Hands On Coding Workshop

Objectives

- Learn more about iterative product development
- Learn about building blocks of web applications
  - Frontend (HTML, Javascript, CSS)
  - APIs and libraries

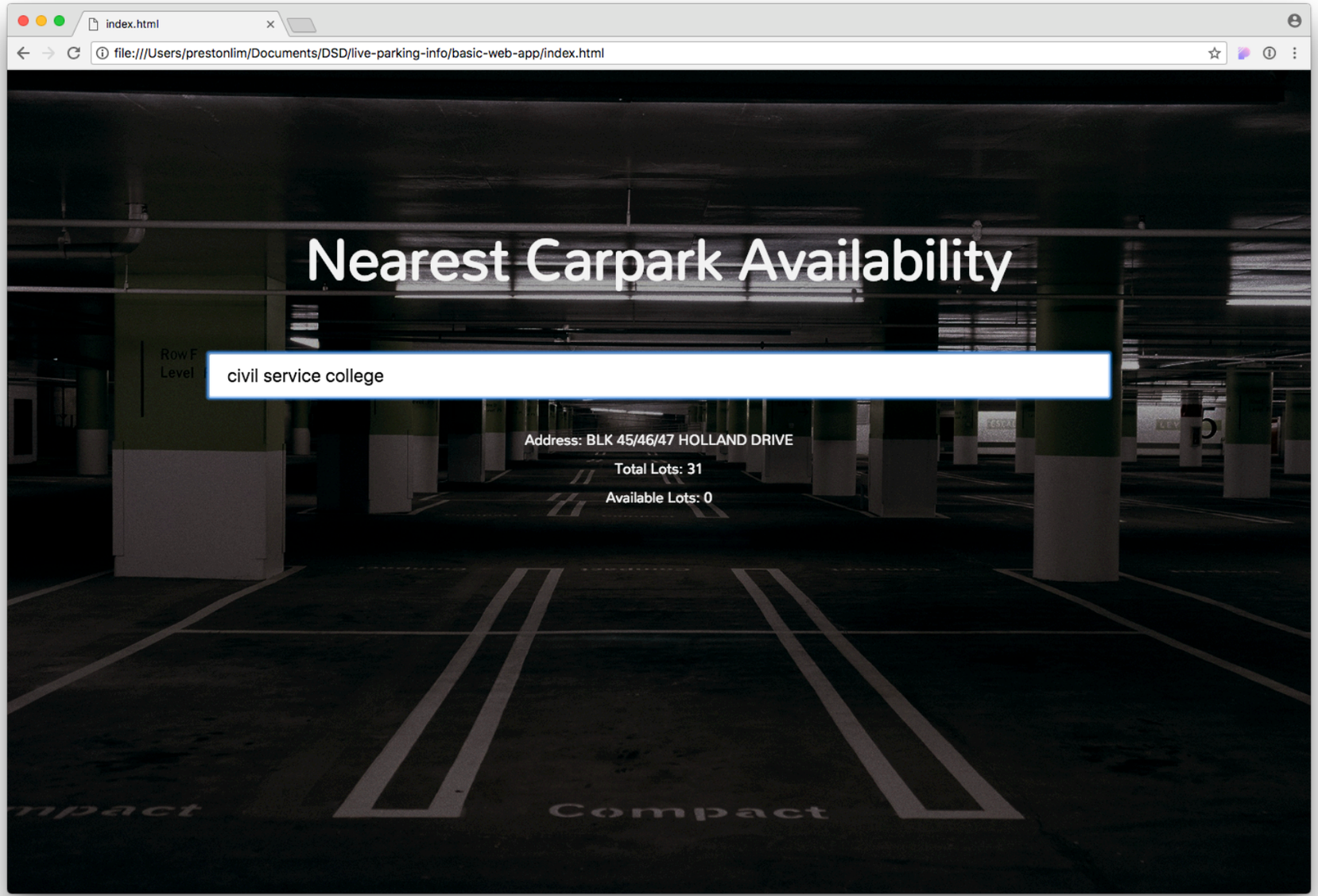
Overview

- Sharing on Parking.sg
- Hands on coding workshop

Sharing on Parking.sg

Hands on coding workshop

What you're going to build



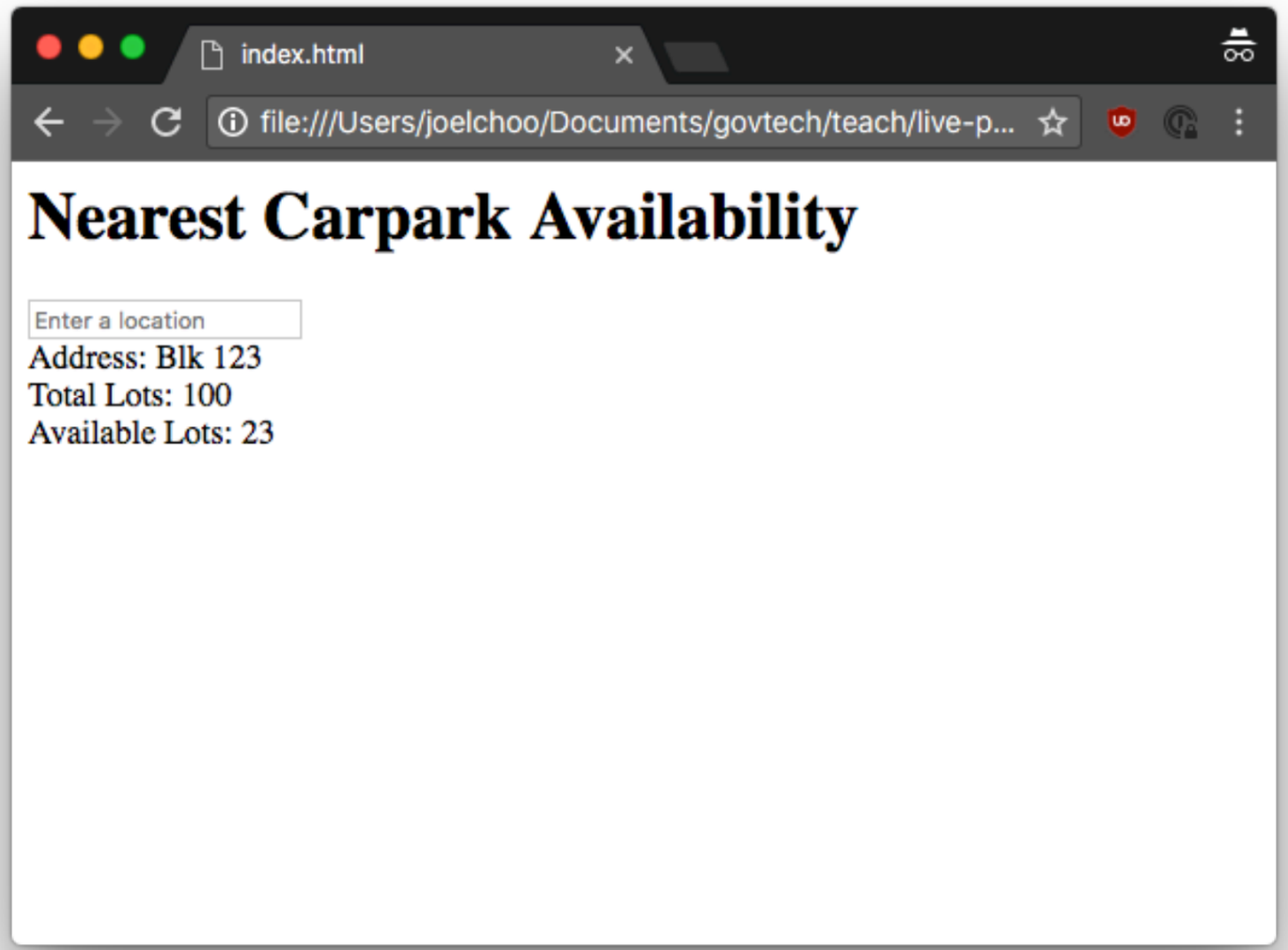
Outline

1. Introduction to HTML - building the skeleton of our site (Joel)
2. Introduction to Javascript - adding interactivity to our site (Nikhil)
3. Introduction to APIs - fetching and displaying data on the site (Arshad)
4. Introduction to CSS - styling the site (Preston)
5. Deploying the site to the internet (Joel)
6. Bonus - A simple API server

Before we begin

- Slides are provided as handouts
- You can also refer to digital copy of the slides

Step 1 - Displaying some text



Setup

- Open the `index.html` file in **Atom**
  - We will be editing our code in Atom
- Open the `index.html` file in **Google Chrome**
  - We will be able to see our progress in Chrome by refreshing the page

`hello`

- Save the file and refresh the page in Chrome
- Websites are just text files interpreted by the browser
- HTML files specify the content that the browser should display

```
<body>
  hello
</body>
```

- HTML uses tags to differentiate between different parts of the page
- <body> - opening tag for the page's content
- </body> - closing tag
- Everything between opening and closing tags is the tag's content

```
<body>
  hello there!
</body>
```

```
<body>
  hello

  there!
</body>
```

- HTML has a specific way of formatting content
- What you see is not necessarily what you get

```
<body>
  <h1>hello</h1>

  there!
</body>
```

- The `h1` tag makes the content inside become a **header**

```
<body>
  <h1>Nearest Carpark Availability</h1>
</body>
```

```
<body>
  <h1>Nearest Carpark Availability</h1>
  <input>
</body>
```

- `input` creates an input box
- `input` is a basic element, does not require a closing tag

```
<body>
  <h1>Nearest Carpark Availability</h1>
  <input placeholder="Enter a location">
</body>
```

