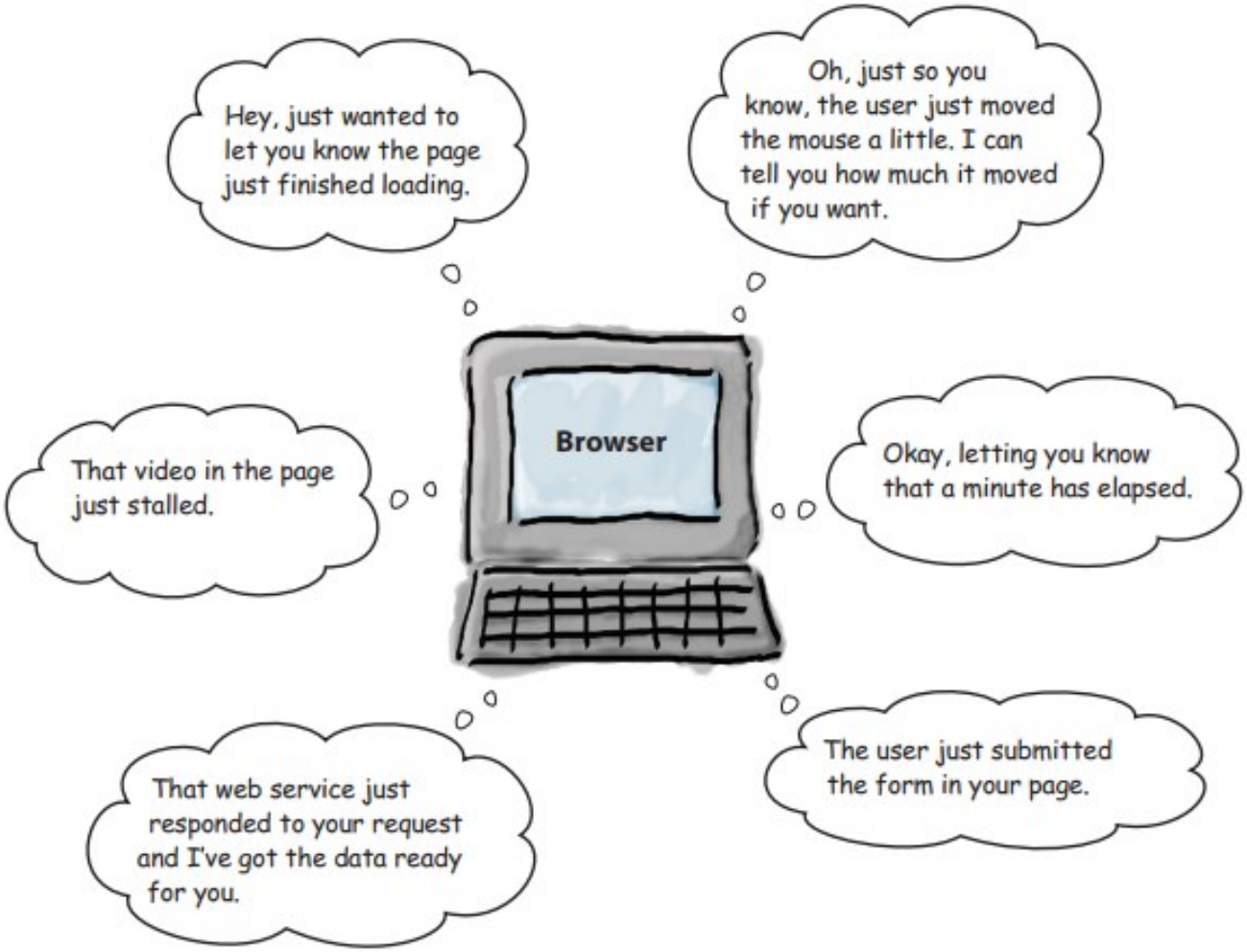
JavaScript

CHAPTER IV. HANDLING EVENTS

Ths. Hồ Đức Lĩnh
Mail: duclinh47th@gmail.com
Phone: 0905 28 68 87

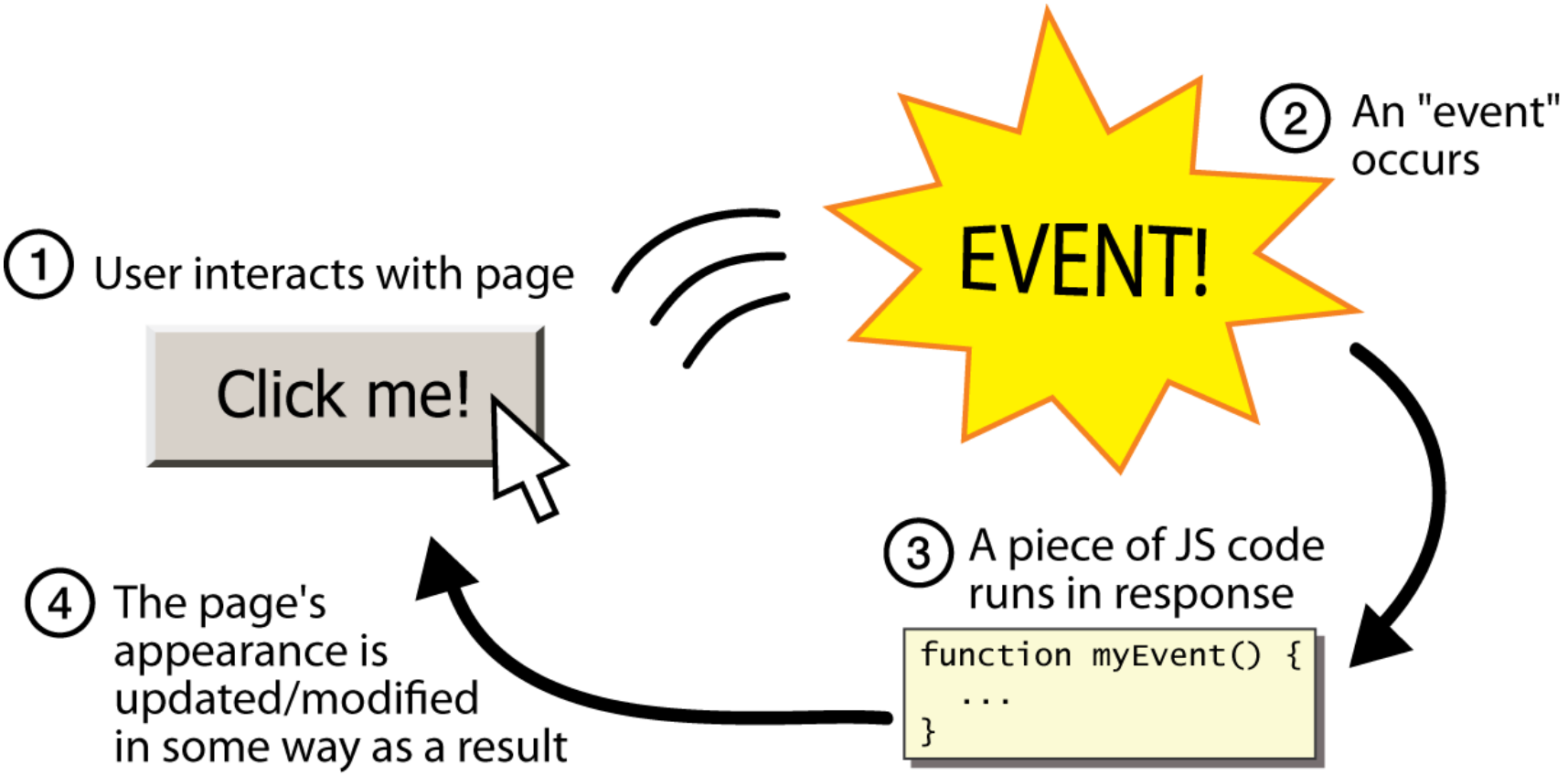
What are events?



What are events?

- We're sure you know by now that after the browser retrieves and displays your page, it doesn't just sit there.
- Behind the scenes, a lot is going on: users are clicking buttons, the mouse location is being tracked, additional data is becoming available on the network, windows are getting resized, timers are going off, the browser's location could be changing, and so on → All these things cause ***events*** to be triggered

What are events?



What's an event handler?

- We write *handlers* to handle events. Handlers are typically small pieces of code that know what to do when an event occurs.
- In terms of code, a handler is just a function.
- When an event occurs, its handler function is called.

POPULAR EVENTS

LOAD

`load` is fired on `window` and the `body` element when the page has finished loading.

MOUSE EVENTS

`click` fires when a mouse button is clicked. `dblclick` when the mouse is clicked two times. Of course in this case `click` is fired just before this event. `mousedown`, `mousemove` and `mouseup` can be used in combination to track drag-and-drop events. Be careful with `mousemove`, as it fires many times during the mouse movement (see *throttling* later)

KEYBOARD EVENTS

`keydown` fires when a keyboard button is pressed (and any time the key repeats while the button *stays* pressed). `keyup` is fired when the key is released.

SCROLL

The `scroll` event is fired on `window` every time you scroll the page. Inside the event handler you can check the current scrolling position by checking `window.scrollY`.

Keep in mind that this event is not a one-time thing. It fires a lot of times during scrolling, not just at the end or beginning of the scrolling, so don't do any heavy computation or manipulation in the handler - use *throttling* instead.

