Advanced JavaScript

Assoc. Prof. Dr. Kanda Runapongsa Saikaew
(krunapon@kku.ac.th)
Dept. of Computer Engineering
Khon Kaen University

Agenda

- Introduction to events
- Introducing JavaScript objects
- Object-oriented JavaScript for beginners
- Object prototypes
- Inheritance in JavaScript
- Working with JSON data

What are events?

- Events are actions or occurrences that happen in the system you are programming, which the system tells you about so you can respond to them in some way if desired
- In the case of the Web, events are fired inside the browser window, and tend to be attached to a specific item that resides in it

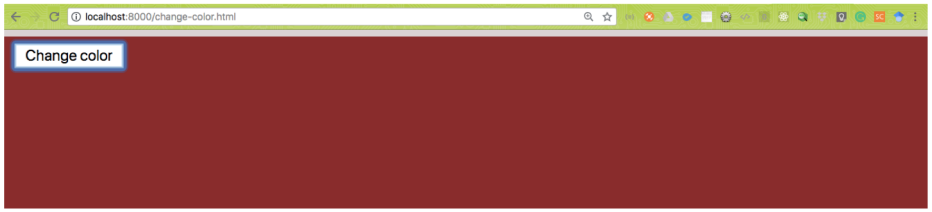
Sample events

- The user clicking the mouse over a certain element or hovering the cursor over a certain element.
- The user pressing a key on the keyboard.
- The user resizing or closing the browser window.
- A web page finishing loading.
- A form being submitted.
- A video being played, or paused, or finishing play.
- An error occurring.

Event handler

- Each available event has an **event handler**, which is a block of code (usually a user-defined JavaScript function) that will be run when the event fires
- When such a block of code is defined to be run in response to an event firing, we say we are **registering an event handler**
- Note that event handlers are sometimes called **event listeners**

Exercise1: Change Color




```
var btn = document.querySelector('button');
function random(number) {
        ..........................................
}
btn.onclick = function() {
            var rndCol = 'rgb(' + random(255) + ','
+ random(255) + ',' + random(255) + ')';
            document.body.style.backgroundColor =
rndCol;
  }
```

Ways of using web events

```
var btn = document.querySelector('button');

btn.onclick = function() {
  var rndCol = 'rgb(' + random(255) + ',' + random(255) + ',' + random(255) + ')';
  document.body.style.backgroundColor = rndCol;
}
```

```
var btn = document.querySelector('button');

function bgChange() {
  var rndCol = 'rgb(' + random(255) + ',' + random(255) + ',' + random(255) + ')';
  document.body.style.backgroundColor = rndCol;
}

btn.onclick = bgChange;
```

Other types of events (1/2)

- btn.onfocus and btn.onblur — The color will change when the button is focused and unfocused
- btn.ondblclick — The color will change only when it is double-clicked
- btn.onmouseover and btn.onmouseout — The color will change when the mouse pointer is moved so it begins hovering over the button, or when it stops hovering over the button and moves off of it

Other types of events (2/2)

- window.onkeypress, window.onkeydown, window.onkeyup
  - The color will change when a key is pressed on the keyboard
  - Keypress refers to a general press (button down and then up)
  - Keydown and keyup refer to just the key down and key up parts of the keystroke, respectively.

Inline event handlers

```
<button onclick="bgChange()">Press me</button>
```

```
function bgChange() {
  var rndCol = 'rgb(' + random(255) + ',' + random(255) + ',' + random(255) + ')';
  document.body.style.backgroundColor = rndCol;
}
```

```
var buttons = document.querySelectorAll('button');

for (var i = 0; i < buttons.length; i++) {
  buttons[i].onclick = bgChange;
}
```

addEventListener()

- The newest type of event mechanism is defined in the Document Object Model (DOM) Level 2 Events Specification - addEventListener()

```
1  var btn = document.querySelector('button');
2
3  function bgChange() {
4    var rndCol = 'rgb(' + random(255) + ',' + random(255) + ',' + random(255) + ')';
5    document.body.style.backgroundColor = rndCol;
6  }
7
8  btn.addEventListener('click', bgChange);
```

addEventListener() and removeEventListener()

- Anonymous function

```
1  btn.addEventListener('click', function() {
2    var rndCol = 'rgb(' + random(255) + ',' + random(255) + ',' + random(255) + ')';
3    document.body.style.backgroundColor = rndCol;
4  });
```

- Remove event listener

```
1  btn.removeEventListener('click', bgChange);
```

Register multiple handlers

- Approach 1

```
1 | myElement.onclick = functionA;
2 | myElement.onclick = functionB;
```

- Approach 2

```
1 | myElement.addEventListener('click', functionA);
2 | myElement.addEventListener('click', functionB);
```

What mechanism should I use?

- Event handler properties have less power and options, but better cross-browser compatibility (being supported as far back as Internet Explorer 8)
- DOM Level 2 Events (addEventListener(), etc.) are more powerful, but can also become more complex and are less well supported (supported as far back as Internet Explorer 9)
  - You should also experiment with these, and aim to use them where possible.

Event objects

- Sometimes inside an event handler function, you might see a parameter specified with a name such as event, evt, or simply e
- This is called the event object

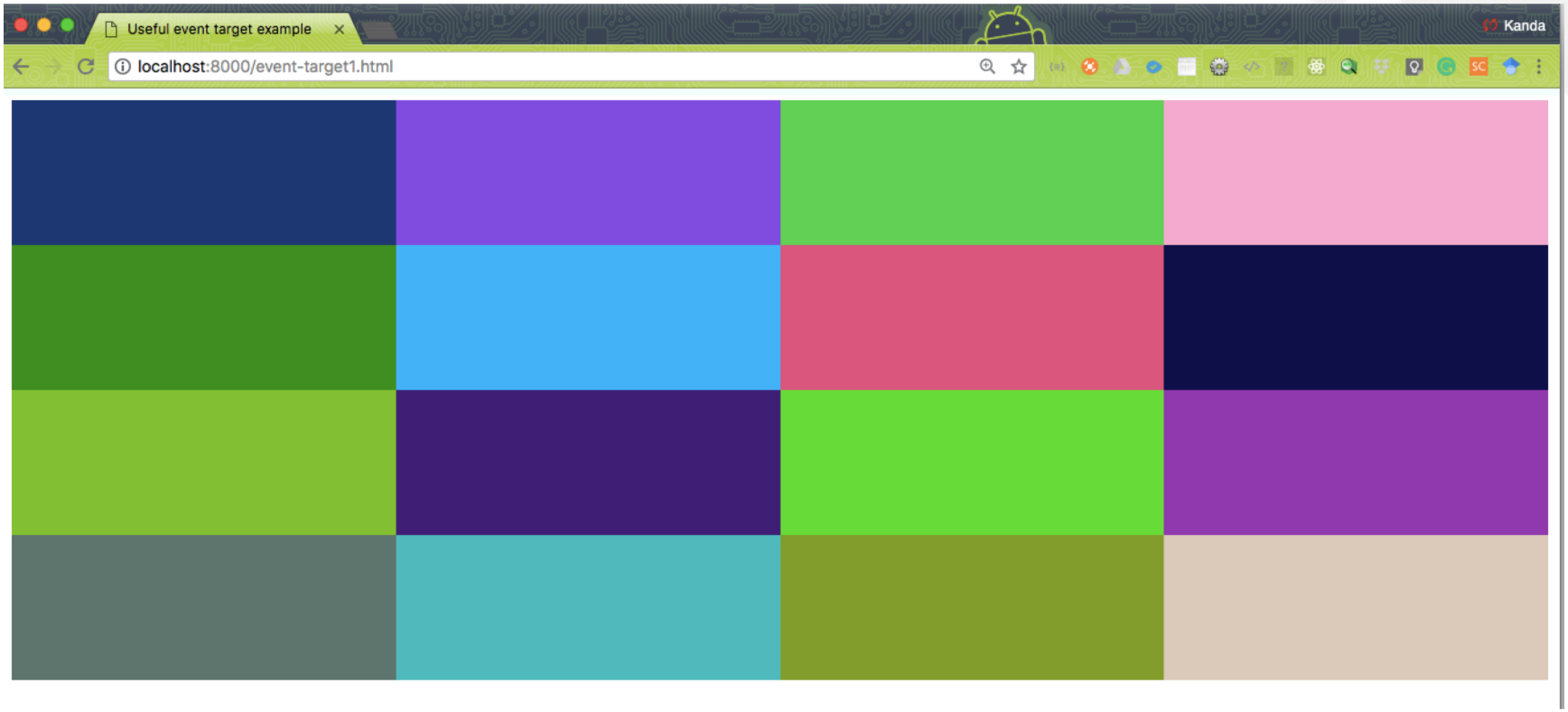
```
1  function bgChange(e) {
2    var rndCol = 'rgb(' + random(255) + ',' + random(255) + ',' + random(255) + ')';
3    e.target.style.backgroundColor = rndCol;
4    console.log(e);
5  }
6
7  btn.addEventListener('click', bgChange);
```

e.target

- The target property of the event object is always a reference to the element that the event has just occurred upon
- So in this example, we are setting a random background color on the button, not the page

```
1  function bgChange(e) {
2    var rndCol = 'rgb(' + random(255) + ',' + random(255) + ',' + random(255) + ')';
3    e.target.style.backgroundColor = rndCol;
4    console.log(e);
5  }
6
7  btn.addEventListener('click', bgChange);
```