- HTML elements/tags can have extra **attributes** given to them
- These **attributes** can change the behaviour and appearance of the element

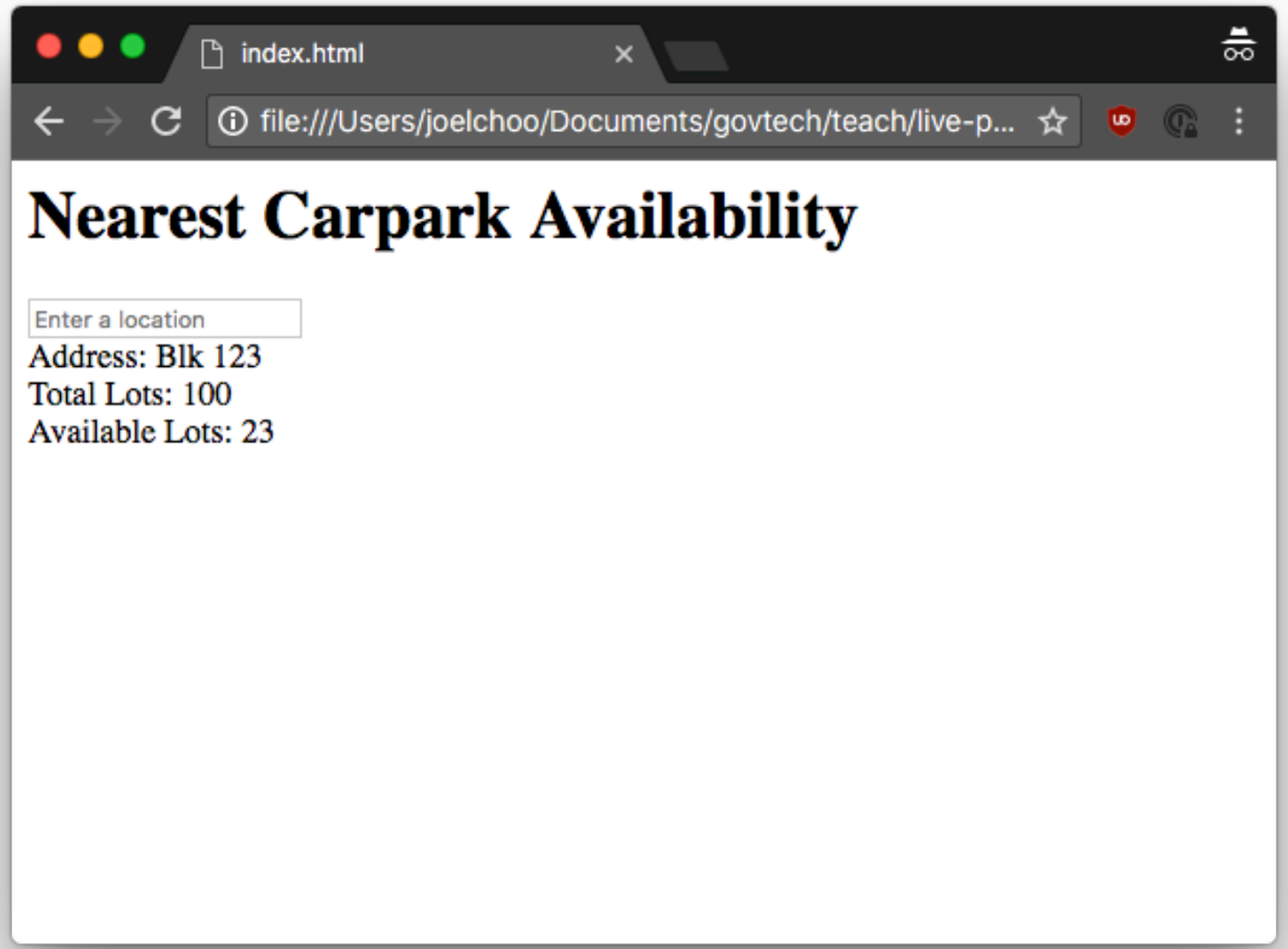
```
<body>
  <h1>Nearest Carpark Availability</h1>
  <input placeholder="Enter a location">

  <div>Hello!</div>
</body>
```

divs are containers for content

```
<body>
  <h1>Nearest Carpark Availability</h1>
  <input placeholder="Enter a location">

  <div>Address: Blk 123</div>
  <div>Total Lots: 456</div>
  <div>Available Lots: 78</div>
</body>
```

`divs` can be used to organise and arrange content

Result


index.html
file:///Users/joelchoo/Documents/govtech/teach/live-p...
Nearest Carpark Availability
Enter a location
Address: Blk 123
Total Lots: 100
Available Lots: 23

A short detour

Adding styles to your page!

```
<body style="background-color: lightskyblue">
    .
    .
    .
</body>
```

https://developer.mozilla.org/en-US/docs/Web/CSS/color_value

Step 1 - A quick recap

- HTML files in browser are webpages
- HTML tells the browser what to display
- HTML tags, attributes

What's next:

Adding interaction to our page!

