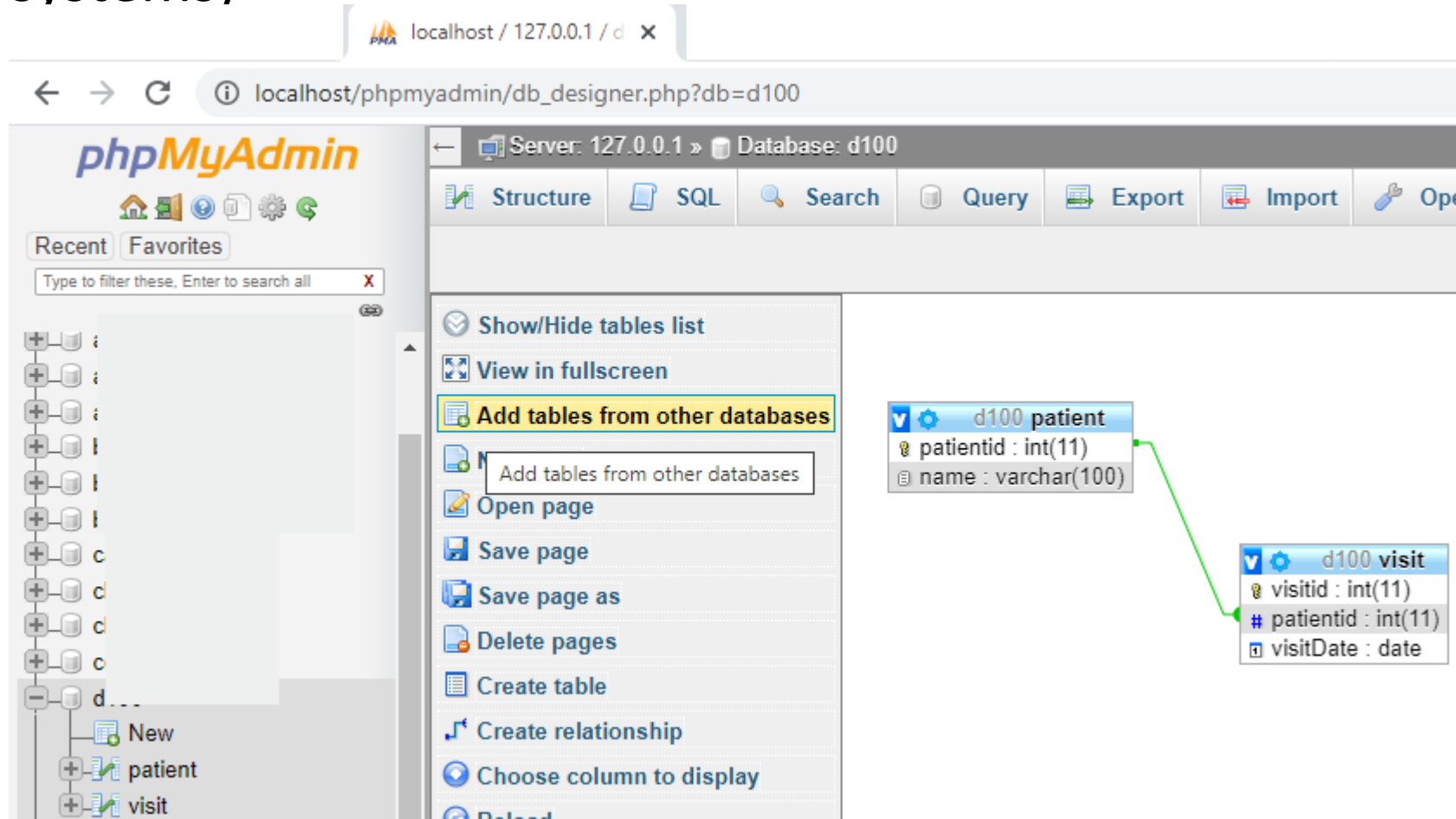
7

8

Tips for future


# Database server we use in this course

- We will use the free and open source relational DB administration tool
- Called phpMyAdmin which is browser based ( thus working on all operating systems)



- You will learn how to use XAMPP to install phpMyAdmin on your machine
- ( in two weeks)

XAMPP Control Panel v3.2.4 [ Compiled: Jun 5th 2019 ]

XAMPP Control Panel v3.2.4

Service	Module	PID(s)	Port(s)	Actions
<input type="checkbox"/>	Apache	3212 5092	80, 443	<input type="button" value="Stop"/> <input type="button" value="Admin"/> <input type="button" value="Config"/> <input type="button" value="Logs"/>
<input type="checkbox"/>	MySQL	25796	3306	<input type="button" value="Stop"/> <input type="button" value="Admin"/> <input type="button" value="Config"/> <input type="button" value="Logs"/>
<input type="checkbox"/>	FileZilla			<input type="button" value="Start"/> <input type="button" value="Admin"/> <input type="button" value="Config"/> <input type="button" value="Logs"/>
<input type="checkbox"/>	Mercury			<input type="button" value="Start"/> <input type="button" value="Admin"/> <input type="button" value="Config"/> <input type="button" value="Logs"/>
<input type="checkbox"/>	Tomcat			<input type="button" value="Start"/> <input type="button" value="Admin"/> <input type="button" value="Config"/> <input type="button" value="Logs"/>

11:28:51 AM [Tomcat] Checking for required tools...

11:28:51 AM [Tomcat] Checking for service (name="Tomcat7"): Service not installed

11:28:51 AM [Tomcat] Service Path: Service Not Installed

11:28:51 AM [Tomcat] Checking default ports...

11:28:51 AM [main] Enabling autostart for module "Apache"

11:28:51 AM [main] Enabling autostart for module "MySQL"

11:28:51 AM [main] Starting Check-Timer

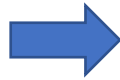
11:28:51 AM [main] Control Panel Ready

11:28:51 AM [Apache] Autostart active: starting...

# ES6 tips of the week/ Arrow functions to replace old ones

- regular functions

```
setTimeout(function() {  
  }, timeout);
```



- arrow functions

```
setTimeout(() => {  
  }, timeout);
```

```
date = function() {  
  return new Date();  
}
```



```
date = () => {  
  return new Date();  
}
```

```
// one parameter  
hello = (val) => "Hello " + val;  
  
hello = val => "Hello " + val;  
  
// two or more parameters  
sum = (a,b) => a+b;
```

# Resources

- [https://www.w3schools.com/nodejs/nodejs\\_intro.asp](https://www.w3schools.com/nodejs/nodejs_intro.asp)
- <https://en.wikipedia.org/wiki/Node.js>
- On JS being asynchronous: <https://youtu.be/6oP-c30s7co?t=6m8s>