Event Properties and Methods

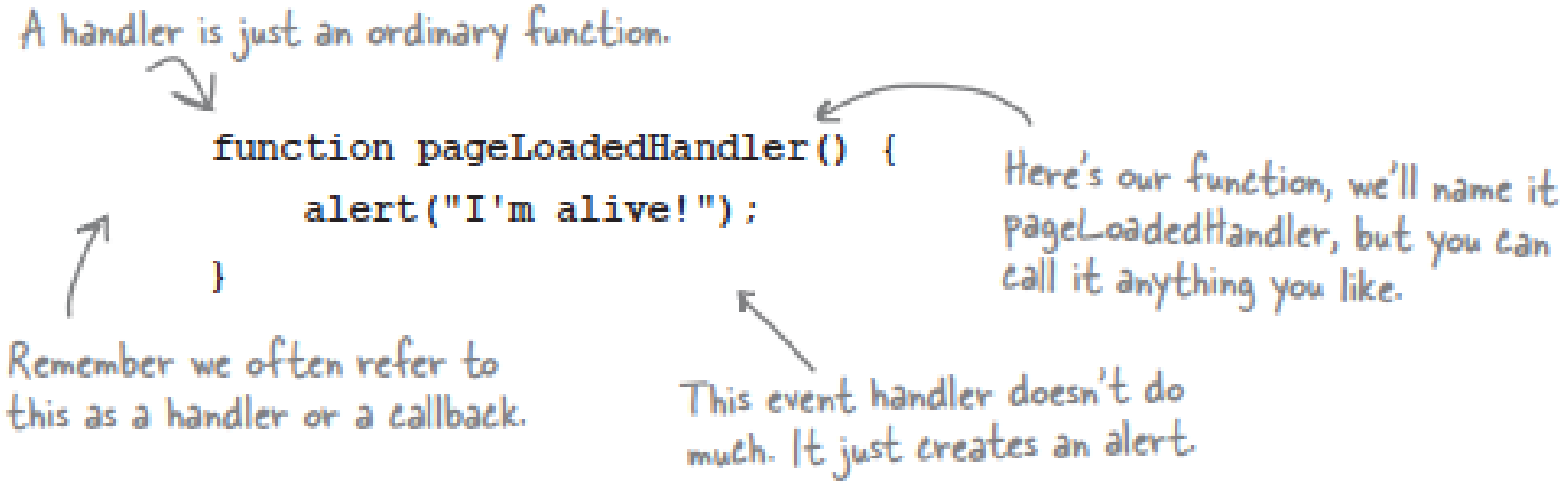
Property/Method	Description
bubbles	Returns whether or not a specific event is a bubbling event
cancelBubble	Sets or returns whether the event should propagate up the hierarchy or not
cancelable	Returns whether or not an event can have its default action prevented
composed	Returns whether the event is composed or not
createEvent()	Creates a new event
composedPath()	Returns the event's path
currentTarget	Returns the element whose event listeners triggered the event
defaultPrevented	Returns whether or not the preventDefault() method was called for the event
eventPhase	Returns which phase of the event flow is currently being evaluated

Event Properties and Methods (Cont ..)

isTrusted	Returns whether or not an event is trusted
preventDefault()	Cancels the event if it is cancelable, meaning that the default action that belongs to the event will not occur
stopImmediatePropagation()	Prevents other listeners of the same event from being called
stopPropagation()	Prevents further propagation of an event during event flow
target	Returns the element that triggered the event
timeStamp	Returns the time (in milliseconds relative to the epoch) at which the event was created
type	Returns the name of the event

How to create your first event handler?

1. First we need to write a function that can handle the page load event when it occurs. In this case, the function is going to announce to the world "I'm alive!" when it knows the page is fully loaded.



How to create your first event handler?

2. Now that we have a handler written and ready to go, we need to wire things up so the browser knows there's a function it should invoke when the load event occurs. To do that we use the onload property of the window object, like this:

```
window.onload = pageLoadedHandler;
```

In the case of the load event, we assign the name of the handler to the window's onload property.

Now when the page load event is generated, the pageLoadedHandler function is going to be called.

We're going to see that different kinds of events are assigned handlers in different ways.

3. That's it! Now, with this code written, we can sit back and know that the browser will invoke the function assigned to the window.onload property when the page is loaded.

Test drive your event

First the browser loads your page, and starts parsing the HTML and building up the DOM.

```
<!doctype html>
<html lang="en">
<head>
  <meta charset="utf-8">
  <title> I'm alive! </title>
  <script>
    window.onload = pageLoadedHandler;
    function pageLoadedHandler() {
        alert("I'm alive!");
    }
  </script>
</head>
<body>
</body>
</html>
```

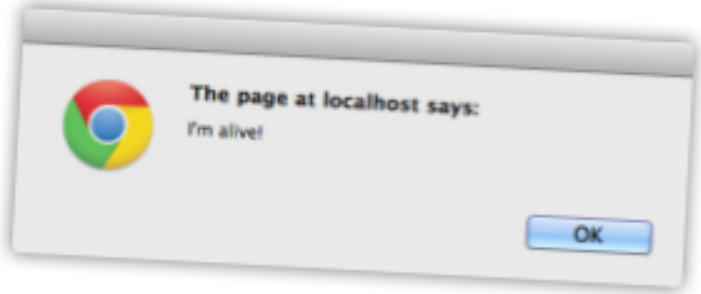
When it gets to your script the browser starts executing the code.

For now, the script just defines a function, and assigns that function to the window.onload property. Remember this function will be invoked when the page is fully loaded.

Then the browser continues parsing the HTML.

When the browser is done parsing the HTML, and the DOM is ready, the browser calls the page load handler.

Which in this case creates the "I'm alive" alert.



creating a game

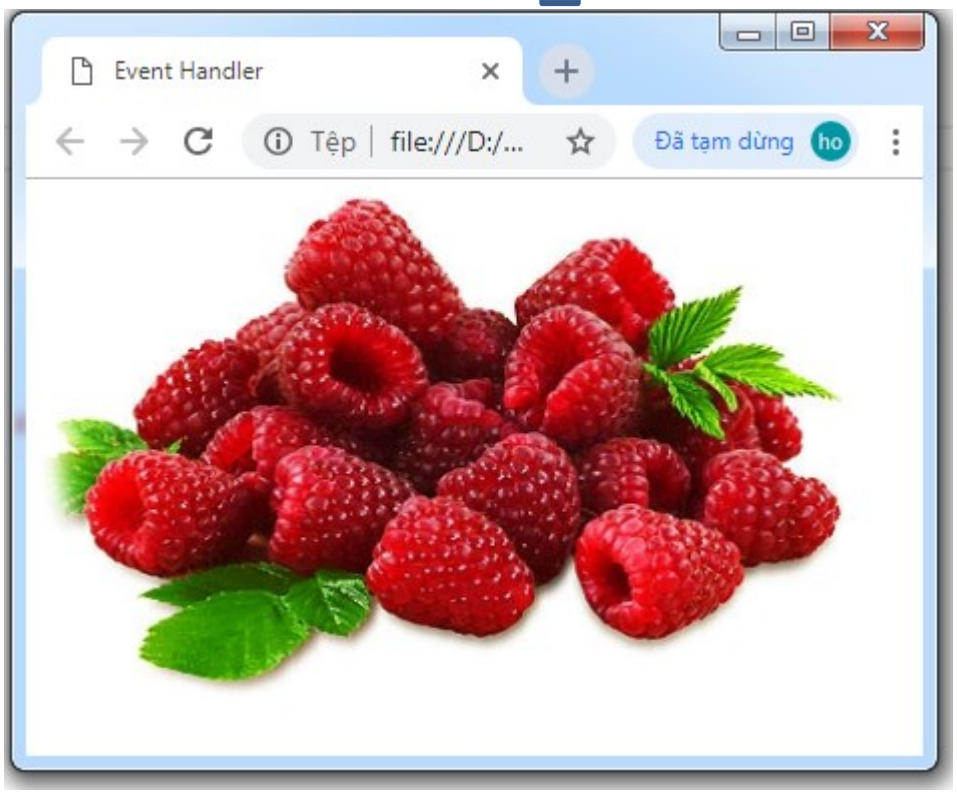


Image before click

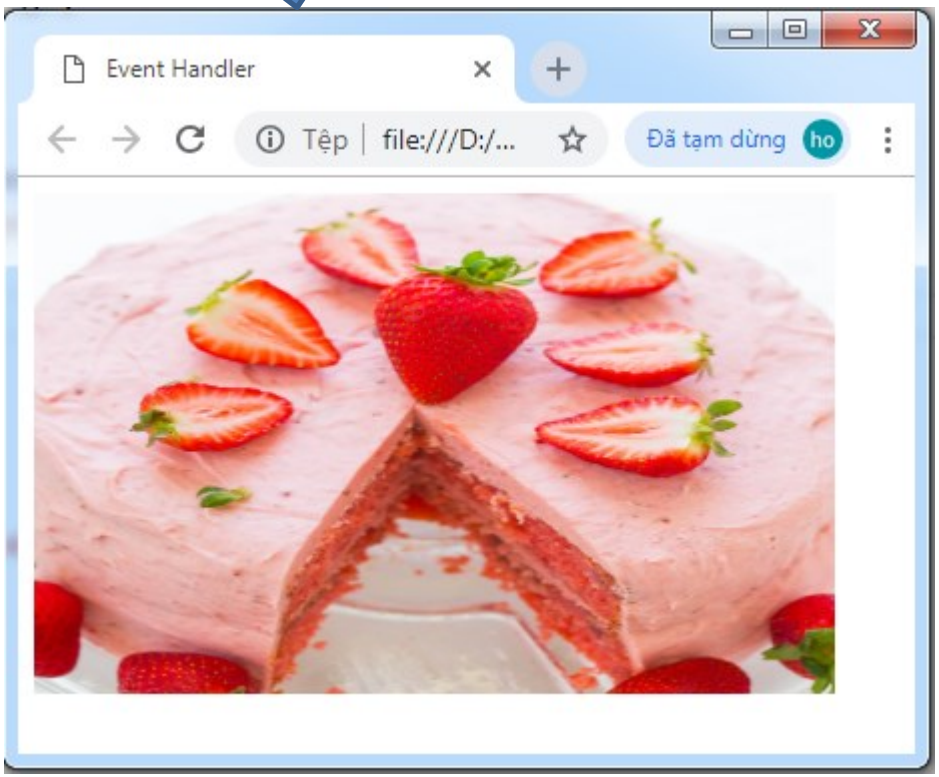


Image After clicked

Implementing the game

1. Access the image object in the DOM and assign a handler to its **onclick** property.

```
window.onload = init;
function init() {
    var image = document.getElementById("zero");
    image.onclick = showAnswer;
}
```

Here we're grabbing a reference to the image element and assigning it to the image variable.

Using the image object from the DOM, we're assigning a handler to its onclick property.

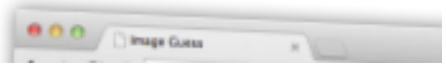
2. In your **handler**, write the **code to change** the **image src attribute**

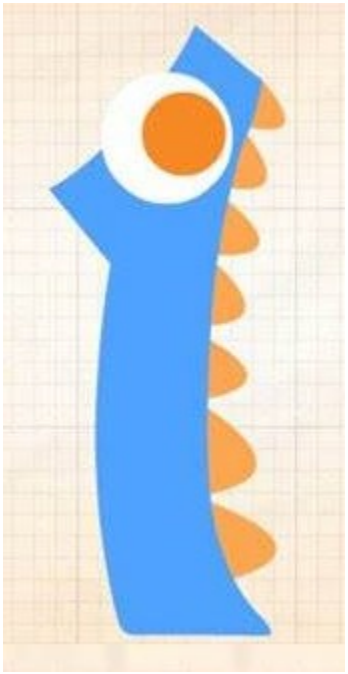
First, we have to get the image from the DOM again.