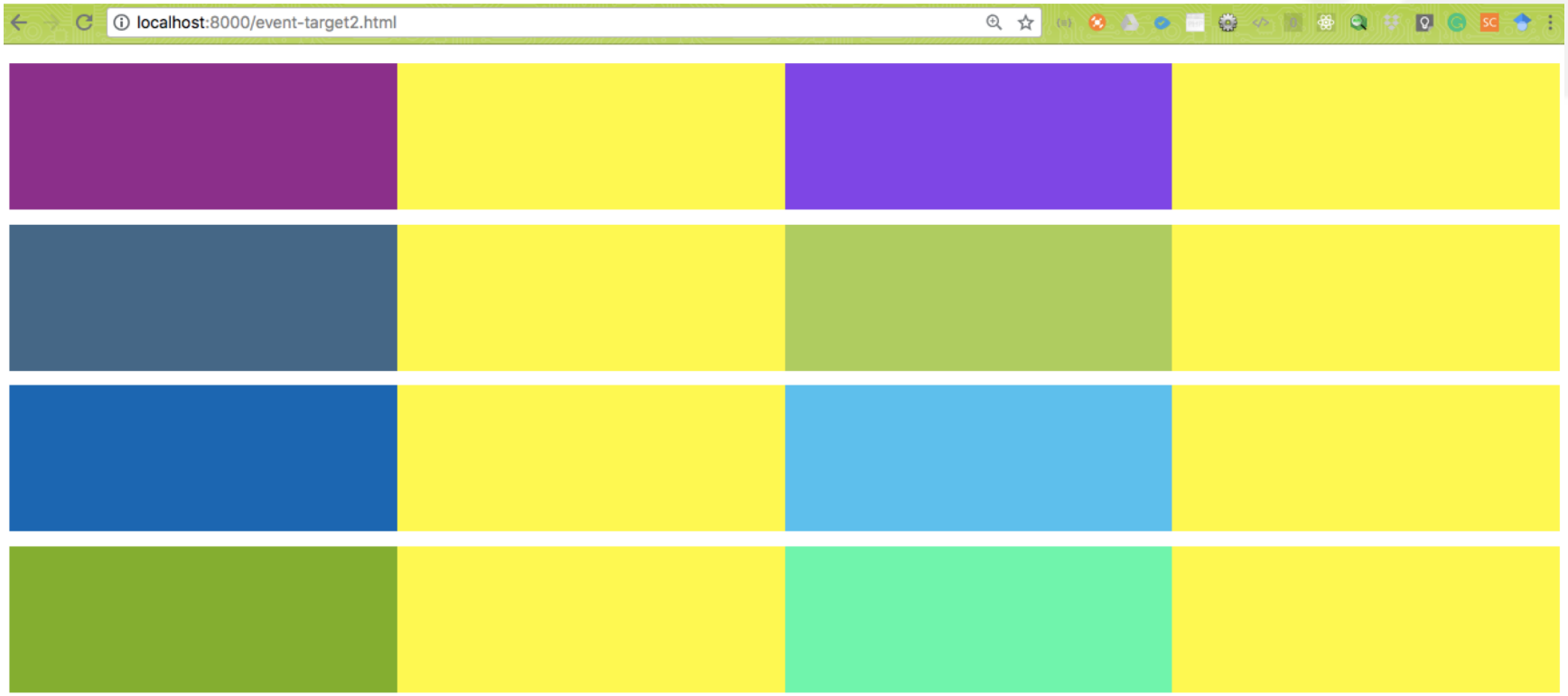
Sample usage of e.target: change-color.html



Code of change-color.html

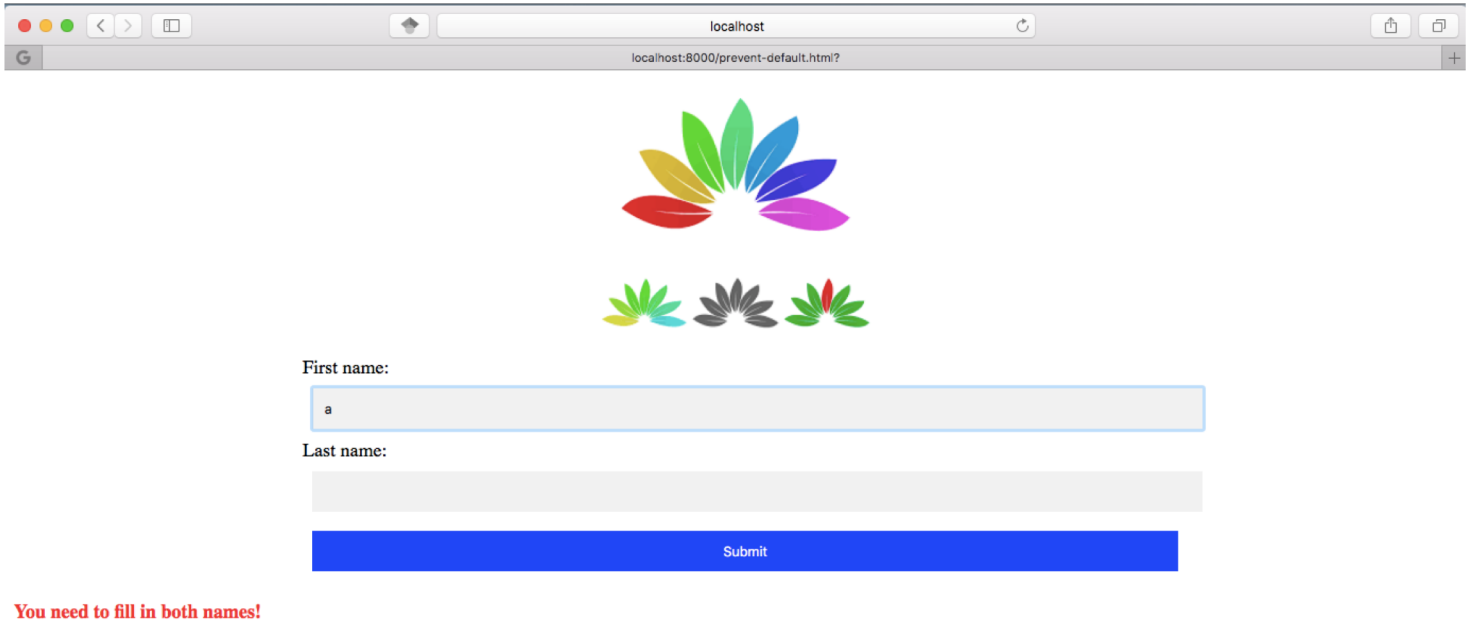
```
 6     <style>
 7       div {
 8         background-color: red;
 9         height: 100px;
10         width: 25%;
11         float: left;
12       }
13     </style>
14   </head>
15   <body>
16     <script>
17       for(var i = 1; i <= 16; i++) {
18         var myDiv = document.createElement('div');
19         document.body.appendChild(myDiv);
20       }
21       function random(number) {
22         return Math.floor(Math.random()*number);
23       }
24       function bgChange() {
25         var rndCol = 'rgb(' + random(255) + ',' + random(255) + ',' + random(255) +
           ')';
26         return rndCol;
27       }
28       var divs = document.querySelectorAll('div');
29       for(var i = 0; i < divs.length; i++) {
30         divs[i].onclick = function(e) {
31           e.target.style.backgroundColor = bgChange();
32         }
33       }
34     </script>
```

Exercise 2



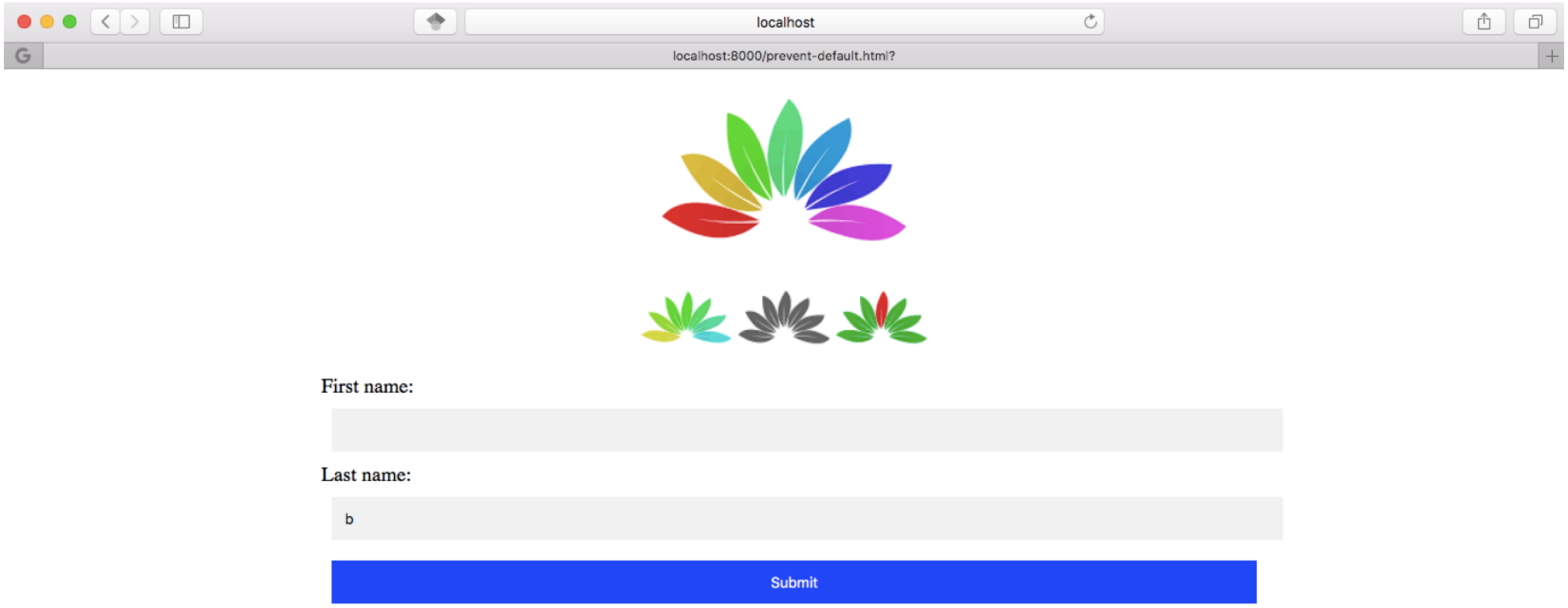
Exercise 3 (1/2)