Metaphor - the human body

Term	Metaphor	Function
HTML	Skeleton	Structure
CSS (styles)	Skin	Styling, formatting
Javascript	Brain	Behaviour/actions

```
<body>
  <h1>Nearest Carpark Availability</h1>
  <input placeholder="Enter a location">

  <div>Address: Blk 123</div>
  <div>Total Lots: 456</div>
  <div>Available Lots: 78</div>
</body>
```

```
<script> alert("Hello!") </script>
```

`script` content is javascript code that the browser will execute

```
<body>
  <h1>Nearest Carpark Availability</h1>
  <input placeholder="Enter a location">

  <div>Address: Blk 123</div>
  <div>Total Lots: 456</div>
  <div>Available Lots: 78</div>
</body>
```

```
<script src="carpark-logic.js"></script>
```

To make it simpler, we can put the javascript code into another file

`carpark-logic.js`

```
alert("Hello!")
```

Note: For subsequent slides, the slide title is the file you should be editing

What's next

- We need to be able to replace the address and parking lot information dynamically based on what the user searches
- We can do that by using Javascript to:

    1. Find the `div` for carpark address
    2. Change the text inside the `div`
    3. Repeat for parking lot information

- To find the correct `div` for Step 1, we need to give the `div` an ID!

index.html

```
<body>
  <h1>Nearest Carpark Availability</h1>
  <input placeholder="Enter a location">

  <div id="carparkAddress"></div>
  <div>Total Lots: 456</div>
  <div>Available Lots: 78</div>
</body>

<script src="carpark-logic.js"></script>
```

Refresh the page - the address should have disappeared

`carpark-logic.js`

```
document.getElementById("carparkAddress").innerText =
  "Address: Blk 123"
```

- `document.getElementById` gets the `carparkAddress` `div` in our HTML
- Changing `innerText` changes the text inside the `carparkAddress` `div`
- Refresh the page and the address should reappear
- Repeat for the parking lot information

index.html

```
<body>
  <h1>Nearest Carpark Availability</h1>
  <input placeholder="Enter a location">

  <div id="carparkAddress"></div>
  <div id="carparkTotalLots"></div>
  <div id="carparkAvailableLots"></div>
</body>

<script src="carpark-logic.js"></script>
```

Refresh the page - total and available lots should have disappeared

carpark-logic.js

```
document.getElementById("carparkAddress").innerText =
  "Address: Blk 123"
document.getElementById("carparkTotalLots").innerText =
  "Total Lots: 456"
document.getElementById("carparkAvailableLots").innerText =
  "Available Lots: 78"
```

Refresh the page - total and available lots should reappear

Introducing functions

- Functions are like recipes - they're a way to group some instructions together

```
function bakeCake() {
  preheatOven()
  mixIngredients()
  putInOven()
}
```

- `bakeCake` is the function name

Introducing functions

- Functions are like recipes - they're a way to group some instructions together

```
function bakeCake(flour, eggs) {
  preheatOven()
  mixIngredients()
  putInOven()
}
```

- `bakeCake` is the function name
- `flour` and `eggs` are the function inputs

carpark-logic.js

```
function addCarparkToPage() {
  document.getElementById("carparkAddress").innerText =
    "Address: Blk 123"
  document.getElementById("carparkTotalLots").innerText =
    "Total Lots: 456"
  document.getElementById("carparkAvailableLots").innerText =
    "Available Lots: 78"
}
```

- Put your existing code into a function (just add the first and last line)
- Function name is `addCarparkToPage`
- Refresh your page - the text should disappear

carpark-logic.js

```
function addCarparkToPage() {
  document.getElementById("carparkAddress").innerText =
    "Address: Blk 123"
  document.getElementById("carparkTotalLots").innerText =
    "Total Lots: 456"
  document.getElementById("carparkAvailableLots").innerText =
    "Available Lots: 78"
}
```

```
addCarparkToPage()
```

- Creating a function does not run the code inside yet
- We need to call the function to run the code inside
- Analogy - writing a recipe is different from making a recipe

carpark-logic.js

```
function addCarparkToPage(address, totalLots, availableLots) {
  document.getElementById("carparkAddress").innerText =
    "Address: " + address
  document.getElementById("carparkTotalLots").innerText =
    "Total Lots: " + totalLots
  document.getElementById("carparkAvailableLots").innerText =
    "Available Lots: " + availableLots
}

addCarparkToPage("Blk 123", "456", "78")
```

- Add function inputs so that we can easily change the address and parking lot information displayed

Recap

- Javascript adds interaction
- Javascript connects to HTML using `getElementById`
- A function is a bunch of code that's grouped together
- Calling a function runs the code in it

What's next?