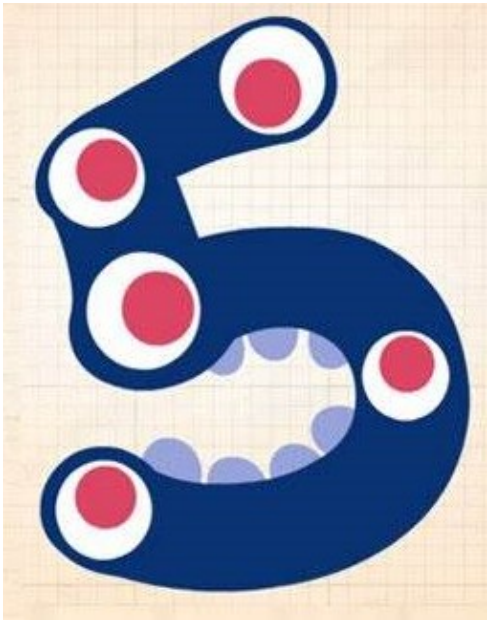
```
function showAnswer() {
    var image = document.getElementById("zero");
    image.src = "zero.jpg";
}
```

Remember the blurred version is named "zeroblur.jpg" and the unblurred is named "zero.jpg".

Once we have the image, we can change it by setting its src property to the unblurred image.

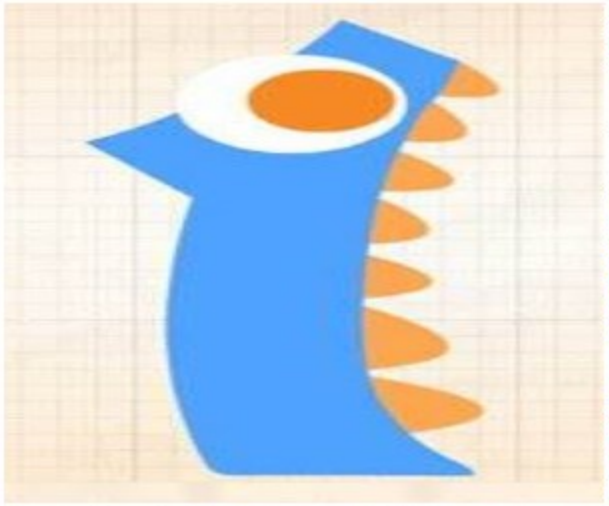


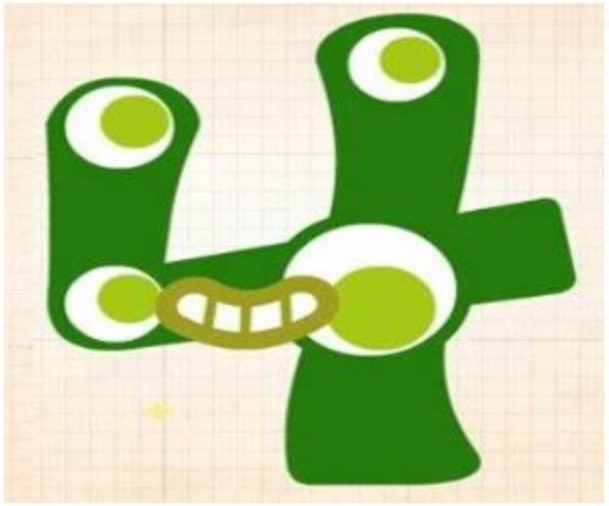




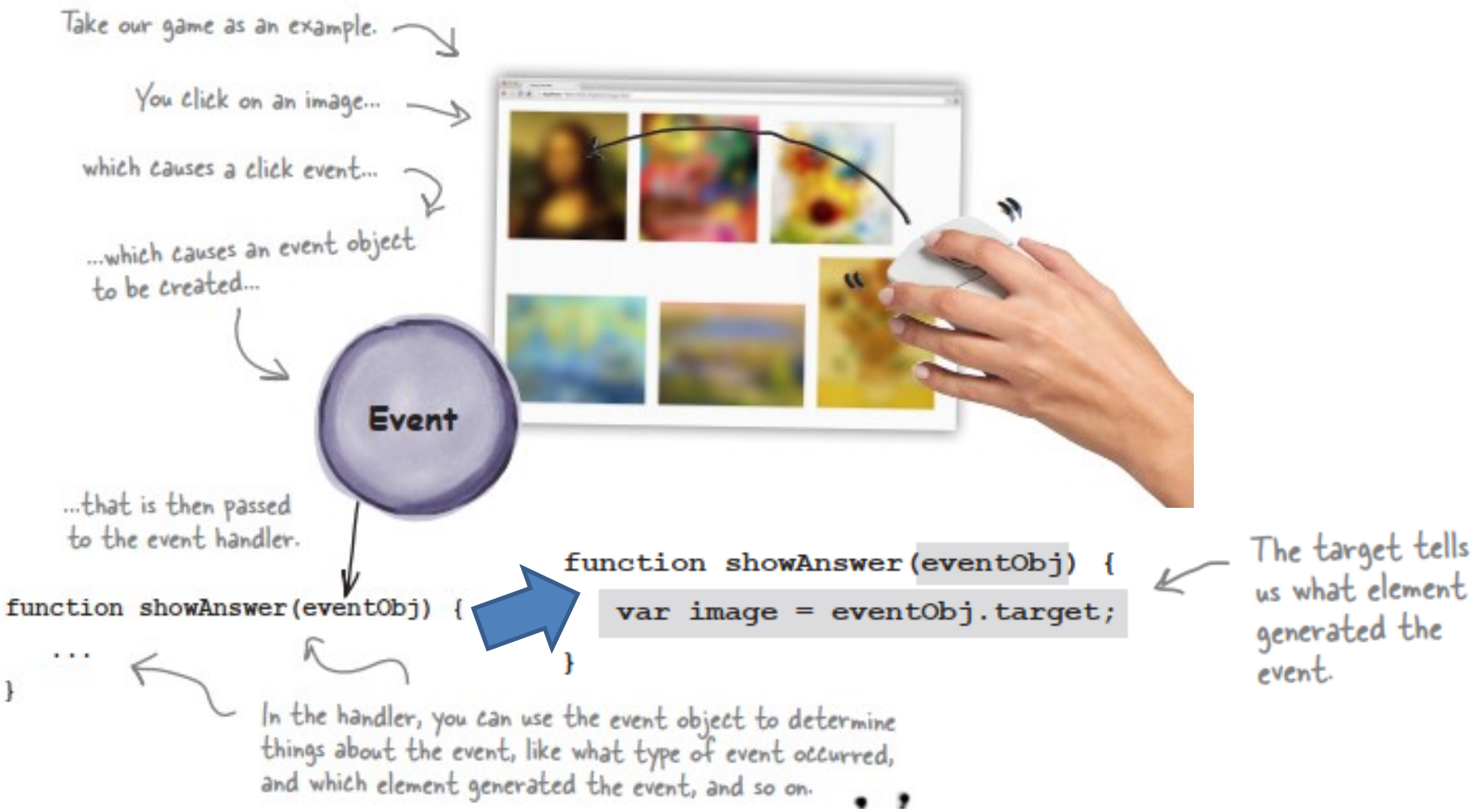
Onload page

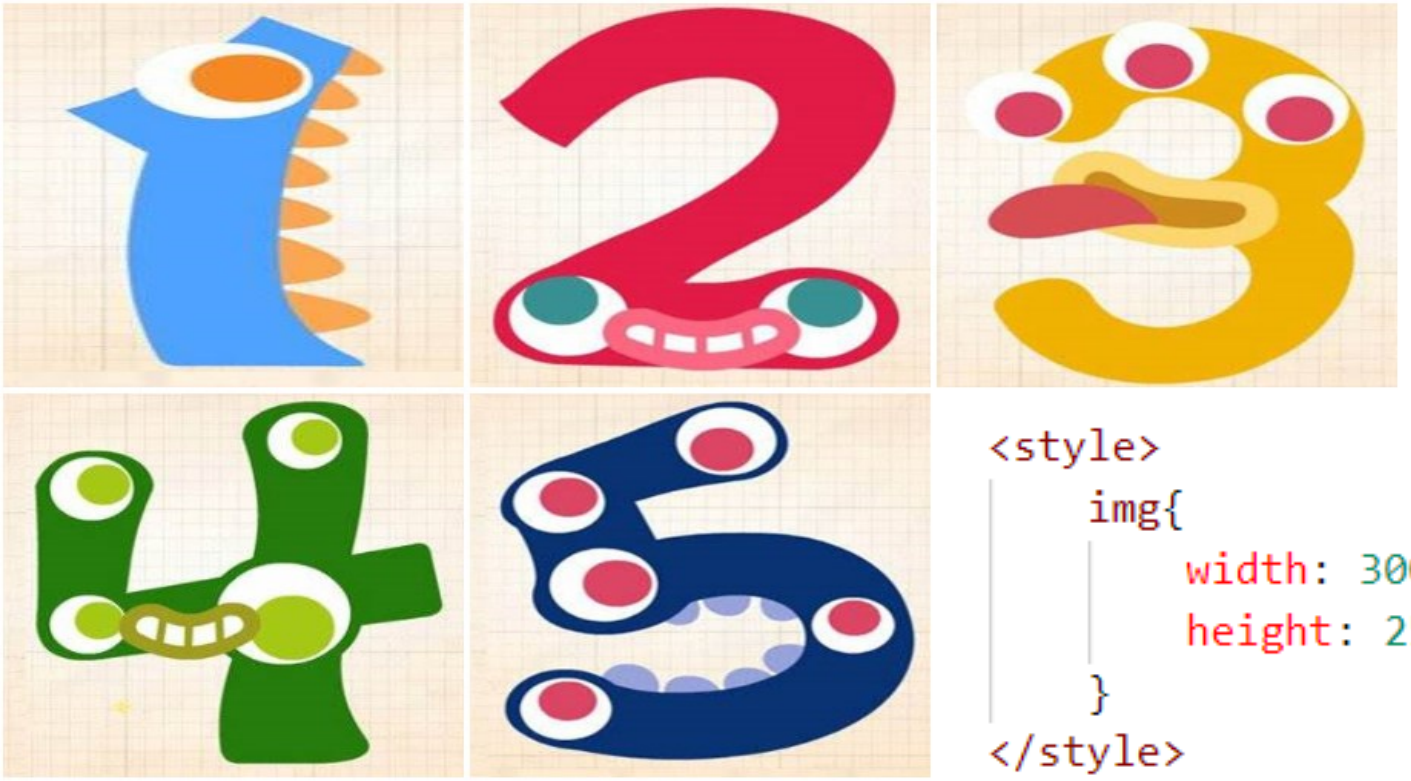
After Clicked on the Image





How the event object works?





```
<style>
    img{
        width: 300px;
        height: 250px;
    }
</style>
```

Here's the HTML again.

```
<body>
    <img src="images/num_1.png" id="one">
    <img src="images/num_2.png" id="two">
    <img src="images/num_3.png" id="three">
    <img src="images/num_4.png" id="four">
    <img src="images/num_5.png" id="five">
</body>
```

Each of the images has an id, and the id corresponds to the unblurred image name. So the image with id "zero" has an unblurred image of "zero.jpg". And the image with id "one" has an unblurred image of "one.jpg" and so on...