- When the user presses enter on only the first name field, there is an error message displaying ”You need to fill in both names!”



Exercise 3 (2/2)

- When the user presses enter on only the last name field, there is an error message displaying ”You need to fill in both names!”



Preventing default behavior

- Sometimes, you'll come across a situation where you want to stop an event doing what it does by default
- The trouble comes when the user has not submitted the data correctly — as a developer, you'll want to stop the submission to the server and give them an error message telling them what's wrong and what needs to be done to put things right.

A Simple HTML Form

```
<form>
  <div>
    <label for="fname">First name: </label>
    <input id="fname" type="text">
  </div>
  <div>
    <label for="lname">Last name: </label>
    <input id="lname" type="text">
  </div>
  <div>
     <input id="submit" type="submit">
  </div>
</form>
<p></p>
```

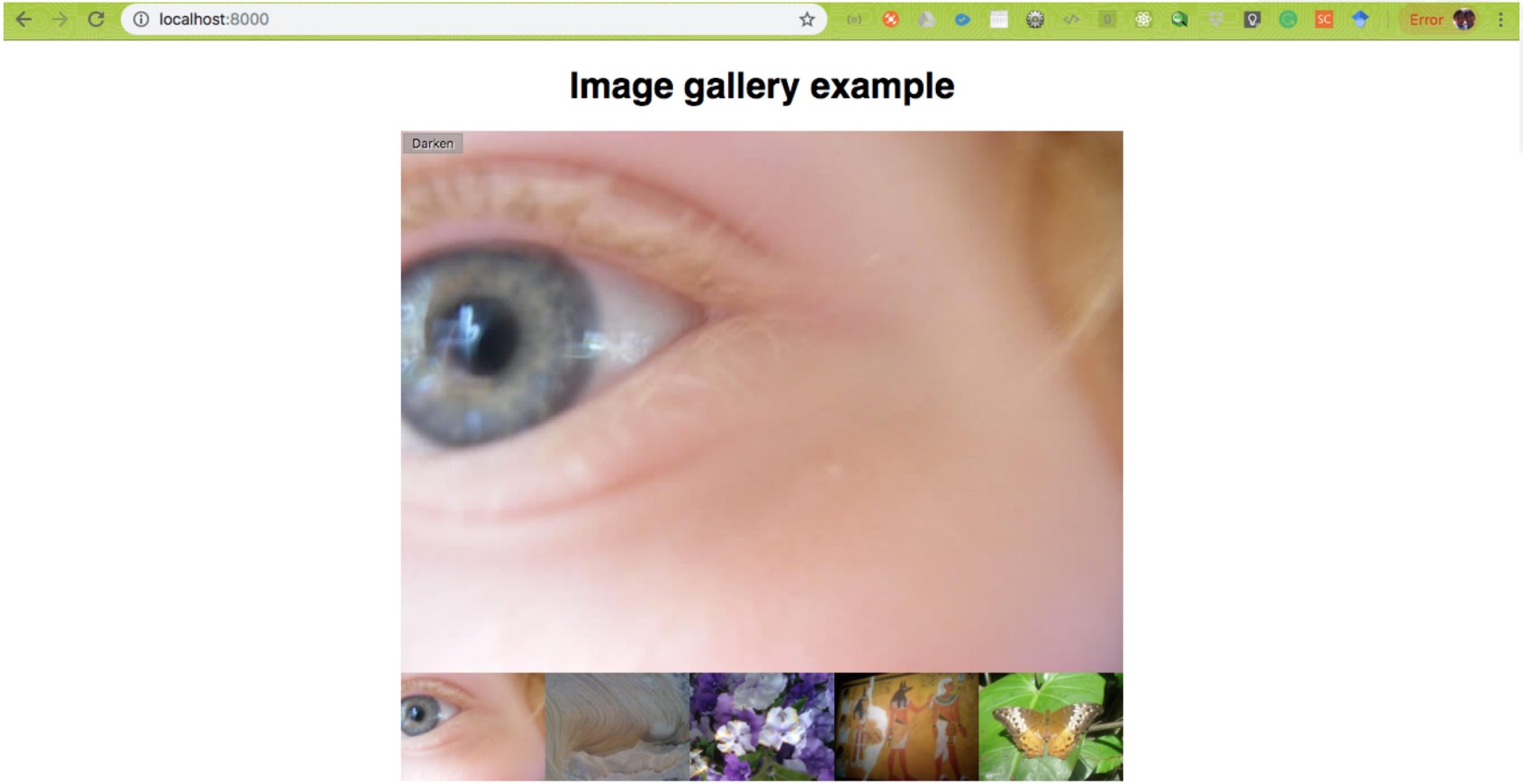
Function preventDefault()

- We can implement a very simple check inside an onsubmit event handler (the submit event is fired on a form when it is submitted) that tests whether the text fields are empty
- If they are, we call the preventDefault() function on the event object — which stops the form submission — and then display an error message in the paragraph below our form to tell the user what's wrong

Calling preventDefault()

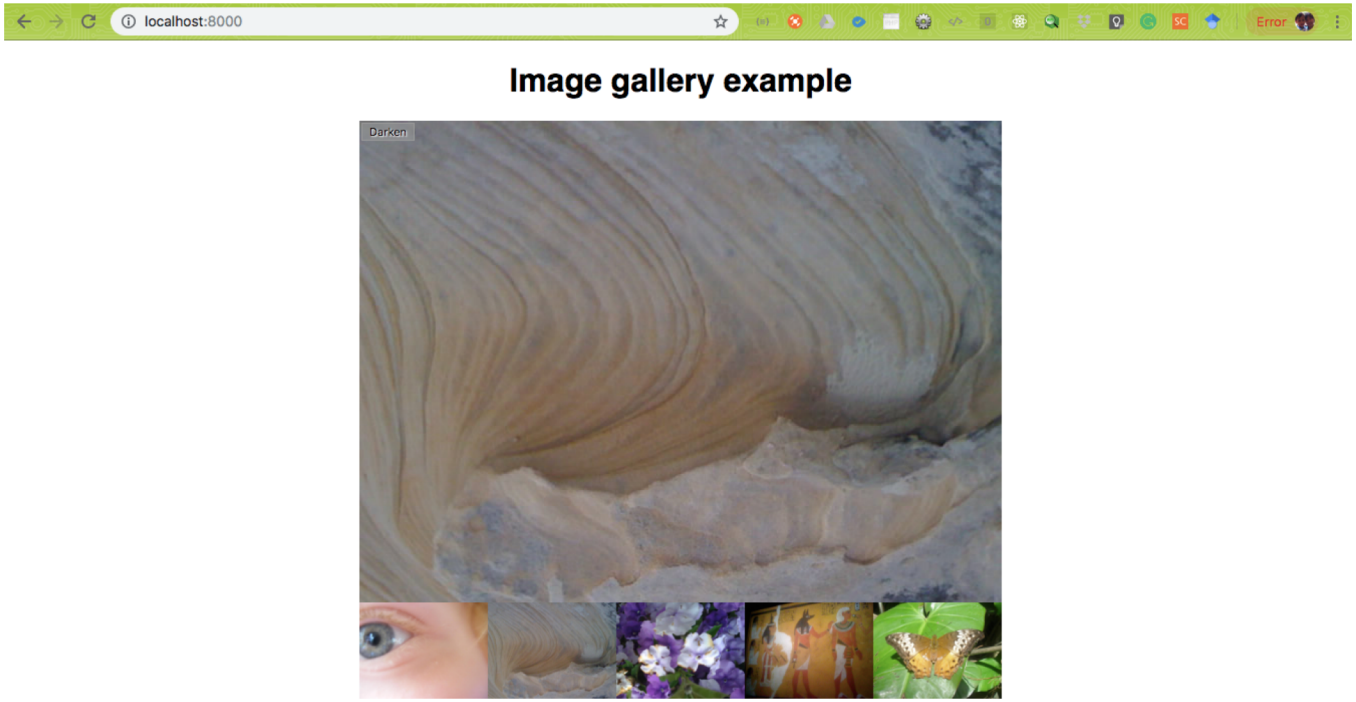
```
1  var form = document.querySelector('form');
2  var fname = document.getElementById('fname');
3  var lname = document.getElementById('lname');
4  var submit = document.getElementById('submit');
5  var para = document.querySelector('p');
6
7  form.onsubmit = function(e) {
8    if (fname.value === '' || lname.value === '') {
9      e.preventDefault();
10     para.textContent = 'You need to fill in both names!';
11   }
12 }
```

