Contents

[Previous Class Topics 2](#_Toc45986719)

[Class Objectives 2](#_Toc45986720)

[Upcoming topics 3](#_Toc45986721)

[Chrome Developer Tools 7](#_Toc45986722)

[Screenshot a single element 7](#_Toc45986723)

[Multi-Line Console Entry 7](#_Toc45986724)

[Watch 7](#_Toc45986725)

[Use previous operation value 7](#_Toc45986726)

[Using $ for more 7](#_Toc45986727)

[Emulating different network speeds 7](#_Toc45986728)

[Emulate devices 8](#_Toc45986729)

[Changing a properties state 8](#_Toc45986730)

[Make JSON readable 8](#_Toc45986731)

[Changing Words You See 8](#_Toc45986732)

[Unused CSS 9](#_Toc45986733)

[Calculate The Code Execution Time 9](#_Toc45986734)

[Use Copy Styles to Learn About How Things Are Used 9](#_Toc45986735)

[Using console.table 9](#_Toc45986736)

Class : Learning Node For Everyone

Start Date : 07/18/2020

Delayed class from July 4th holiday

## Previous Class Topics

* Introduction
  + What You Need
  + What is Node
  + Resources
  + Contacting Instructors
  + Class structure
* Getting Started
  + Installing
  + Setting up
    - Node
    - Mongo DB
    - MySQL
    - Editors (IDE)
* Developing Code (all languages)
  + Introduction to Concepts and Tools
  + Best Practices
* How node works
  + Your first server
    - Testing your first server
  + REPL Terminal
    - REPL commands
    - Using REPL
  + Online editor
    - Using online editor
* Your first homework
  + Starter code
  + Assignment

## Class Objectives

* Run maze in browser tester
* Talk about comments
* Show developer tools in Chrome
* Assignment

## Upcoming topics

* Async and Node
* DB with Node (Mongo & MySQL)
* Programming to handle errors
* Securing your code
* Modeling Data
* Using PUG
* Advanced Concepts

Running last class MAZE starter code in the browser.

To do this there are several ways you can do it. I did it simply by removing calls to referenced objects in the code so it appears as a single program. This is the most simplistic way to get the maze to run as is in the online tester. Here is that code.

/\*\*

\* Generates the maze asynchronously.

\* @param {Number} x Width of the maze.

\* @param {Number} y Height of the maze.

\* @returns {Promise} finished when resolved.

\*/

function maze(x,y) {

return new Promise((resolve, reject) => {

let n=x\*y-1;

if (n<0) {

reject(new Error(`illegal maze dimensions (${x} x ${y} < 1)`));

} else {

let horiz =[]; for (let j= 0; j<x+1; j++) horiz[j]= [];

let verti =[]; for (let j= 0; j<x+1; j++) verti[j]= [];

let here = [Math.floor(Math.random()\*x), Math.floor(Math.random()\*y)];

let path = [here];

let unvisited = [];

for (let j = 0; j<x+2; j++) {

unvisited[j] = [];

for (let k= 0; k<y+1; k++)

unvisited[j].push(j>0 && j<x+1 && k>0 && (j != here[0]+1 || k != here[1]+1));

}

while (0<n) {

let potential = [[here[0]+1, here[1]], [here[0],here[1]+1],

[here[0]-1, here[1]], [here[0],here[1]-1]];

let neighbors = [];

for (let j = 0; j < 4; j++)

if (unvisited[potential[j][0]+1][potential[j][1]+1])

neighbors.push(potential[j]);

if (neighbors.length) {

n = n-1;

let next= neighbors[Math.floor(Math.random()\*neighbors.length)];

unvisited[next[0]+1][next[1]+1]= false;

if (next[0] == here[0])

horiz[next[0]][(next[1]+here[1]-1)/2]= true;

else

verti[(next[0]+here[0]-1)/2][next[1]]= true;

path.push(here = next);

} else

here = path.pop();

}

resolve({x: x, y: y, horiz: horiz, verti: verti});

}

});

}

/\*\*

\* A mere way of generating text.

\* The text (Since it can be large) is generated in a non-blocking way.

\* @param {Object} m Maze object.

\* @param {Stream} writeTo Optinally, include here a function to write to.

\* @returns {Promise} finished when the text is generated.