Remember you're getting passed an event object each time an image is clicked on.

```
function showImage(evenObject){
    //reference to the image element
    var image = evenObject.target;
    //use the id property of that object to get the name of image
    var name = image.id;
    name = "images/"+ name +".jpg";
    // console.log(name);
    //set the src of the image to that name
    image.src = name;
}
```

The event object's target property is a reference to the image element that was clicked.

We can then use the id property of that object to get the name of the unblurred image.

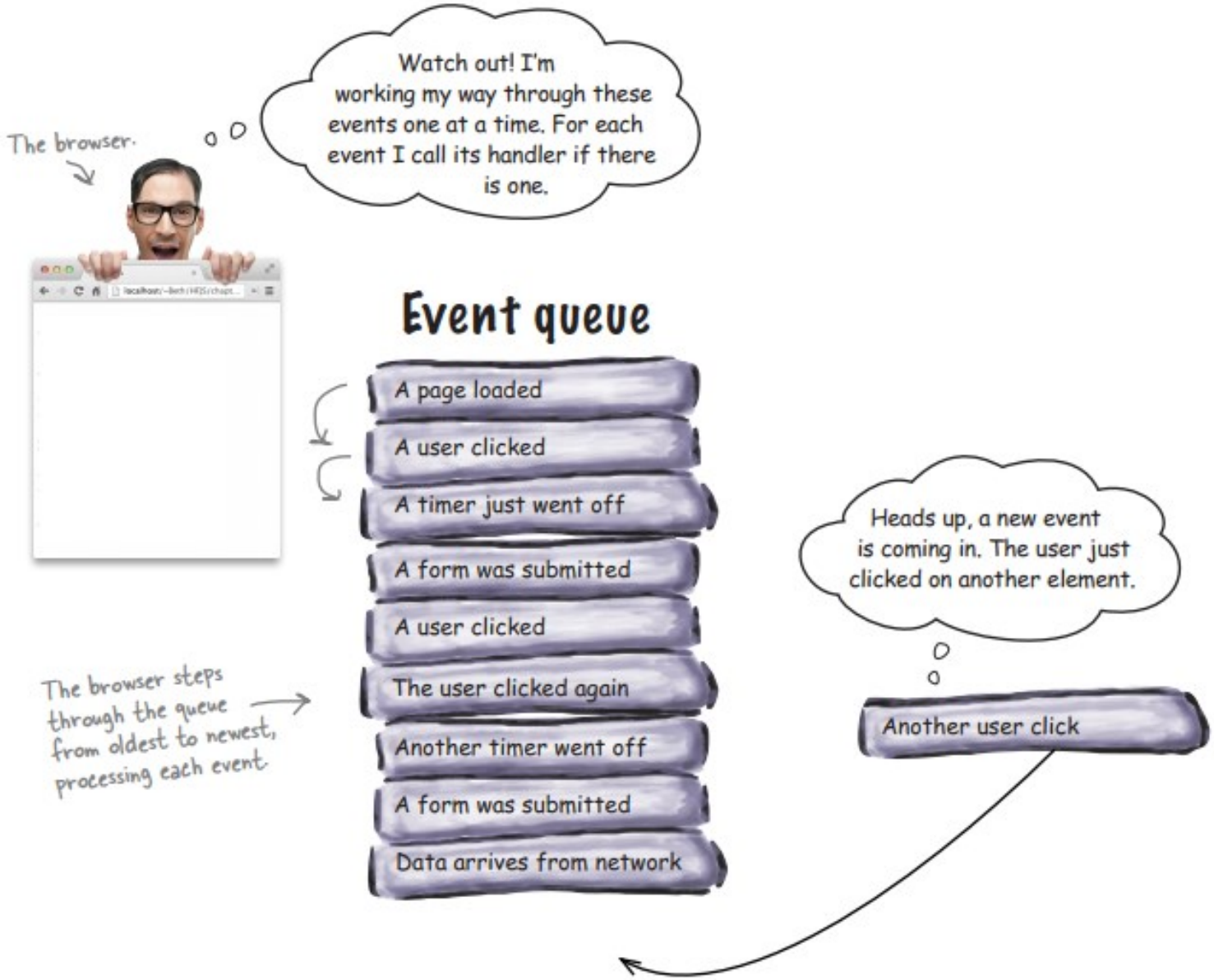
As you know, once you change the src property of the image, the browser will immediately retrieve that new image and display it in the page in place of the blurred version.

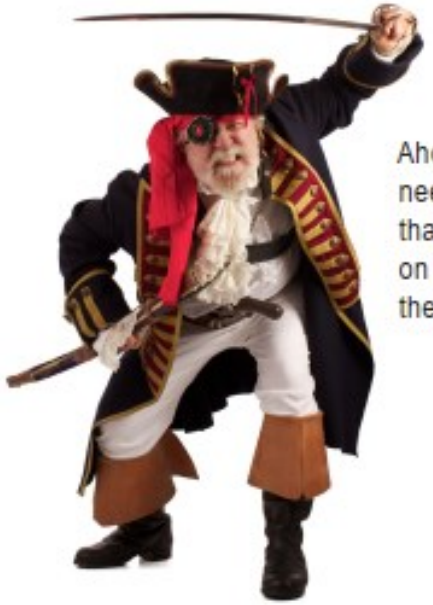
Suggestion Code

```
<script>
    window.onload = init;
        function init(){
            var images = document.getElementsByTagName("img");
            for(var i = 0; i< images.length; i++){
                // console.log(images[i]);
                images[i].onclick = showImage;
            }
        }

        function showImage(evenObject){
            //reference to the image element that clicked
            var image = evenObject.target;
            //use the id property of that object to get the name of image
            var name = image.id;
            name = "images/"+ name +".jpg";
            // console.log(name);
            //set the src of the image to that name
            image.src = name;
        }
</script>
```

Events and queues





Exercise

Ahoy matey! You've got a treasure map in your possession and we need your help in determining the coordinates of the treasure. To do that you're going to write a bit of code that displays the coordinates on the map as you pass the mouse over the map. We've got some of the code on the next page, but you'll have to help finish it.

Here's the map and X marks the spot!

After your code is written, just move the mouse over the X to see the coordinates of the treasure.

Your code will display the coordinates below the map.

COORDINATES: 10, 20

P.S. We highly encourage you to do this exercise, because we don't think the pirates are going to be too happy if they don't get their coordinates... Oh, and you'll need this to complete your code:

MOUSE EVENTS

`click` fires when a mouse button is clicked. `dblclick` when the mouse is clicked two times. Of course in this case `click` is fired just before this event. `mousedown`, `mousemove` and `mouseup` can be used in combination to track drag-and-drop events. Be careful with `mousemove`, as it fires many times during the mouse movement (see *throttling* later)

The mousemove event

The mousemove event notifies your handler when a mouse moves over a particular element. You set up your handler using the element's **onmousemove** property. Once you've done that you'll be passed an event object that provides these properties:

clientX, clientY: the x (and y) position in pixels of your mouse from the left side (and top) of the browser window.

screenX, screenY: the x (and y) position in pixels of your mouse from the left side (and top) of the user's screen.

pageX, pageY: the x (and y) position in pixels of your mouse from the left side (and top) of the browser's page.

Move mouse to find the coordinates ...

Map coordinates: 573 , 197

```html
<div class="pirate-map">
    <img src="images/map.jpg" alt="" id="map">
    <p id="coords">Move mouse to find the coordinates ...</p>
</div>
```

```html
<script>
    window.onload = init;

    function init(){
        var map = document.getElementById("map");
        map.onmousemove = showCoords;
    }

    function showCoords(evenObj){
        var map = document.getElementById("coords");
        var x = evenObj.clientX;
        var y = evenObj.clientY;
        // var x = evenObj.pageX;
        // var y = evenObj.pageY;
        map.innerHTML = "Map coordinates: " + x+ " , " + y;
        map.setAttribute("class","map-coords");
    }
</script>
```

```html
<style>
    .pirate-map{
        margin: 0 auto;
        width: 700px;
        height: 450px;
    }
    .pirate-map img{
        width: 100%;
        height: 100%;
    }
    .map-coords{
        font-size: 25px;
        color: red;
    }
</style>
```

Even more events

- we've seen three types of events:
  - the **load event**, which occurs when the browser has loaded the page;
  - the **click event**, which occurs when a user clicks on an element in the page;
  - the **mousemove event**, which occurs when a user moves the mouse over an element

time-based events

- with time-based events, rather than assigning a handler to a property, you call a function, setTimeout, instead and pass it your handler

First we write an event handler. This is the handler that will be called when the time event has occurred.

```
function timerHandler() {
    alert("Hey what are you doing just sitting there staring at a blank screen?");
}
```

All we're doing in this event handler is showing an alert.

```
setTimeout(timerHandler, 5000);
```

And here, we call setTimeout, which takes two arguments: the event handler and a time duration (in milliseconds).

Using setTimeout is a bit like setting a stop watch.

Here we're asking the timer to wait 5000 milliseconds (5 seconds).

And then call the handler timerHandler.

What’s happened after The page loaded a 3 second?

```
<script>
    window.onload = init;
    function init(){
        setTimeout(TimerHandler,3000);
    }
    function TimerHandler() {
        alert("Hey what are you doing just sitting there staring at a blank screen?");
    }
</script>
```

file:///D:/MY%20LECTURES/JavaScript/Prepaird/event4.html

t/Prepaird/event4.html

Trang này cho biết

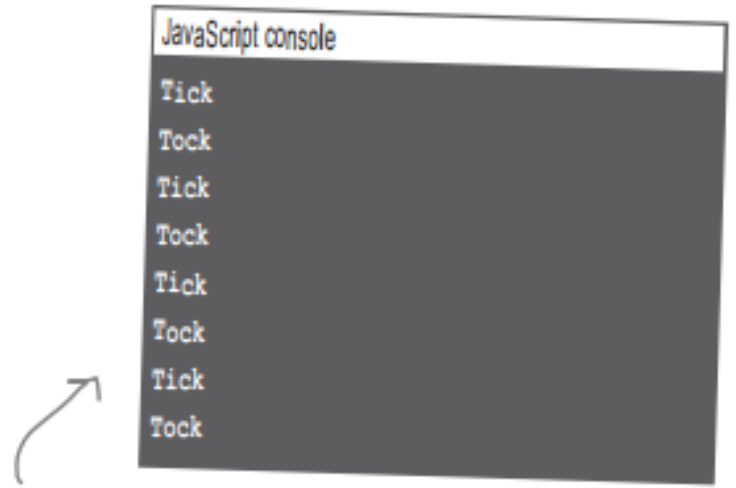
Hey what are you doing just sitting there staring at a blank screen?