Exercise 4: Image Gallery (1/3)



Exercise 4: Image Gallery (2/3)

- When clicking the image below, the clicked image is displayed at the center



Exercise 4: Image Gallery (3/3)

Getting started

- To get this assessment started, you should go and grab the ZIP file for the example and unzip it somewhere on your computer

```
<h1>Image gallery example</h1>

<div class="full-img">
  <img class="displayed-img" src="images/pic1.jpg">
  <div class="overlay"></div>
  <button class="dark">Darken</button>
</div>

<div class="thumb-bar">

</div>
```

Initial JavaScript for Image Gallery

```
1  var displayedImage = document.querySelector('.displayed-img');
2  var thumbBar = document.querySelector('.thumb-bar');
3
4  btn = document.querySelector('button');
5  var overlay = document.querySelector('.overlay');
6
7  /* Looping through images */
8
9    var newImage = document.createElement('img');
10   newImage.setAttribute('src', xxx);
11   thumbBar.appendChild(newImage);
12
13 /* Wiring up the Darken/Lighten button */
14
15
```

Agenda

- Introduction to events
- **Introducing JavaScript objects**
- Object-oriented JavaScript for beginners
- Object prototypes
- Inheritance in JavaScript
- Working with JSON data

Object basics

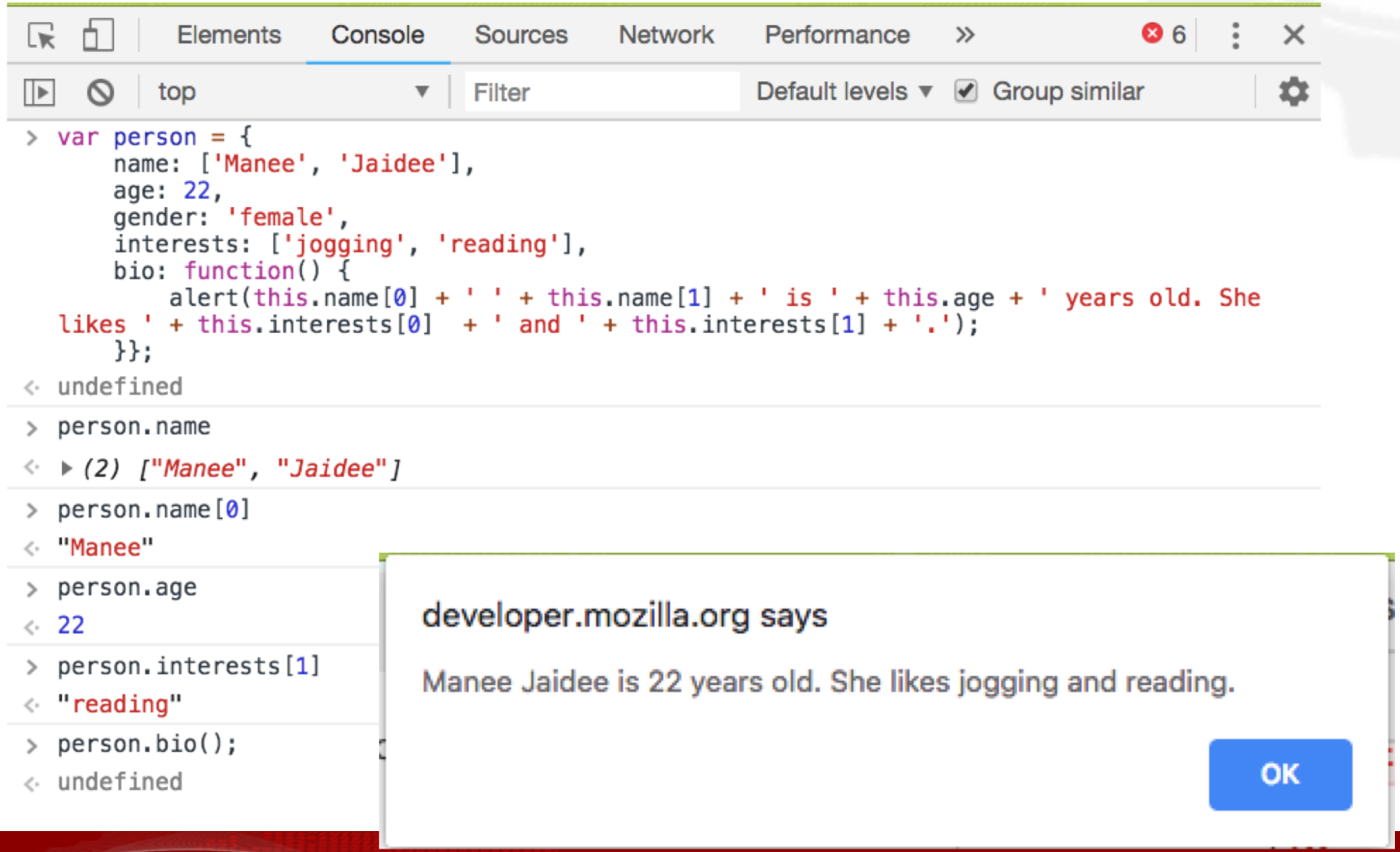
- An object is a collection of related data and/or functionality (which usually consists of several variables and functions)
- The value of an object can be pretty much anything
  - Strings, numbers, arrays, or functions

Sample objects

- var person = {

```
    name: ['Bob', 'Smith'],
    age: 32,
    gender: 'male',
    interests: ['music', 'skiing'],
    greeting: function() {
    alert('Hi! I\'m ' + this.name[0] + '.');
  }
};
```

Sample objects on console



Dot notation

- You accessed the object's properties and methods using **dot notation**
- The object name (person) acts as the **namespace**
  - It must be entered first to access anything **encapsulated** inside the object

```
1 | person.age
2 | person.interests[1]
3 | person.bio()
```

Bracket notation

- There is another way to access object properties – using bracket notation

```
1 person.age
2 person.name.first
```

```
1 person['age']
2 person['name']['first']
```

- It is no wonder that objects are sometimes called **associative arrays** — they map strings to values in the same way that arrays map numbers to values.

Setting object members

```
> var person = {
    name: ['Bob', 'Smith'],
    age: 32,
    gender: 'male',
    interests: ['music', 'skiing'],
    bio: function() {
      alert(this.name[0] + ' ' +
  this.name[1] + ' is ' + this.age + '
  years old. He likes ' + this.interests[0]
  + ' and ' + this.interests[1] + '.');
    },
    greeting: function() {
      alert('Hi! I\'m ' + this.name[0] +
  '.');
    }
  };
<- undefined
> person['eyes'] = 'hazel';
<- "hazel"
> person.eyes
<- "hazel"
> person.farewell = function() {
  alert("Good bye");}
<- f () { alert("Good bye");}
> person.farewell
<- f () { alert("Good bye");}
>
```

```
> var myDataName = 'height';
  var myDataValue = '1.75m';
  person[myDataName] = myDataValue;
<- "1.75m"
> person[myDataName]
<- "1.75m"
> person.height
<- "1.75m"
```

What is “this”?

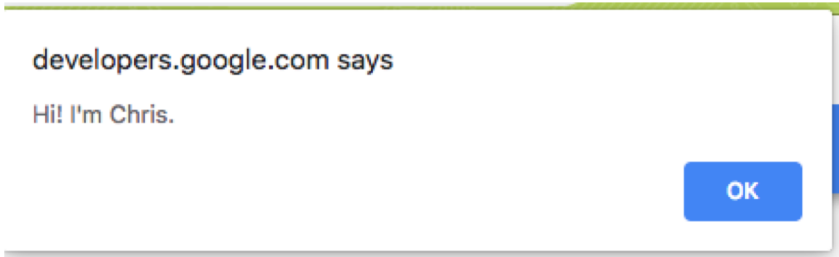
```
1 | greeting: function() {
2 |   alert('Hi! I\'m ' + this.name.first + '.');
3 | }
```

The **this** keyword refers to the current object the code is being written inside
— so in this case this is equivalent to person.