Reacting to user input

index.html

```
<body>
  <h1>Nearest Carpark Availability</h1>
  <input id="locationInput" placeholder="Enter a location">

  <div id="carparkAddress"></div>
  <div id="carparkTotalLots"></div>
  <div id="carparkAvailableLots"></div>
</body>
```

```
<script src="carpark-logic.js"></script>
```

Give the input element an ID

carpark-logic.js

```
function addCarparkToPage(address, totalLots, availableLots) {
   .
   .
   .
}

addCarparkToPage("Blk 123", "456", "78")

document.getElementById("locationInput")
```

Retrieve the input element using its ID

carpark-logic.js

```
function addCarparkToPage(address, totalLots, availableLots) {
   .
   .
   .
}

addCarparkToPage("Blk 123", "456", "78")

var locationInput = document.getElementById("locationInput")
```

- `var` is short for variable
- We're saving the element into the `locationInput` variable
- Analogy - saving your word document into Document.docx

`carpark-logic.js`

```
var locationInput = document.getElementById("locationInput")

locationInput.addEventListener("keydown", grabLocation)
```

- React to user typing in the input box by adding an **event listener** (`addEventListener`)
- `grabLocation` is a function that will be called when the user types something (`keydown`)

carpark-logic.js

```
var locationInput = document.getElementById("locationInput")

function grabLocation(event) {
  addCarparkToPage("Blk 123", "456", "78")
}

locationInput.addEventListener("keydown", grabLocation)
```

Wrap the `addCarparkToPage` function call inside a new function `grabLocation`

carpark-logic.js

```
function grabLocation(event) {
  if (event.key === "Enter") {
    addCarparkToPage("Blk 123", "456", "78")
  }
}
```

We only want to care about when the `Enter` key is pressed

What's next

Now we need to get the location that we typed in.

We can do that in a similar way to how we injected the text.

Find the input element, and extract it's content.

carpark-logic.js

```
function grabLocation(event) {
   if (event.key === "Enter") {
      var location = locationInput.value
      addCarparkToPage(location, "456", "78")
   }
}
```

`locationInput.value` gets the text that has been typed into the input box

Recap

- React to user input by registering **event listeners**
- Only react to Enter key press
- Get the text that the user has typed in

What's next?

1. Grab user input location
2. Get X and Y coordinates of given location
3. Get carpark closest to X and Y coordinates

What is an API call?

- Way for apps to communicate (over the internet)
- An API call is like a phone call to a wise person who has the answers to your questions
    - **"What time is it now?"**
    - **"Where are all the carparks in Singapore?"**
    - **"What is the current price of Bitcoin?"**

How do we make an API call?

- We need the help of a request library
- A library is a set of functions that someone else has written
- Like using tools that a wise man created

Why do we use APIs/Libraries?

- Don't reinvent the wheel
  - Passport: For password authentication
  - OpenCV: For image recognition
- Keeps our app simple!
- Some information can only be provided by certain people/ organisations (e.g. price of Bitcoin)

Our API call

"What are the X and Y coordinates of this location?"