OK

Exercise

Here's the code.

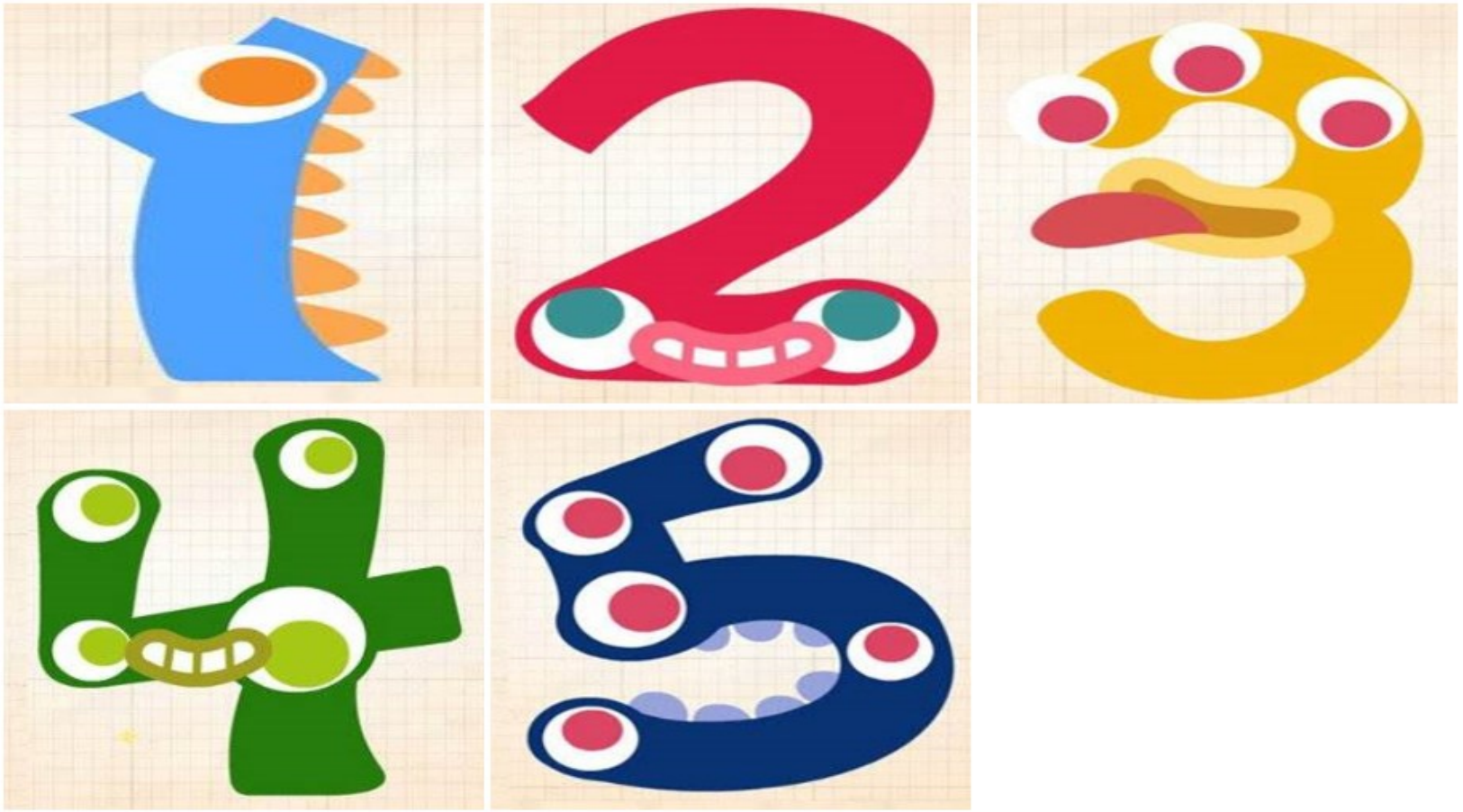
```
var tick = true;
function ticker() {
    if (tick) {
        console.log("Tick");
        tick = false;
    } else {
        console.log("Tock");
        tick = true;
    }
}
setInterval(ticker, 1000);
```



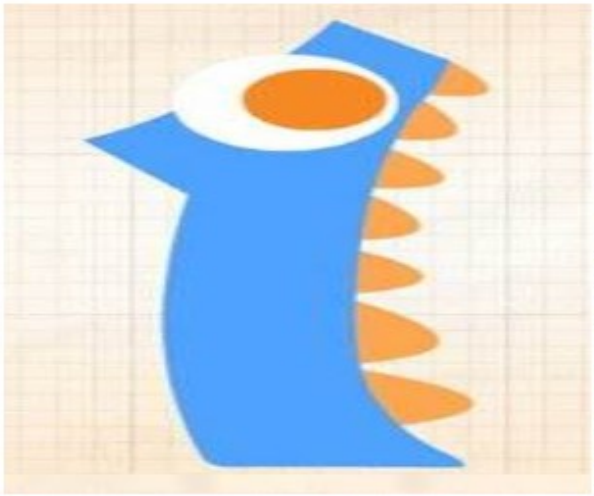
Here's the output.

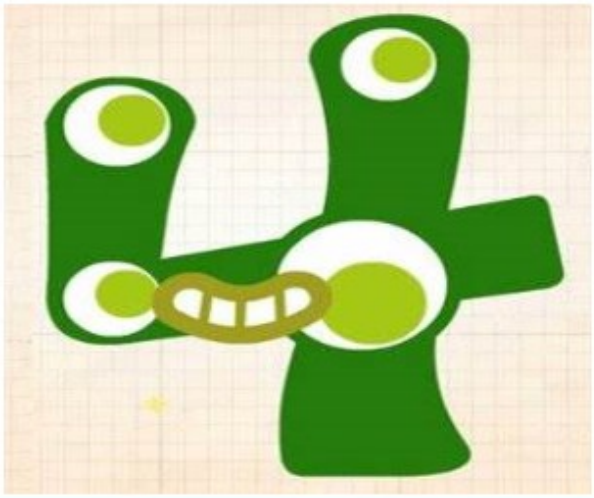
Your analysis goes here.

Onload page

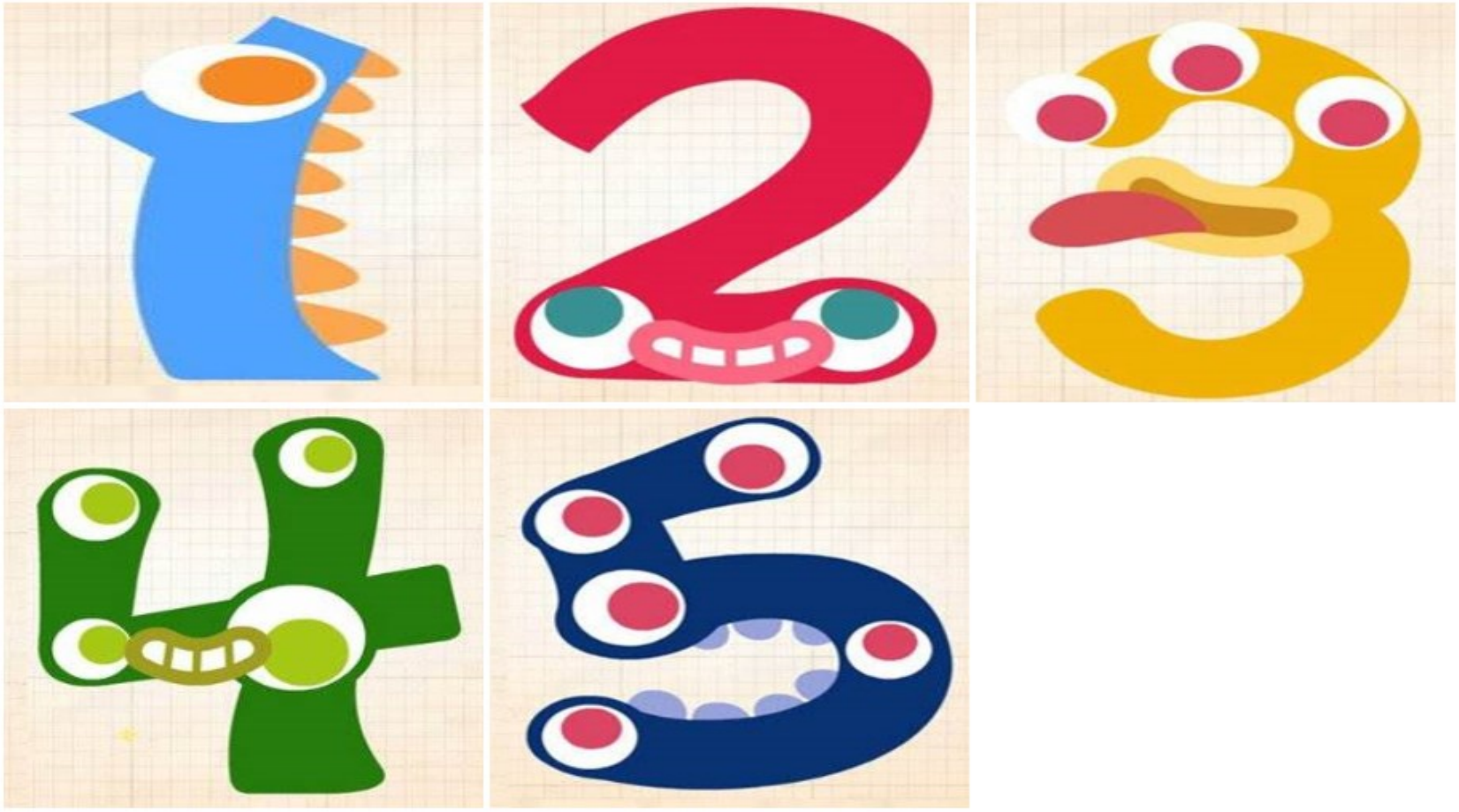


Clicked on number Two





After 1000 milliseconds – Flipping again



```
function showImage(evenObject){
    //reference to the image element that clicked
    var image = evenObject.target;
    //use the id property of that object to get the name of image
    var name = image.id;
    name = "images/"+ name +".jpg";
    // console.log(name);
    //set the src of the image to that name
    image.src = name;
    //set timeout to replace image
    setTimeout(replaceImage,1000,image);
}
function replaceImage(image){
    var name = image.id;
    name = "images/"+"num_"+name+".png";
    image.src = name;
}
```

Event Soup


click
Get this event when you click (or tap) in a web page.
load
The event you get when the browser has completed loading a web page.
unload
This event is generated when you close the browser window, or navigate away from a web page.
mousemove
When you move your mouse over an element, you'll generate this event.
mouseover
When you put your mouse over an element, you'll generate this event.
keypress
This event is generated every time you press a key.
mouseout
And you'll generate this event when you move your mouse off an element.
resize
Whenever you resize your browser window, this event is generated.
dragstart
If you drag an element in the page, you'll generate this event.
touchstart
On touch devices, you'll generate a touchstart event when you touch and hold an element.
play
Got <video> in your page? You'll get this event when you click the play button.
drop
You'll get this event when you drop an element you've been dragging.
touchend
And you'll get this event when you stop touching.
pause
And this one when you click the pause button.
We've scratched the surface of events, using load, click, mousemove, mouseover, mouseout, resize and timer events. Check out this delicious soup of events you'll encounter and will want to explore in your web programming.