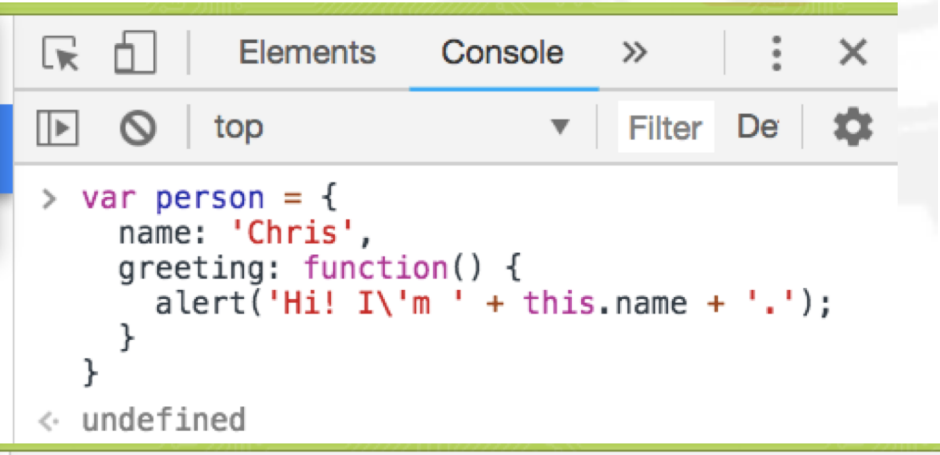
Example of using this

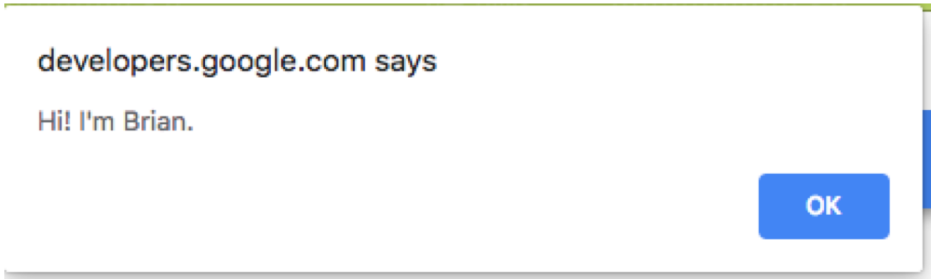
```
1  var person1 = {
2    name: 'Chris',
3    greeting: function() {
4      alert('Hi! I\'m ' + this.name + '.');
5    }
6  }
7
8  var person2 = {
9    name: 'Brian',
10   greeting: function() {
11     alert('Hi! I\'m ' + this.name + '.');
12   }
13 }
```

Better code for variable person

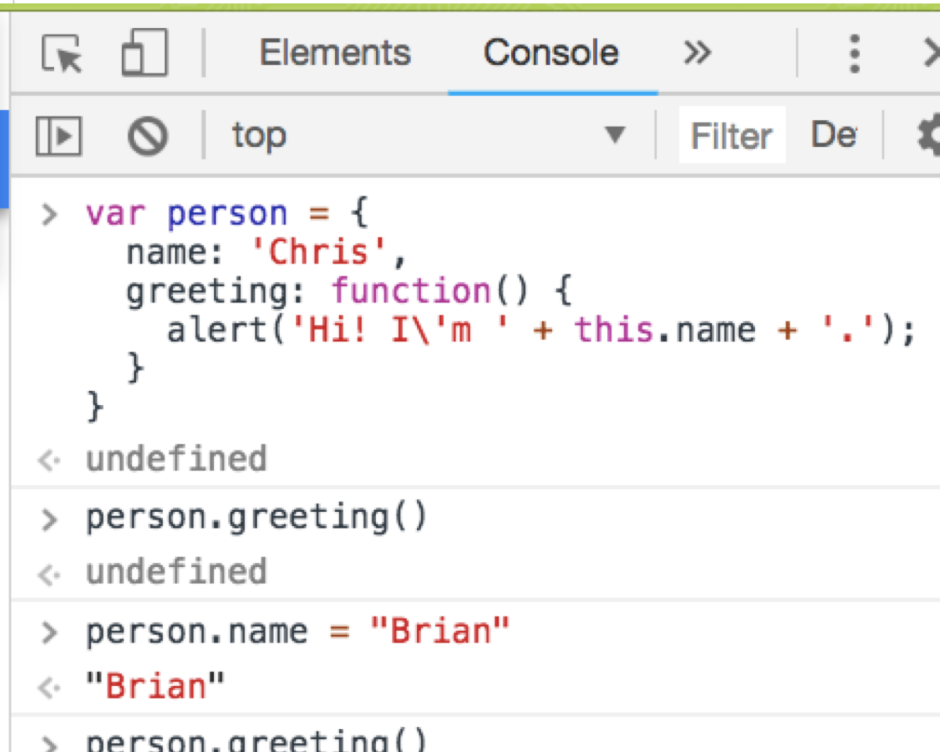


DevTools Console, stack redundant messages or display
clear or persist output or save it to a file, filter output, and
ole settings.





DevTools Console, stack redundant messages or display
, clear or persist output or save it to a file, filter output, and
ole settings.



as a dedicated panel or as a drawer next to any other panel.

nessages, or display them on their own lines.

Question

- What is the output of this code?

```
> var person2 = {
    name: 'Chris',
    greeting: function() {
      alert('Hi! I\'m ' + name + '.');
    }
  }
<· undefined
> person2.greeting()
<· undefined
>
```

Agenda

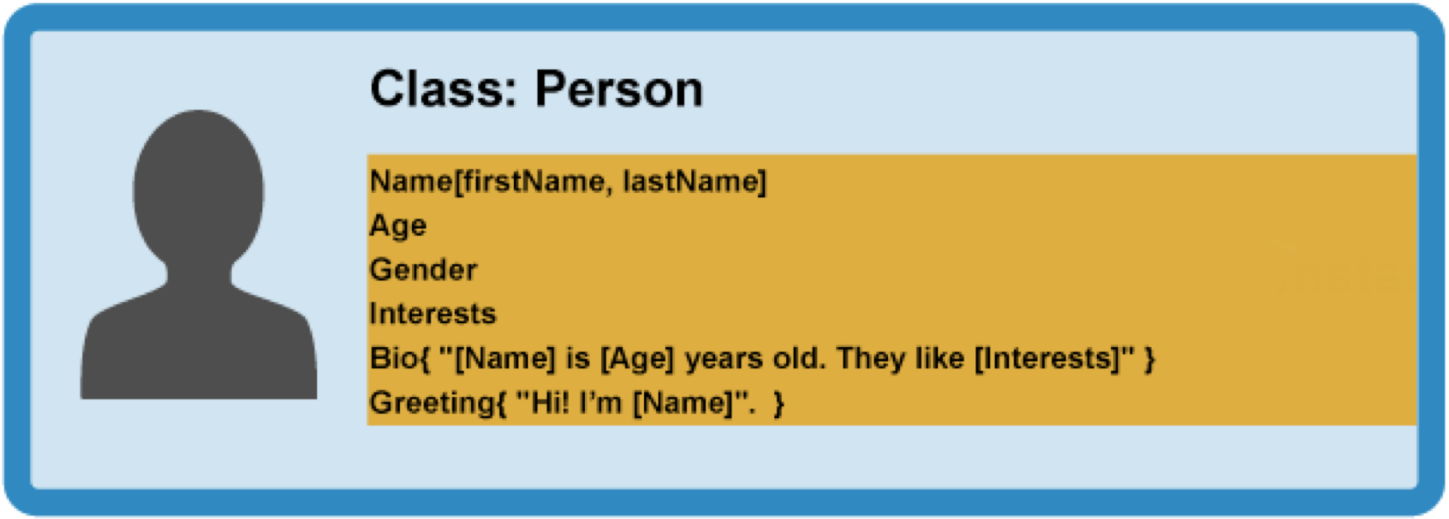
- Introduction to events
- Introducing JavaScript objects
- **Object-oriented JavaScript for beginners**
- Object prototypes
- Inheritance in JavaScript
- Working with JSON data
- Object building practice

Object-oriented programming – the basics

- The basic idea of OOP is that we use objects to model real world things that we want to represent inside our programs
- Objects can contain related data and code, which represent information about the thing you are trying to model and functionality or behavior that you want it to have