https://docs.onemap.sg/#onemap-rest-apis

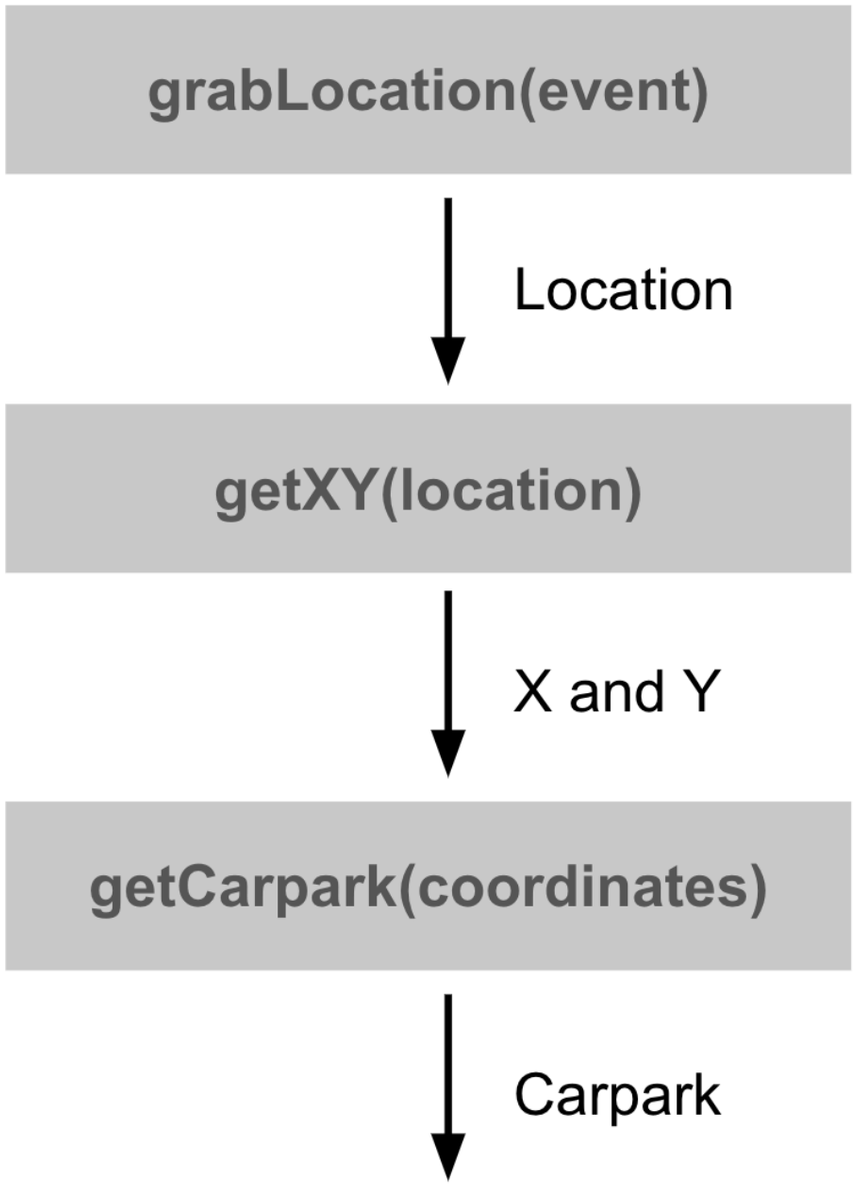
index.html

```
<head>
  <script src="https://unpkg.com/axios/dist/axios.min.js"></script>
</head>

<body>
  ...
</body>

<script src="carpark-logic.js"></script>
```

- Use the script tag to add the `axios` library to your code
- `head` loads our library before the rest of the page

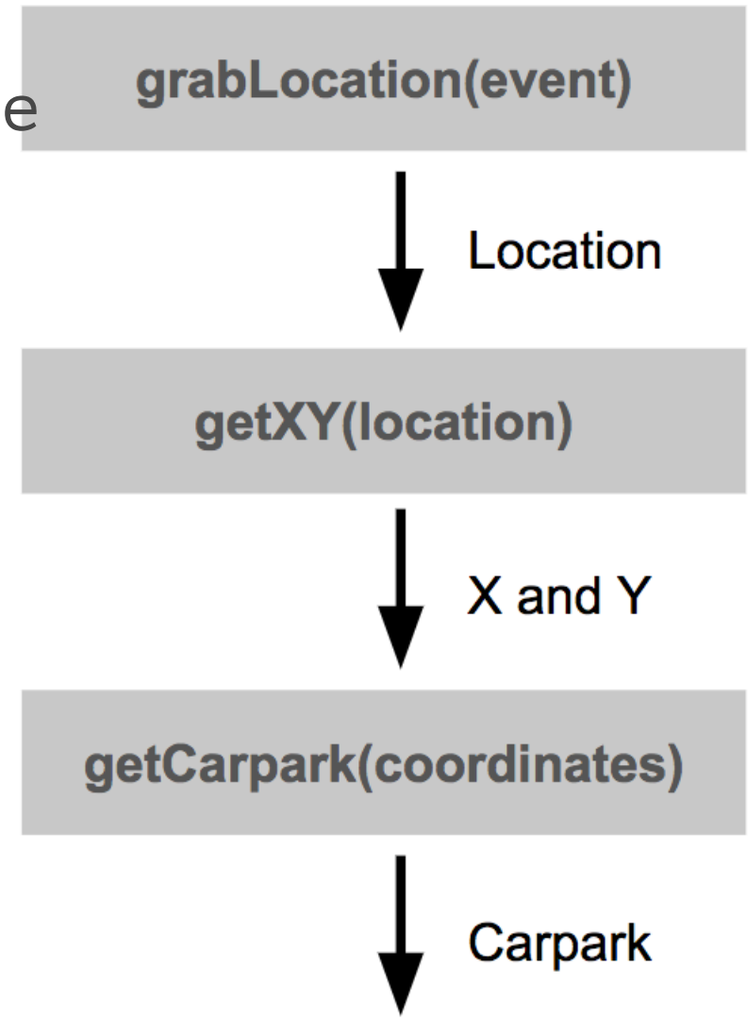
Let's sketch out our logic

1. Grab user input location
2. Get X and Y coordinates of given location
3. Get carpark closest to X and Y coordinates



carpark-logic.js

```
function grabLocation(event) {
  if (event.key === "Enter") {
    var location = locationInput.value
    getXY(location)
  }
}

function getXY(location) {
  var coordinates = {
    X: "1111",
    Y: "2222"
  }
  getCarpark(coordinates)
}

function getCarpark(coordinates) {
  addCarparkToPage(coordinates.X, coordinates.Y, "78")
}
```



carpark-logic.js

```
function getXY(location) {
  axios.get("https://developers.onemap.sg/commonapi/search", {
    params: {
      searchVal: location,
      returnGeom: "Y",
      getAddrDetails: "N"
    }
  }).then(response => {
    var coordinates = {
      X: "1111",
      Y: "2222"
    }
    getCarpark(coordinates)
  })
}
```