Bài tập: Sử dụng sự kiện Hover vào từng Image thì lật ảnh tương ứng, di chuyển chuột ra vị trí khác thì Image đó được lật lại

JSON

What is JSON?

- JSON stands for **J**ava**S**cript **O**bject **N**otation
- JSON is a lightweight data-interchange format
- JSON is "self-describing" and easy to understand
- JSON is language independent *

Why use JSON?

- Since the JSON format is text only, it can easily be sent to and from a server, and used as a data format by any programming language.
- JavaScript has a built in function to convert a string, written in JSON format, into native JavaScript objects:

  JSON.parse()

- So, if you receive data from a server, in JSON format, you can use it like any other JavaScript object.

JSON Syntax

- JSON syntax is derived from JavaScript object notation syntax:
  - Data is in name/value pairs
  - Data is separated by commas
  - Curly braces hold objects
  - Square brackets hold arrays

JSON Data - A Name and a Value

- JSON data is written as **name/value pairs**.
- A name/value pair consists of a field name (in double quotes), followed by a colon, followed by a value:

Example

```
"name":"John"
```

JSON - Evaluates to JavaScript Objects

- The JSON format is almost identical to JavaScript objects.
- In JSON, *keys* must be strings, written with double quotes:

JSON

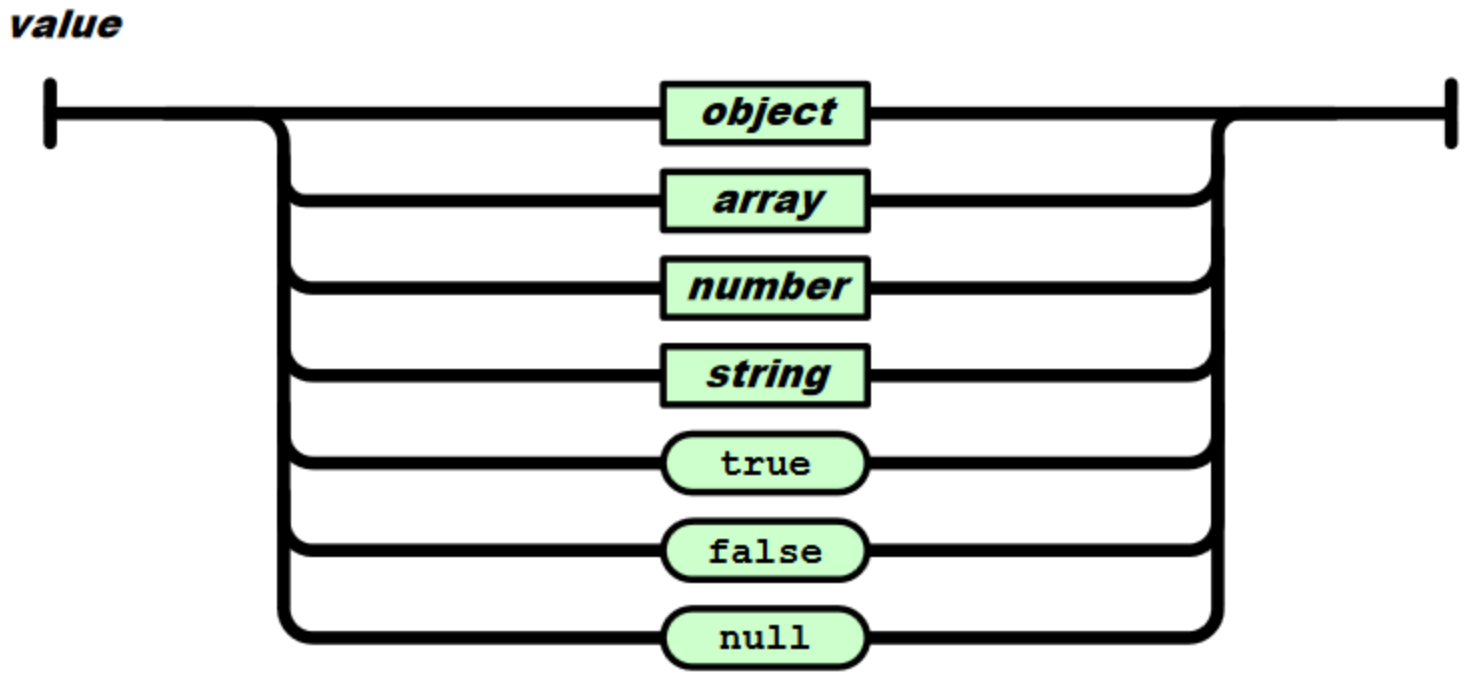
```
{ "name":"John" }
```

JavaScript

```
{ name:"John" }
```

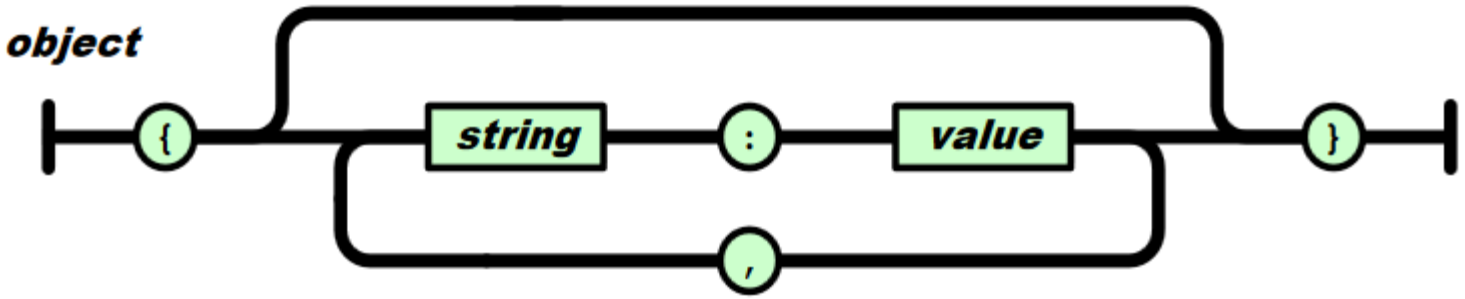
JSON Values

- A JSON value can be an **object**, **array**, **number**, **string**, **true**, **false**, or **null**.



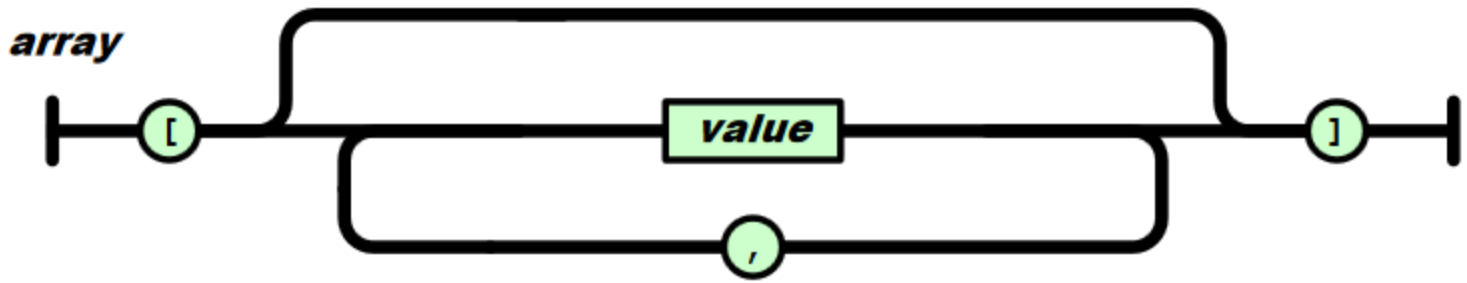
JSON Values - Objects

- An object structure is represented as a pair of curly bracket tokens surrounding zero or more name/value pairs.
- A name is a string
- A single colon token follows each name, separating the namefrom the value.



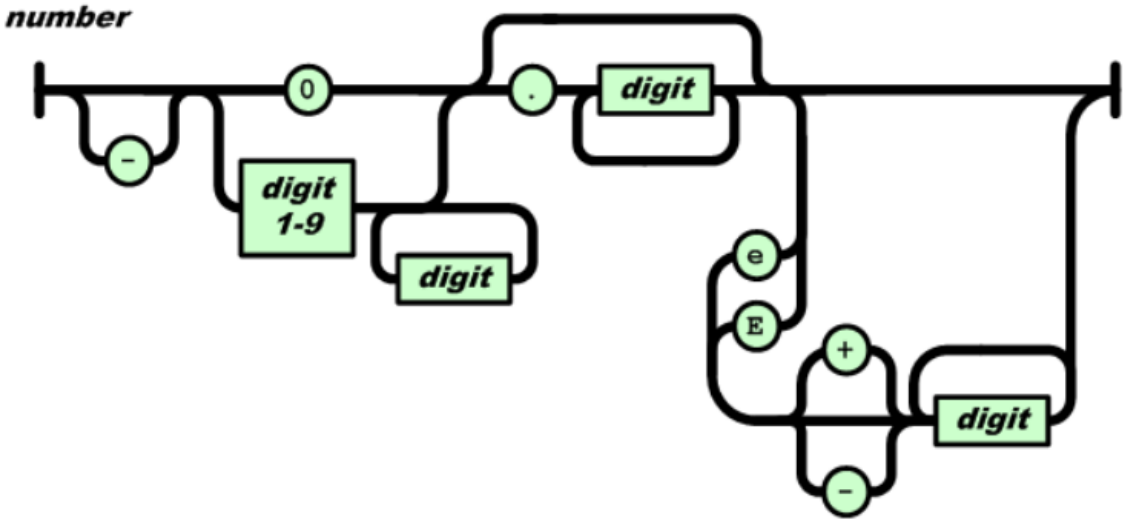
JSON Values - Arrays

- An array structure is a pair of square bracket tokens surrounding zero or more values.
- The values are separated by commas.

