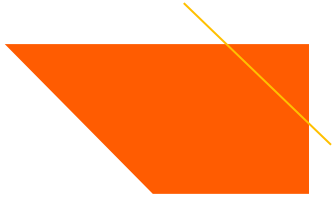
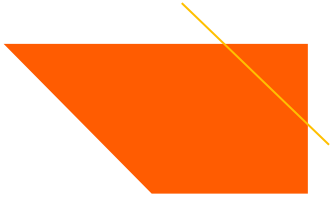
Web Application Development

COSI 152A

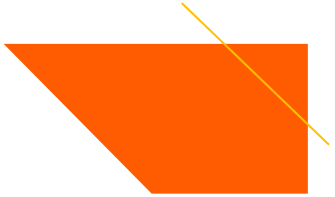


Node.js



Node.js

- Node.js is a platform for interpreting JavaScript code and running applications.
- Node.js is built with Google Chrome’s JavaScript engine, and it’s considered to be powerful and able to support JavaScript as a server-side language.
- Developers can now commit to mastering JavaScript to build a complete web application instead of having to master multiple languages to accomplish the same task.



Why Developing with Node.js?

- Full stack development using JavaScript and Node.js has the following advantages:
  - A single language (JavaScript) as the core language for development.
  - Node.js is backed by Google's V8 JavaScript interpreter.
  - Large and experienced community of developers.
  - More supportive, open-source tools are now being built.
  - More jobs are available.

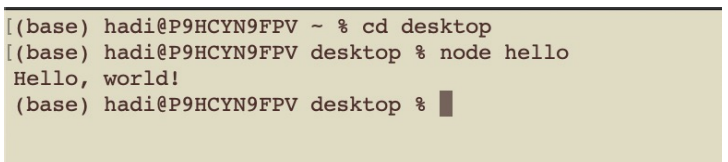
Running JavaScript file with Node.js

- Write your JS code using a text editor and save it with ".js" extension.
- Open a terminal and navigate to your file location.
- Run your JavaScript code:

node <file-name>

```
console.log("Hello, world!");
```

hello.js



REPL Environment

- REPL environment can have access to your JavaScript file content.
  - For example, you can access a variable in your code with REPL.

```
let messages = [
  "A change of environment can be a good thing!",
  "You will make it!",
  "Just run with the code!"
];
```

List an array of messages.

- Load the JavaScript file into REPL

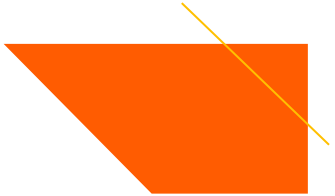
```
> .load messages.js
"use strict";
let messages = [
  "A change of environment can be a good thing!",
  "You will make it!",
  "Just run with the code!"
];
```

Loading an array with three strings

- Use the file's content in REPL

```
> messages.forEach(message => console.log(message));
```

Log each message by using a single-line arrow function.



Module

- A Node.js application is made up of many JavaScript files that have access to one another’s contents when necessary.
  - Each JavaScript file or folder containing a code library is called a module.
  - Using module helps you make your code more organized.

Module

- Consider messages.js file:

```
let messages = [
  "A change of environment can be a good thing!",
  "You will make it!",
  "Just run with the code!"
];
```

List an array of messages.

- Keep messages separate from the code to display them.
- To manage these messages in another file, change the variable definition:

```
exports.messages  = [
    "A change of environment can be a good thing!",
    "You will make it!",
    "Just run with the code!"
  ];
```

List an array of messages.

- The module is ready to be required (imported) by another JavaScript file:

printMessages.js

```
const messageModule = require("./messages");
messageModule.messages.forEach(m  => console.log(m));
```

Require the local messages.js module.

Refer to the module's array through messageModule.messages.

ForEach loop

- In JavaScript, the forEach loop is a higher-order function used to iterate through the elements of an array and perform a specified action (a callback function) on each element.

```
array.forEach(function(currentValue, index, array) {
    // Your code to process the current element goes here
});
```

```
messageModule.messages.forEach(function(m){
    console.log(m);
});
```

- A callback function is a function that is passed as an argument to another function and is executed after the completion of that function.