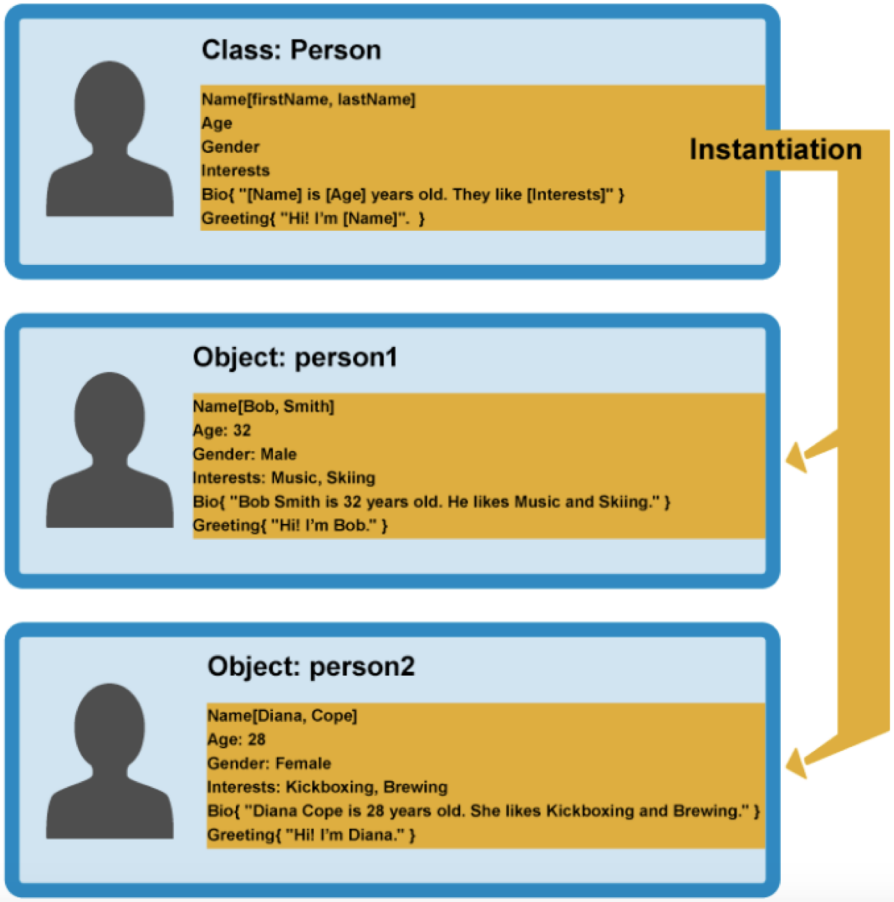
Defining an object template

- Abstraction – creating a simple model of a more complex thing, which represents its most important aspects in a way that is easy to work with for our programs' purposes



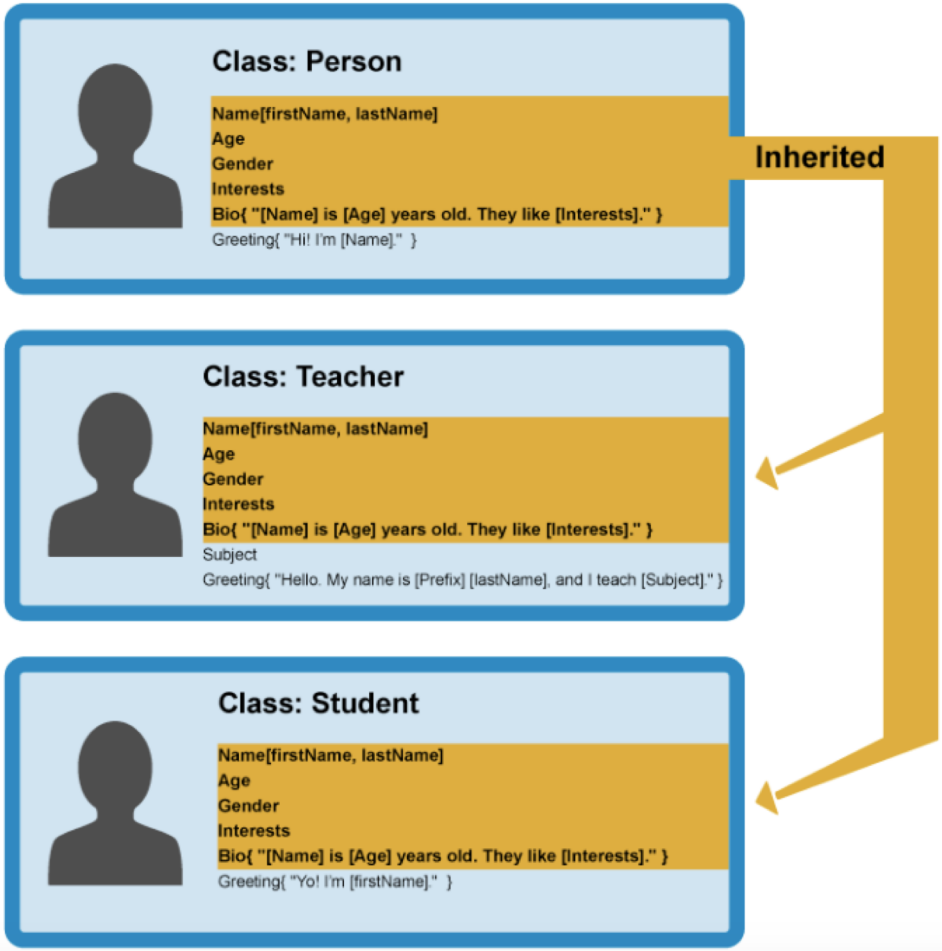
Creating actual objects

- This process of creating an object instance from a class is called **instantiation** — the object instance is **instantiated** from the class.



Specialist classes

- We can create new classes based on other classes — these new **child classes** can be made to **inherit** the data and code features of their **parent class**



Constructors and object instances

- JavaScript uses special functions called **constructor functions** to define objects and their features

```
> function createNewPerson(name) {
      var obj  = {};
      obj.name = name;
      obj.greeting = function() {
          alert('Hi! I\'m ' + this.name +
  '.');}
      return obj;
  }
<· undefined
> var salva = createNewPerson('Salva');
<· undefined
> salva.name;
<· "Salva"
> salva.greeting();
>
```

Constructors to create some objects

```
> function Person(name) {
      this.name = name;
      this.greeting = function() {
          alert('Hi! I\'m ' + this.name +
  '.');
      };
  }
<· undefined
> var person1 = new Person('Bob');
<· undefined
> var person2 = new Person('Sarah');
<· undefined
> person1.name
<· "Bob"
> person1.greeting()
<· undefined
> person2.name
<· "Sarah"
> person2.greeting()
<· undefined
```

The new keyword

- The new keyword is used to tell the browser we want to create a new object instance, followed by the function name with its required parameters contained in parentheses, and the result is stored in a variable

New objects have been created

```
function Person(name) {
  this.name = name;
  this.greeting = function() {
    alert('Hi! I\'m ' + this.name + '.');
  };
}
```

After the new objects have been created, the `person1` and `person2` variables contain the following objects:

```
{
  name: 'Bob',
  greeting: function() {
    alert('Hi! I\'m ' + this.name + '.');
  }
}

{
  name: 'Sarah',
  greeting: function() {
    alert('Hi! I\'m ' + this.name + '.');
  }
}
```

The Object() constructor

- We can use the Object() constructor to create a new object
  - generic objects have a constructor, which generates an empty object

```
> var person1 = new Object();
<- undefined
> person1.name = 'Chris';
<- "Chris"
> person1['age'] = 38;
<- 38
> person1.greeting = function() {
      alert('Hi! I\'m ' + this.name + '.');
  };
<- f () {
      alert('Hi! I\'m ' + this.name + '.');
  }
> person1.greeting()
<- undefined
> var person2 = new Object({
    name: 'Chris',
    age: 38,
    greeting: function() {
      alert('Hi! I\'m ' + this.name + '.');
    }
  });
<- undefined
> person2.greeting()
<- undefined
```

Using the create() method

- Constructors can help you give your code order—you can create constructors in one place, then create instances as needed, and it is clear where they came from

```
> person1
<· ▶ {name: "Chris", age: 38, greeting: f}
> var person3 = Object.create(person1);
<· undefined
> person3.name
<· "Chris"
> person3.greeting()
<· undefined
>
```

Agenda

- Introduction to events
- Introducing JavaScript objects
- Object-oriented JavaScript for beginners
- **Object prototypes**
- Inheritance in JavaScript
- Working with JSON data
- Object building practice

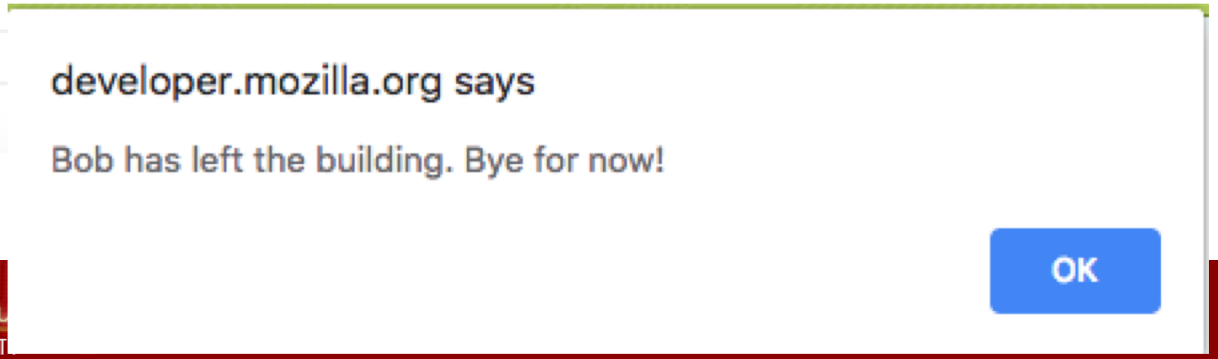
A prototype-based language

- JavaScript is often described as a **prototype-based language**
  - **E**ach object has a **prototype object**, which acts as a template object that it inherits methods and properties from
- An object's prototype object may also have a prototype object, which it inherits methods and properties from, and so on