\*/

function display(m, writeTo) {

return new Promise((resolve, reject) => {

let text = [];

for (let j= 0; j<m.x\*2+1; j++) {

let line = [];

if (0 == j%2)

for (let k=0; k<m.y\*4+1; k++)

if (0 == k%4)

line[k] = '+';

else

if (j>0 && m.verti[j/2-1][Math.floor(k/4)])

line[k] = ' ';

else

line[k] = '-';

else

for (let k=0; k<m.y\*4+1; k++)

if (0 == k%4)

if (k>0 && m.horiz[(j-1)/2][k/4-1])

line[k] = ' ';

else

line[k] = '|';

else

line[k] = ' ';

if (0 == j) line[1] = line[2] = line[3] = ' ';

if (m.x\*2-1 == j) line[4\*m.y]= ' ';

text.push(line.join('')+'\r\n');

}

const OUTPUT = text.join('');

if (typeof writeTo === 'function')

writeTo(OUTPUT);

resolve(OUTPUT);

});

}

module.exports = {

maze: maze,

display: display

}

const X = 20,

Y = 20;

console.log(`Generating a maze of ${X} x ${Y}...`);

const origin = new Date().getTime();

maze(X, Y).then((m) => {

const time = new Date().getTime() - origin;

console.log(`Done in ${time <= 1000 ? time+'ms' : Math.round(time/1000)+'s'}!`);

display(m, console.log);

//Here you can pass a given stream (ie: stream) and it's write function;

//An example could be: maze.display(m, stream.write);

}, (err) => console.error(err));

# Chrome Developer Tools

Things you should know to be a good debugger and developer of web applications.

## Screenshot a single element

Select an element and press cmd-shift-p (or ctrl-shift-p in Windows) to open the Command Menu, and select Capture node screenshot

## Multi-Line Console Entry

To write commands that span over multiple lines in the Console, press shift-enter.

Once you’re ready, press enter at the end of the script to execute it.

## Watch

Instead of writing again and again a variable name or an expression you are going to check a lot during a debug session, add it to the Watch Expression list.

## Use previous operation value

Use $\_ to reference the return value of the previous operation executed in the Console

## Using $ for more

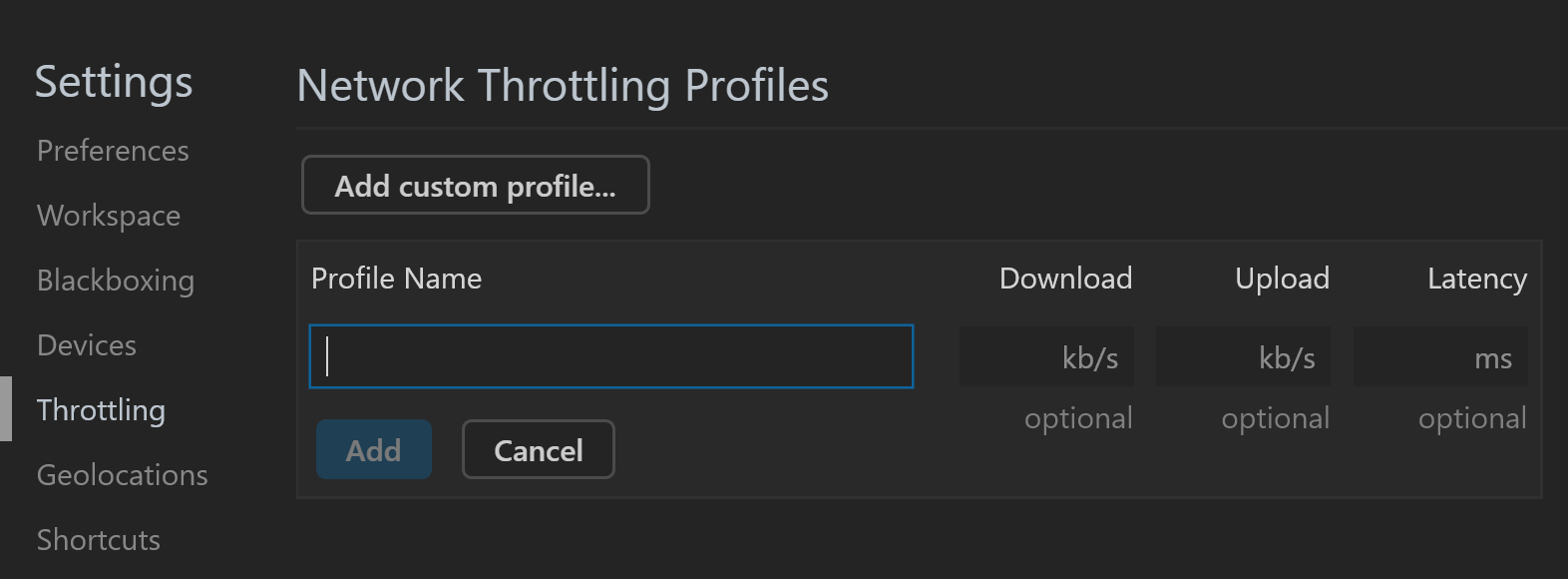
There are a few more tricks up the $ sleeve. Sometimes, a selector might be too complicated to write by heart, or you just don't know a selector specific enough. If you pick an element in the Elements tab you can retrieve it with the $0 variable in the console.