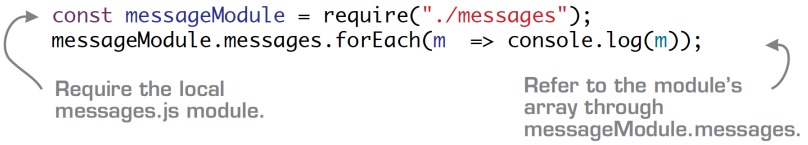
```
function(currentValue, index, array) {
    // Your code to process the current element goes here
};
```

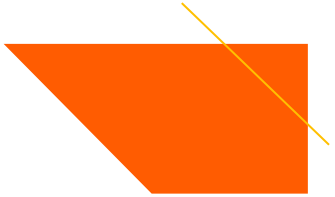
```
function(m){
    console.log(m);
};
```

Arrow function

- Arrow functions were introduced in ECMAScript 6 (ES6) and provide a more compact syntax compared to traditional function expressions.
- Arrow functions are often used for simple functions and callbacks.
- Here's the basic syntax:

```
(parameter1, parameter2, ..., parameterN) => { expression }
```





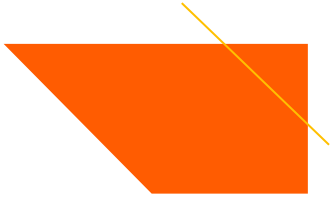
Node Package Manager (npm)

- With your installation of Node.js, you also got npm, a package manager for Node.js.
- npm is responsible for managing the external packages in your application.
  - modules that others built and made available online
- Use npm to install, remove, and modify packages.
- Type `npm -l` in terminal to see a list of npm commands with explanations.

npm Commands

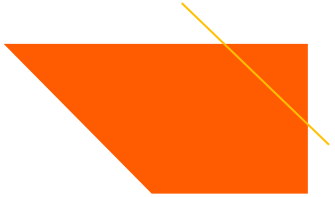
npm command	Description
`npm init`	Initializes a Node.js application and creates a package.json file
`npm install <package>`	Installs a Node.js package
`npm publish`	Saves and uploads a package you build to the npm package community
`npm start`	Runs your Node.js application (provided that the package.json file is set up to use this command)
`npm stop`	Quits the running application
`npm docs <package>`	Opens the likely documentation page (web page) for your specified package

- `npm install <package>` ,appending --save (or –S) installs the package as a dependency for your application.
  - This command extension is called flag.
- `npm uninstall <package>` reverses the install action.



Node packages

- Functionalities can be incorporated in your application by finding a package that performs the task at https://www.npmjs.com.
- E.g., Adding the ability in a restaurant application to find where the users are located based on their ZIP codes.
- To add this feature, install the cities package which converts text addresses to location coordinates.
- To install this package, first initialize a Node.js project and create a package.json file that npm will use to install cities.



Node.js Applications

- Every Node application or module contains a package.json file
  - The file define the properties of that project.
  - The file lives at the root level of your project.
  - Specify the version of the current release, name of your application, and the main application file.
- This file is important for npm to save any packages to the node community online.

Initializing a Node.js Application

- To initialize a Node application:
  - create your application folder with a proper name (e.g. recipe_connection)
  - Navigate to the folder and initialize your application using: `npm init`
  - You'll be prompted for details of your project such as author, the entry point etc.
  - enter your name, use main.js as the entry point, and press Enter to accept all the default options.

Installing a Node.js package

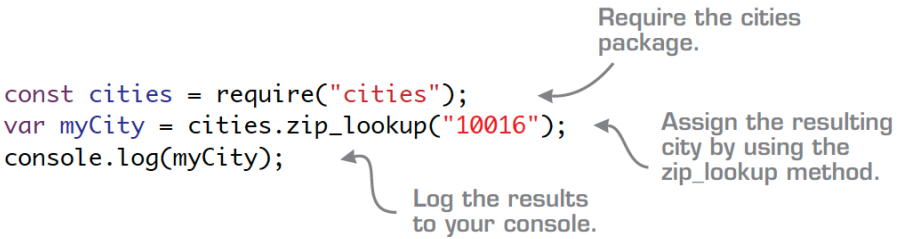
- Now your application has a starting point for saving and managing application configurations and packages.
- Navigating to your project folder and run: `npm install cities --save`
- Your package.json file after installation of cities

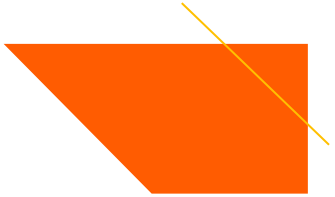
```
{
  "name": "recipe_connection",
  "version": "1.0.0",
  "description": "An app to share cooking recipes",
  "main": "main.js",

  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "author": "Jon Wexler",
  "license": "ISC",
  "dependencies": {
    "cities": "^2.0.0"
  }
}
```

Display dependencies section of package.json.

Installing a Node.js package

- Start testing the newly installed package inside index.js





Thank You!