Make the API call

carpark-logic.js

```
function getXY(location) {
  axios.get("https://developers.onemap.sg/commonapi/search", {
    params: {
      searchVal: location,
      returnGeom: "Y",
      getAddrDetails: "N"
    }
  }).then(response => {
    console.log(response)
    var coordinates = {
      X: "1111",
      Y: "2222"
    }
    getCarpark(coordinates)
  })
}
```

What is returned by the API?

carpark-logic.js

```
function getXY(location) {
  axios.get("https://developers.onemap.sg/commonapi/search", {
    params: {
      searchVal: location,
      returnGeom: "Y",
      getAddrDetails: "N"
    }
  }).then(response => {
    console.log(response)
    var coordinates = response.data.results[0]
    getCarpark(coordinates)
  })
}
```

Extract the coordinates from the `response`

carpark-logic.js

```
function getXY(location) {
  axios.get("https://developers.onemap.sg/commonapi/search", {
    params: {
      searchVal: location,
      returnGeom: "Y",
      getAddrDetails: "N"
    }
  }).then(response => {
    var coordinates = response.data.results[0]
    console.log(coordinates)
    getCarpark(coordinates)
  })
}
```

`console.log` the X and Y coordinates for us to check that it is correct

What's next

- Now we want to get the nearest carpark to that X and Y value
- Slightly too complicated for now, so we can use a library that Joel wrote

index.html

```
<head>
  <script src="https://unpkg.com/axios/dist/axios.min.js"></script>
</head>

<body>
  ...
</body>

<script src="carpark-library.js"></script>
<script src="carpark-logic.js"></script>
```

Add `carpark-library.js` to your code

carpark-logic.js

```
function getCarpark(coordinates) {
  getNearestCarparkTo(coordinates.X, coordinates.Y).then(carpark => {
    addCarparkToPage(carpark.address, carpark.total_lots, carpark.lots_available)
  })
}
```

Call the `getNearestCarparkTo(...)` function from Joel's library

Recap

- Made API call to convert location to X and Y
- Use library function to get the nearest carpark

What's next?

- Content and logic is all done!
- Next - styling your site!

index.html

```
<head>
  <script src="https://unpkg.com/axios/dist/axios.min.js"></script>
  <link rel="stylesheet" type="text/css" href="carpark-style.css"></link>
</head>

<body>
  <h1>Nearest Carpark Availability</h1>
  <input id="locationInput" placeholder="Enter a location">

  <div id="carparkAddress"></div>
  <div id="carparkTotalLots"></div>
  <div id="carparkAvailableLots"></div>
</body>

<script src="carpark-library.js"></script>
<script src="carpark-logic.js"></script>
```

index.html

```
<head>
  <script src="https://unpkg.com/axios/dist/axios.min.js"></script>
  <link rel="stylesheet" type="text/css" href="carpark-style.css"></link>
</head>

<body>
  <h1>Nearest Carpark Availability</h1>
  <input id="locationInput" placeholder="Enter a location">

  <div id="carparkAddress"></div>
  <div id="carparkTotalLots"></div>
  <div id="carparkAvailableLots"></div>
</body>

<script src="carpark-library.js"></script>
<script src="carpark-logic.js"></script>
```

carpark-style.css

```
body {
  background-image: url("background-image.png");
  background-size: cover;
  background-position: center;

  text-align: center;
  color: #F2F2F2;
  margin-top: 10%;
}
```

Style the body - add background image

carpark-style.css

```
body {
  background-image: url("background-image.png");
  background-size: cover;
  background-position: center;

  text-align: center;
  color: #F2F2F2;
  margin-top: 10%;
}
```

Style the body - font and margin

carpark-style.css

```
body {
  /* ... */
}

h1 {
  font-size: 64px;
}
```

Style the h1

carpark-style.css

```
h1 {
  /* ... */
}

input {
  font-size: 20px;
  width: 70%;
  padding: 12px 20px;
  margin-bottom: 30px;
  border-radius: 4px;
}
```

Style the `input` (search box)

carpark-style.css

```
@import url('https://fonts.googleapis.com/css?family=Mandali');

body {
  background-image: url("background-image.png");
  background-size: cover;
  background-position: center;

  text-align: center;
  color: #F2F2F2;
  margin-top: 10%;
  font-family: "Mandali";
}

...
```

Import a font package from fonts.google.com

carpark-style.css

```
input {
  /* ... */
}

@media only screen
  and (max-device-width: 480px) {
  input {
    font-size: 32px;
    width: 95%;
  }
}
```

@media query: checking for screen size and then defining a specific style for the element

Recap

- Use CSS Selectors to style the `body`, `h1`, and `input`
- Use `@media` queries to set style for mobile devices

Recap

Term	Metaphor	Function
HTML	Skeleton	Structure
CSS (styles)	Skin	Styling, formatting
Javascript	Brain	Behaviour/actions

Next Step!