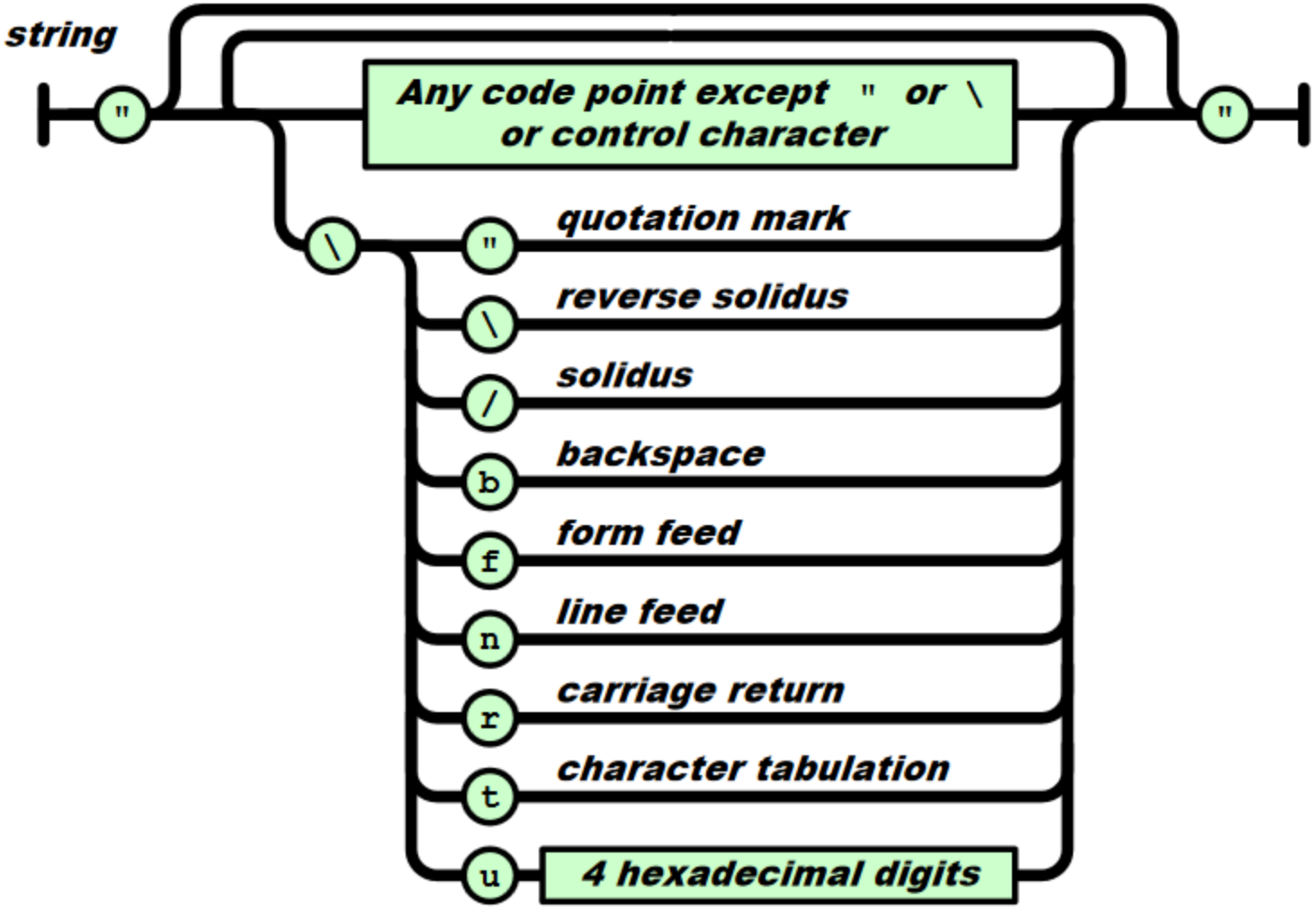


JSON Values - Numbers

- A number is a sequence of decimal digits with no superfluous leading zero. It may have a preceding minus sign (U+002D).
- It may have a fractional part prefixed by a decimal point (U+002E).
- It may have an exponent, prefixed by e(U+0065) or E(U+0045) and optionally +(U+002B) or –(U+002D). The digits are the code points U+0030 through U+0039

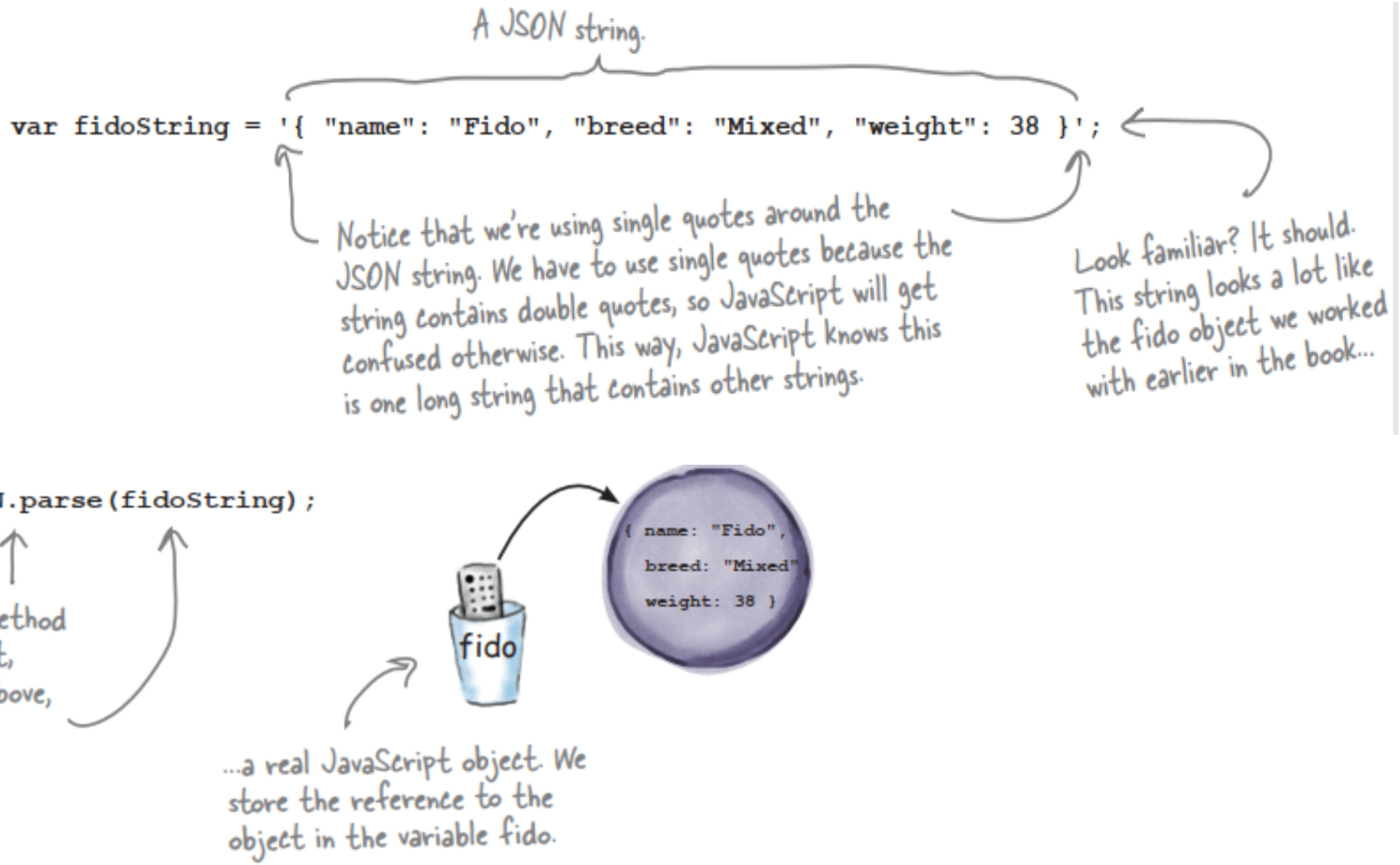


JSON Values - String



JSON Uses JavaScript Syntax

- JavaScript: create an object and assign data to it, like this:



JSON.parse()

- A common use of JSON is to exchange data to/from a web server.
- When receiving data from a web server, the data is always a string.
- Parse the data with **JSON.parse(),** and the data becomes a JavaScript object.

JSON.stringify - Converts JavaScript data to JSON data

Syntax Example:

```
var salesOrder = { number: 2341, name: "Chuck Norris", address:
{street:"123 Any Street", city:"New York", state:"NY", zip: 01234}};
var myJSON = JSON.stringify(salesOrder);
```

HTML

```
<p id="sales-order"></p><script>var salesOrder
= { number: 2341, name: "Chuck Norris",
address: {street:"123 Any Street", city:"New
York", state:"NY", zip:24569}};var myJSON =
JSON.stringify(salesOrder);document.getElementById
order").innerHTML = myJSON;</script>
```

Result

```
{"number":2341,"name":"Chuck Norris","addr
{"street":"123 Any Street","city":"New
York","state":"NY","zip":24569}}
```

Example

- JSON data file: **pets.json**

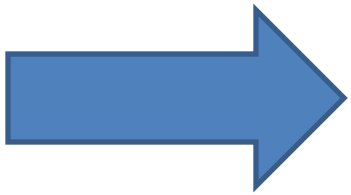
```
[ { "type":"cat",
    "name":"Pickles",
    "weight":7,
    "likes":["sleeping","purring","eating butter"]
  },
  { "type":"dog",
    "name":"Tilla",
    "weight":25,
    "likes":["sleeping","eating","walking"]
  }
]
```

- The JavaScript (**pets.html**) processing Json Data File

```
window.onload = init;

function init() {
    getPetData();
}

function getPetData() {
    var request = new XMLHttpRequest();
    request.open("GET", "pets.json");
    request.onreadystatechange = function() {
        var div = document.getElementById("pets");
        if (this.readyState == this.DONE && this.status == 200) {
            if (this.responseText != null) {
                div.innerHTML = this.responseText;
            }
            else {
                div.innerHTML = "Error: no data";
            }
        }
    };
    request.send();
}
```



[{ "type":"cat", "name":"Pickles", "weight":7, "likes": ["sleeping","purring","eating butter"] }, { "type":"dog", "name":"Tilla", "weight":25, "likes": ["sleeping","eating","walking"] }]

The XMLHttpRequest Object

- The XMLHttpRequest object is the heart of most Ajax programs
- It lets you talk to the world outside the browser.
- You use it to send requests to a web server to retrieve data.
- As is the case with some other built-in JavaScript objects likedocument, window and JSON, XMLHttpRequestis a built-in object that is available in all modern browsers.