The prototype property

```
> function Person(first, last, age, gender,
  interests) {

    // property and method definitions
    this.first = first;
    this.last = last;
    this.age = age;
    this.gender = gender;
    this.interests = interests;
  }
<· undefined
> var person1 = new Person('Bob', 'Smith',
  32, 'male', ['music', 'skiing']);
<· undefined
> Person.prototype.farewell = function() {
    alert(this.first + ' has left the
  building. Bye for now!');
  };
<· f () {
    alert(this.first + ' has left the
  building. Bye for now!');
  }
> person1.farewell();
>
```



Agenda

- Introduction to events
- Introducing JavaScript objects
- Object-oriented JavaScript for beginners
- Object prototypes
- **Inheritance in JavaScript**
- Working with JSON data
- Object building practice

Getting started

```
> function Person(first, last, age, gender,
  interests) {
    this.name = {
      first,
      last
    };
    this.age = age;
    this.gender = gender;
    this.interests = interests;
  };
<· undefined
> Person.prototype.greeting = function() {
    alert('Hi! I\'m ' + this.name.first +
  '.');
  };
<· f () {
    alert('Hi! I\'m ' + this.name.first +
  '.');
  }
>
```

Person ← Teacher

- We want to create a Teacher class, which inherits all the members from person, but also includes
  - A new property, subject – this will contain the subject the teacher teaches
  - An updated greeting() method, which sounds a bit more formal than the standard greeting() method – more suitable for a teacher addressing some students at school

Defining a Teacher() constructor function (using inheritance)

```
1  function Teacher(first, last, age, gender, interests, subject) {
2    Person.call(this, first, last, age, gender, interests);
3
4    this.subject = subject;
5  }
```

- The call() function basically allows you to call a function defined somewhere else, but in the current context
- The last line inside the constructor simply defines the new subject property that teachers are going to have, which generic people don't have.

Defining a Teacher() constructor function (defining new)

```
1  function Teacher(first, last, age, gender, interests, subject)
2    this.name = {
3      first,
4      last
5    };
6    this.age = age;
7    this.gender = gender;
8    this.interests = interests;
9    this.subject = subject;
10 }
```

- Redefining the properties anew, not inheriting them from Person()
- It also takes more lines of code.

Giving Teacher() a new greeting() function

- The easiest way to do this is to define it on Teacher()'s prototype

```
> function Person(first, last, age, gender,
interests) {
  this.name = {
    first,
    last
  };
  this.age = age;
  this.gender = gender;
  this.interests = interests;
};
<- undefined
```

```
> function Teacher(first, last, age,
gender, interests, subject) {
  Person.call(this, first, last, age,
gender, interests);

  this.subject = subject;
}
<- undefined
```

```
> Teacher.prototype.greeting = function() {
  var prefix;

  if (this.gender === 'male' ||
this.gender === 'Male' || this.gender ===
'm' || this.gender === 'M') {
    prefix = 'Mr.';
  } else if (this.gender === 'female' ||
this.gender === 'Female' || this.gender
=== 'f' || this.gender === 'F') {
    prefix = 'Mrs.';
  } else {
    prefix = 'Mx.';
  }

  alert('Hello. My name is ' + prefix + '
' + this.name.last + ', and I teach ' +
this.subject + '.');
};
```

Agenda

- Introduction to events
- Introducing JavaScript objects
- Object-oriented JavaScript for beginners
- Object prototypes
- Inheritance in JavaScript
- **Working with JSON data**

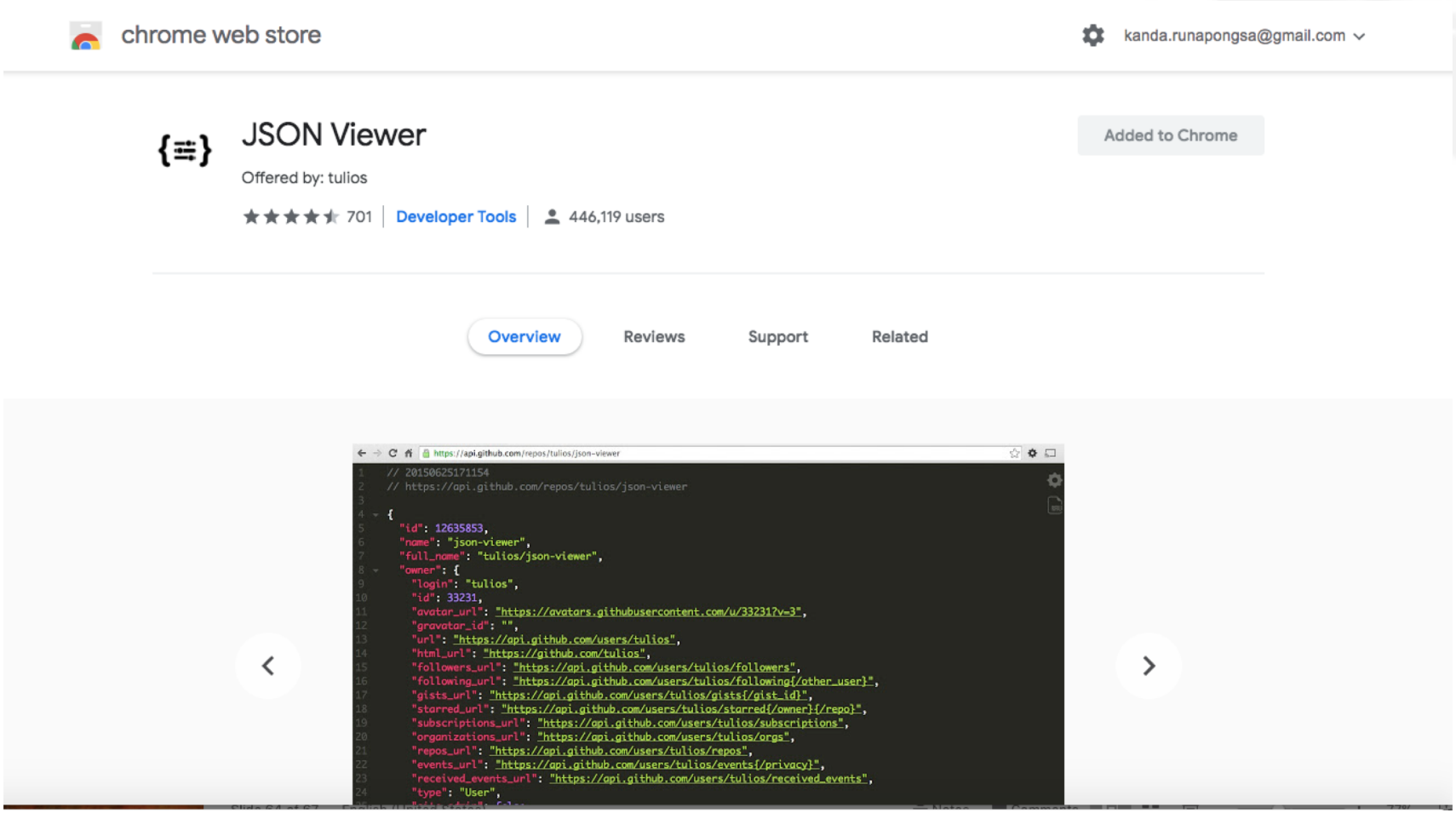
Working with JSON data

- JavaScript Object Notation (JSON) is a standard text-based format for representing structured data based on JavaScript
- It is commonly used for transmitting data in web applications

JSON object

- JSON exists as a string – useful when you want to transmit data across a network
- It needs to be converted to a native JavaScript object when you want to access data
  - Using JSON object
- JSON object can be stored in its own file, which is basically just a text file with an extension of .json, and a MIME type of application/json