- Deploying your app to the internet
- Up to now, your website is just on your computer
- We will now deploy it to the internet so everyone can see and use it!

Deployment

1. Go to https://www.bitballoon.com/
2. Drag and drop the app folder on to the site.
3. Wait for it to process your data
4. Voila! Your site has been hosted!

Recap

1. What a website consists of
2. HTML + Javascript + CSS
3. API calls - communication over the internet
4. Deploying websites

That's all folks!

Any questions?

Bonus Material

Building your own simple API server

Next Step - Building your own API

- Build a simple backend to serve a simple API
- We'll replace my helper API with a real API

Create a new file called `backend.js`

`backend.js`

```
var express = require("express")

var app = express()

app.listen(3000, function () {
  console.log("Server running...")
})
```

- Open the command line (TAs can help!)
- Type `node backend.js` and hit enter
- You should see `Server running...`

backend.js

```
var express = require("express")

var app = express()

app.listen(3000, function () {
  console.log("Server running...")
})
```

- Importing `express`, a lightweight server

backend.js

```
var express = require("express")

var app = express()

app.listen(3000, function () {
  console.log("Server running...")
})
```

- Creating our server

backend.js

```
var express = require("express")

var app = express()

app.listen(3000, function () {
  console.log("Server running...")
})
```

- Running our server - start listening to requests

Right now your server is listening, but doesn't respond to any requests.

You've set up a phone, but you're not answering any calls.

Next step: make your server respond to incoming requests.

backend.js

```
var express = require("express")

var app = express()

app.get("/", function (req, res) {
  res.send("Hello")
})

app.listen(3000, function () {
  console.log("Server running...")
})
```

- Listen to "/"
- When "/" is requested, send Hello as the response

backend.js

```
var express = require("express")
var getNearestCarparkTo = require("./carpark-library.js")

var app = express()

app.get("/", function (req, res) {
  getNearestCarparkTo(req.query.x, req.query.y).then(carpark => {
    res.set("Access-Control-Allow-Origin", "*")
    res.send(carpark)
  })
})
```

- Send the x and y values to the helper API
- Helper API responds with the nearest carpark
- Take the carpark and send it back to the requester

backend.js

```
var express = require("express")
var getNearestCarparkTo = require("./carpark-library.js")

var app = express()

app.get("/", function (req, res) {
  getNearestCarparkTo(req.query.x, req.query.y).then(carpark => {
    res.set("Access-Control-Allow-Origin", "*")
    res.send(carpark)
  })
})

app.listen(3000, function () {
  console.log("Server running...")
})
```

index.html

```
<head>
  <script src="https://unpkg.com/axios/dist/axios.min.js"></script>
  <link rel="stylesheet" type="text/css" href="carpark-style.css"></link>
</head>

<body>
  <h1>Nearest Carpark Availability</h1>
  <input id="locationInput" placeholder="Enter a location">

  <div id="carparkAddress"></div>
  <div id="carparkTotalLots"></div>
  <div id="carparkAvailableLots"></div>
</body>

<script src="carpark-library.js"></script>
<script src="carpark-logic.js"></script>
```

Delete the highlighted line

index.html

```
<head>
  <script src="https://unpkg.com/axios/dist/axios.min.js"></script>
  <link rel="stylesheet" type="text/css" href="carpark-style.css"></link>
</head>

<body>
  <h1>Nearest Carpark Availability</h1>
  <input id="locationInput" placeholder="Enter a location">

  <div id="carparkAddress"></div>
  <div id="carparkTotalLots"></div>
  <div id="carparkAvailableLots"></div>
</body>

<script src="carpark-logic.js"></script>
```

Refresh the page - the search should no longer be working

carpark-logic.js

```
function getCarpark(coordinates) {
  getNearestCarparkTo(coordinates.X, coordinates.Y).then(carpark => {
    addCarparkToPage(carpark.address, carpark.total_lots, carpark.lots_available)
  })
}
```

Delete the highlighted lines

carpark-logic.js

```
function getCarpark(coordinates) {
  var url = "http://localhost:3000"
  axios.get(url, {
    params: {
      x: coordinates.X,
      y: coordinates.Y
    }
  }).then(carpark => {
    var carpark = response.data
    addCarparkToPage(carpark.address, carpark.total_lots, carpark.lots_available)
  })
}
```

Set up `carpark-logic` to use our backend API