The console goes even further, allowing us to access not only the last selection, but the last 5, in order. The selections are exposed through the $0 - $4 variables

## Emulating different network speeds

Using the Network tab again we can test our site in various internet speeds. The default preset is online, and you'll enjoy the full bandwidth of your internet connection.

Besides online, there are a few more presets available: Fast 3G, Slow 3G and offline, which vary in upload speed, download speed and latency. If you need to emulate a different, more exotic speed, you can add your own profile through the Add... button:



## Emulate devices

Those of us working on responsive applications know the feeling where you work really hard to make a beautiful layout, only to see it misbehave on devices with different resolutions.

When you click on the button that looks like a tablet and a phone, you'll see that your browser's viewport changes to reflect a different device's dimensions.

You can choose a device from a list of presets containing various popular devices, such as iPhone X, iPad Pro, Pixel 2, Pixel 2 XL and more. The list is admittedly a bit outdated, but I think it's good enough for most cases.

If you can't find a device that fits your needs, you can set a custom resolution.

## Changing a properties state

Have you ever faced a situation where you wanted to play with an element's :hover-specific CSS, but every time you moved your mouse to the styles section in the dev tools the element was no longer hovered?

Well, Chrome dev tools exposes a nice way to lock an element's state, so you can fiddle with its properties with peace. This way you can quickly toggle an element's :active, :hover, :focus, :focus-within and :visited properties:

## Make JSON readable

Type the following commands in the console and you will get beautifully formatted JSON

let book = {"date": "2019–03–22","book": "Harry potter","author": "J.K.Rowling"};

JSON.stringify(book, null, 2)

## Changing Words You See

Now here comes a cool trick that lets you edit any text displayed on the website. This trick avoids the need of selecting the element and editing it in elements panel. To do this,

1. Goto any website.

2. Execute the below code in your console to make the site editable

document.designMode = 'on'

3. Now click on any heading or text from the website and you can directly start typing and changing the text displayed.

## Unused CSS

Using this technique, you will be able to quickly find the redundant css that is not used anywhere on the site.

This allows to minimize the css file size by removing that unused code.

1. Goto any tab like console tab and press Escape key.

2. You will see the coverage tab. (Click on the three dots on the left side and select coverage if coverage tab is not displayed for you by default)

3. Click on the reload button displayed to start coverage

4. It will display all the js and css files

5. Search for .css in the search box to filter the result

6. Double click on any .css file and it will show you the unused css by highlighting it with red color

## Calculate The Code Execution Time

console.time and console.timeEnd functions allows us to find out the time taken for executing a particular code.

Type in the following.

console.time('startManyLoops');

for (let x = 1; x < 10000000000; x++);

console.timeEnd('startManyLoops');

## Use Copy Styles to Learn About How Things Are Used

## Using console.table

Instead of using console.log for table and JSON you can use console.table to see it formatted nicely. I provided you a simple server to print it to a server console also. It is called dev\_tools.js. Call it with node dev\_tools.js then browse to the server in a browser. http://127.0.0.1:8081/

const ages = [32, 15, 19, 12];

let responseMsg = "";

// ???? is there at least one adult in the group?

const adultPresent = ages.some(age => age >= 18);

responseMsg = responseMsg + "\n" + adultPresent;

console.table(ages);

// ?? is everyone old enough to drink?

const allOldEnough = ages.every(age => age >= 19);

console.log(allOldEnough);