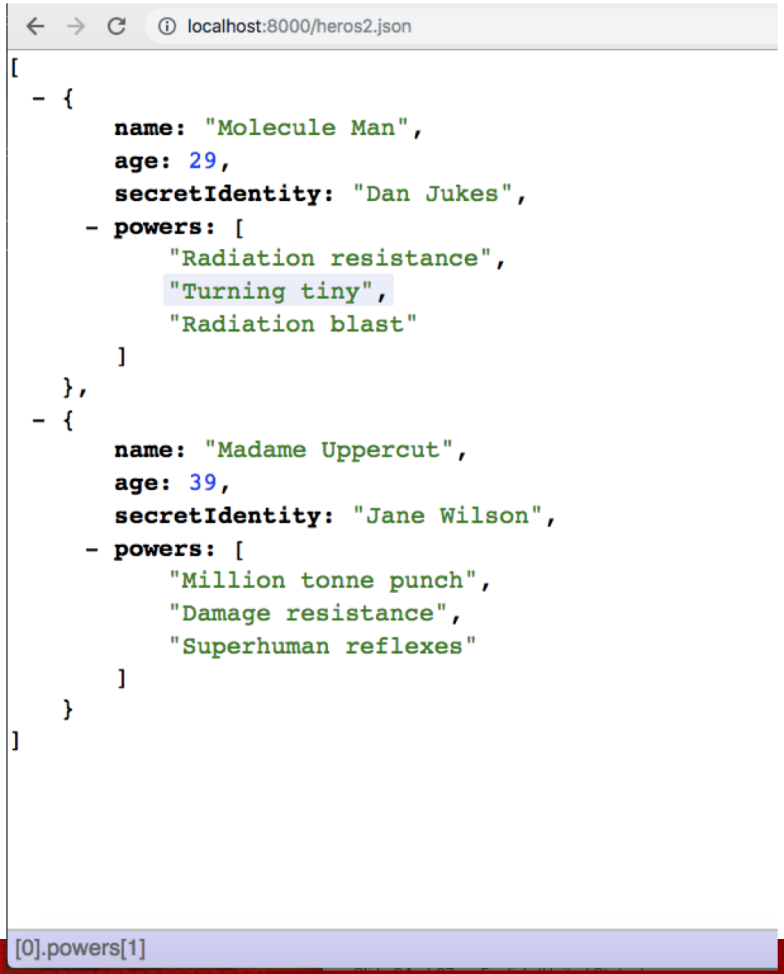
Installing JSON Viewer Chrome Extension



Sample JSON data 1

Arrays as JSON

- An Array can also be a JSON object



Sample JSON parsing

```
1  <!DOCTYPE html>
2  <html>
3      <h2>JSON Object Creation in JavaScript</h2>
4      <p id="demo"></p>
5      <script>
6          var obj = {"name":"Manee Deejai",
7                     "street":"123 Mittrapap Rd.",
8                      "phone":"043 009700"};
9          document.getElementById("demo").innerHTML =
10             obj.name + "<br/>" +
11             obj.street + "<br/>" +
12             obj.phone;
13     </script>
14 </html>
```

Obtaining the JSON

- Use an API called XMLHttpRequest (often called **XHR**)
- This is a very useful JavaScript object that allows us to make network requests to retrieve resources from a server via JavaScript

Sample JSON Parsing Result

localhost:8000/heros.json


```
{
   squadName: "Super hero squad",
   homeTown: "Metro City",
   formed: 2016,
   secretBase: "Super tower",
   active: true,
 - members: [
     - {
           name: "Molecule Man",
           age: 29,
           secretIdentity: "Dan Jukes",
       - powers: [
             "Radiation resistance",
             "Turning tiny",
             "Radiation blast"
           ]
       },
     - {
           name: "Madame Uppercut",
           age: 39,
           secretIdentity: "Jane Wilson",
       - powers: [
             "Million tonne punch",
             "Damage resistance",
             "Superhuman reflexes"
           ]
       },
```


members

localhost:8000/heros.html

Molecule Man

Madame Uppercut

Eternal Flame

Sample JSON Parsing code

```
 9  <body>
10    <section></section>
11    <script>
12    var section = document.querySelector('section');
13    var requestURL = 'http://localhost:8000/heros.json';
14    var request = new XMLHttpRequest();
15    request.open('GET', requestURL);
16    request.responseType = 'json';
17    request.send();
18    request.onload = function() {
19        var superHeroes = request.response;
20        showHeroes(superHeroes);
21    }
22    function showHeroes(jsonObj) {
23        var heroes = jsonObj['members'];
24        for(var i = 0; i < heroes.length; i++) {
25            var myArticle = document.createElement('article');
26            var myH2 = document.createElement('h2');
27            myH2.textContent = heroes[i].name;
28            myArticle.appendChild(myH2);
29            section.appendChild(myArticle);
30        }
31    }
32    </script>
33  </body>
```

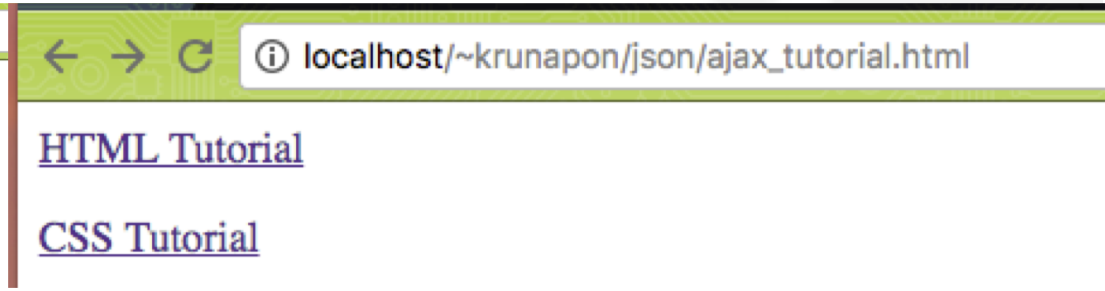
Exercise 5

- File Input

localhost/~krunapon/json/tutorials.json

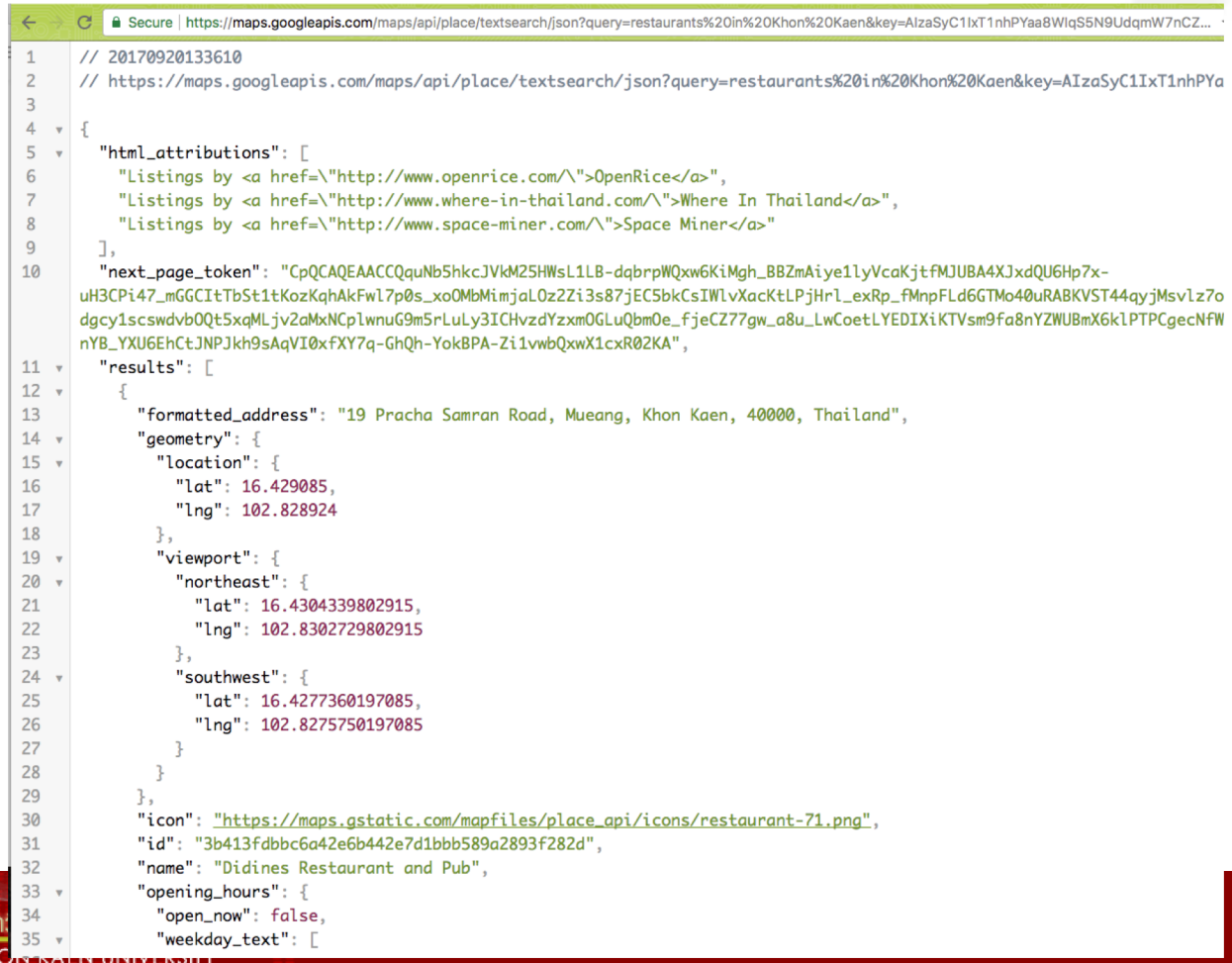
```
1  // 20170920124935
2  // http://localhost/~krunapon/json/tutorials.json
3
4  {
5    "tutorials": [
6      {
7        "display": "HTML Tutorial",
8        "url": "http://www.w3schools.com/html/default.asp"
9      },
10     {
11       "display": "CSS Tutorial",
12       "url": "http://www.w3schools.com/css/default.asp"
13     }
14   ]
15 }
```

- Output



Exercise 6

- Web Input: https://maps.googleapis.com/maps/api/place/textsearch/json?query=restaurants%20in%20Khon%20Kaen&key=AIzaSyC1IxT1nhPYaa8WIqS5N9UdqmW7nCZr42A



Exercise 6

- Web Output

References

- https://medium.freecodecamp.org/how-to-build-an-html-calculator-app-from-scratch-using-javascript-4454b8714b98
- https://developer.mozilla.org/en-US/docs/Learn/JavaScript/First_steps/A_first_splash
- https://developer.mozilla.org/en-US/docs/Learn/JavaScript/First_steps/Silly_story_generator