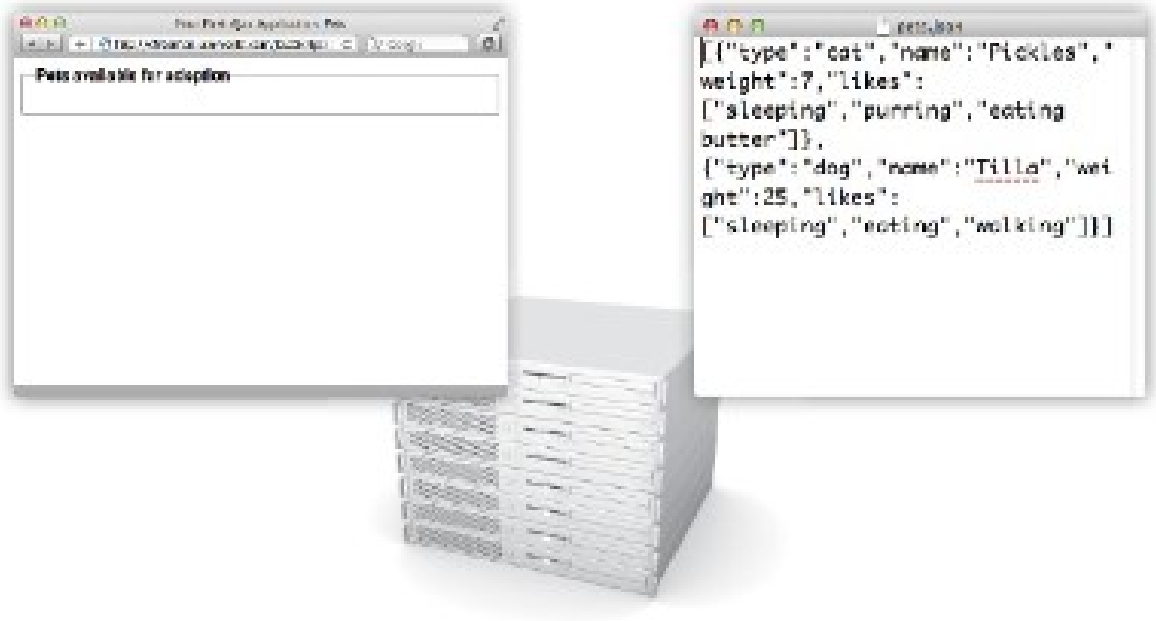
XMLHttpRequest possible values for the ready state

Number	Property	Description
0	UNSENT	open() hasn't been called yet.
1	OPENED	open() has been called.
2	HEADERS_RECEIVED	The headers have been received.
3	LOADING	The response body is being received.
4	DONE	The response is complete and ready for processing.

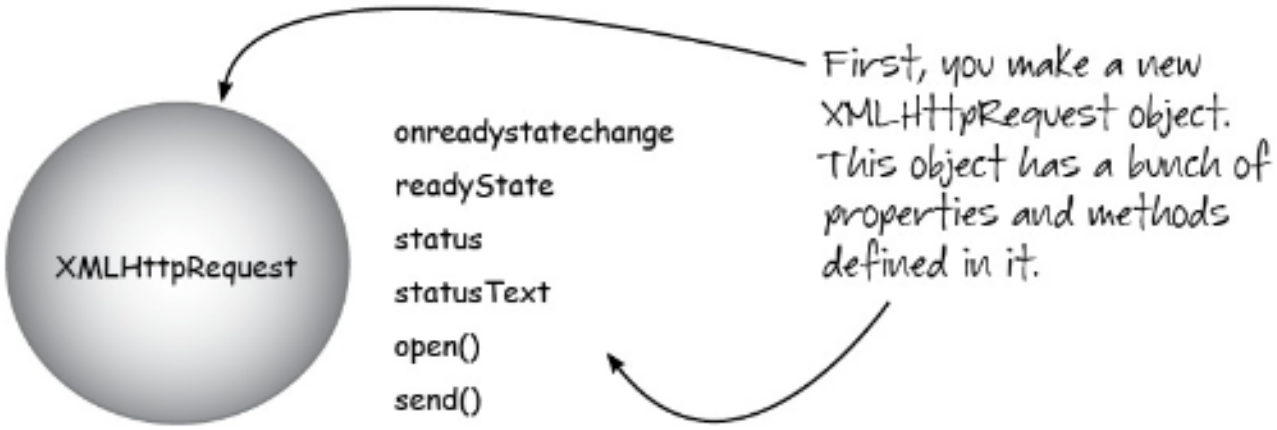
how XMLHttpRequest works?

The browser, with pets.html loaded, and your Javascript code running in it.

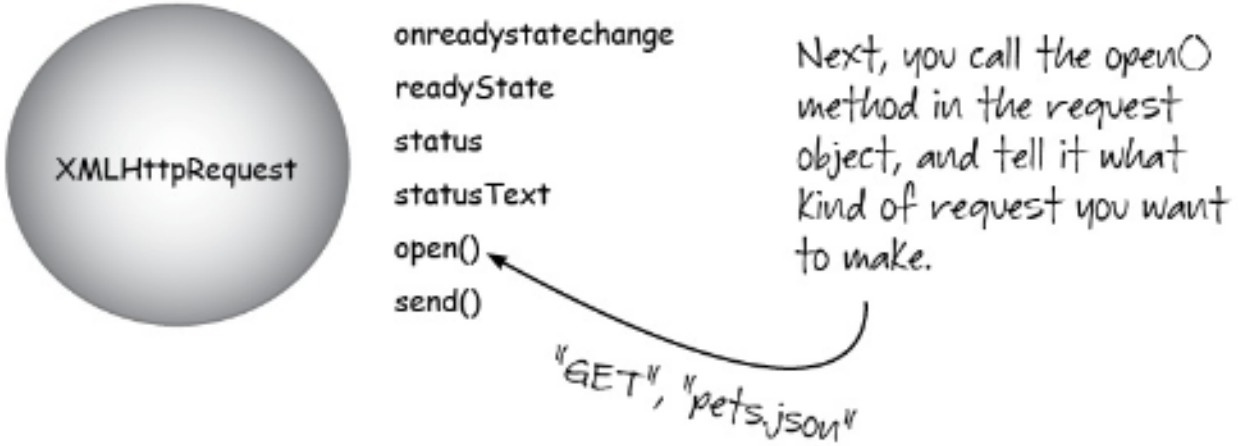
pets.json



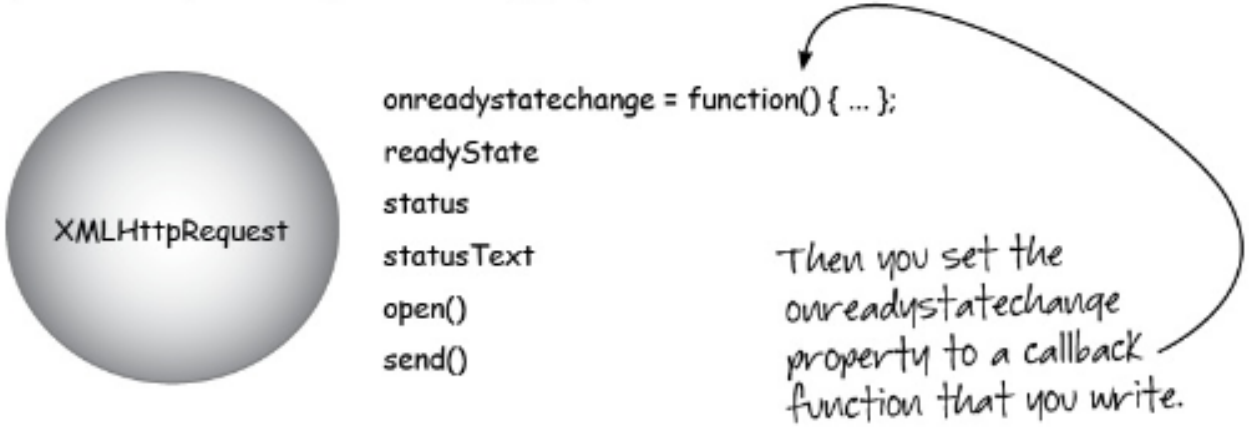
(1) var request = new XMLHttpRequest();



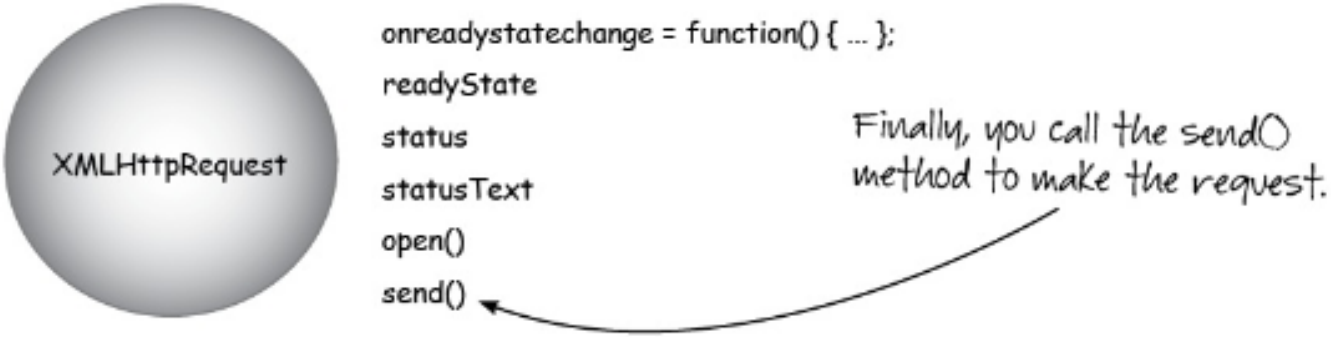
(2) request.open("GET", "pets.json");



(3) request.onreadystatechange = function() { ... };



(4) request.send();




"I need pets.json. When you're ready with your response, the browser will call my callback function."
XMLHttpRequest
[{"type":"cat","name":"Pickles","weight":7,"likes":["sleeping","purring","eating butter"]},{"type":"dog","name":"Tilla","weight":25,"likes":["sleeping","eating","walking"]}]