CSC435: Web Programming

Lecture 14: Timers, Walk the Dom Tree

Bei Xiao

American University

March 5, Tuesday, 2019

Future lecture plan

Lectures	Content	homework
March 5	OOP, Timers, DOM	Homework 4 is out
March 8	Walk the DOM tree	CP 3 is out
March 19	JS review	CP 3 due
March 22	Midterm exam	Homework 4 Due
March 26	Ajax/JSON	CP4 is out
March 30	Server side programming	Project 5 is out

Readings (Important for Exams!)

- What you need to know about JavaScript OOP:
- http://eloquentjavascript.net/1st_edition/chapter8.html
- http://javascriptissexy.com/oop-in-javascript-what-you-need-to-know/
- JavaScript Timers:
- https://www.w3schools.com/js/js_timing.asp
- JavaScript Event:
- https://eloquentjavascript.net/2nd_edition/14_event.html

Practice JS!

https://www.w3resource.com/javascript-exercises/javascript-dom-exercises.php

Setting a timer

method	description
setTimeout(*function, delayMS*);	arranges to call given function after given delay in ms
setInterval(*function, delayMS*);	arranges to call function repeatedly every *delayMS* ms
clearTimeout(*timerID*); clearInterval(*timerID*);	stops the given timer

- both `setTimeout` and `setInterval` return an ID representing the timer
  - this ID can be passed to `clearTimeout`/`Interval` later to stop the timer

Setting a timer

method	description
setTimeout(*function, delayMS*);	arranges to call given function after given delay in ms
setInterval(*function, delayMS*);	arranges to call function repeatedly every *delayMS* ms
clearTimeout(*timerID*); clearInterval(*timerID*);	stops the given timer

- both `setTimeout` and `setInterval` return an ID representing the timer
  - this ID can be passed to `clearTimeout`/`Interval` later to stop the timer

setTimeout Example

```
<button id="clickme">Click me!</button>
<span id="output"></span>
```
HTML

```
window.onload = function() {
  document.getElementById("clickme").onclick =
delayedMessage;
};

function delayedMessage() {
  document.getElementById("output").innerHTML = "Wait for
it...";
  setTimeout(sayBooyah, 5000);
}

function sayBooyah() {   // called when the timer goes off
  document.getElementById("output").innerHTML = "BOOYAH!";
}
```
JS

output Click me!

setInterval Example

```
let timer = null; // stores ID of interval timer
function delayMsg2() {
    timer = setInterval(mowgli, 1000);# put the event in
function call.
}

function mowgli() {
  document.getElementById("outputText").innerHTML +=
"Mowgli!"
}
JS
```

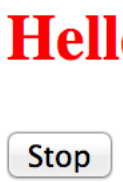
clearInterval Example

```
let timer = null; // stores ID of interval timer
function toggleDelayMessage() {
  if (timer === null) {
    timer = setInterval(mowgli, 1000);
  } else {
    clearInterval(timer);
    timer = null;
  }
}
function mowgli() {
  document.getElementById("outputText").innerHTML +=
"Mowgli!"
}
```
JS

Exercise 1: change text color

Add .Js to the following .html and use se function setInterval() and clearInterval() to automatically altering two text colors with a set interval.

The text “Hello World” will be altered between red and blue. When the user click “stop” it should stop being changed.




```
<body>
  <div id="my_box">
    <h2>Hello World</h2>
  </div>
  <button id = "stop"
>Stop</button>
</body

HTML
```

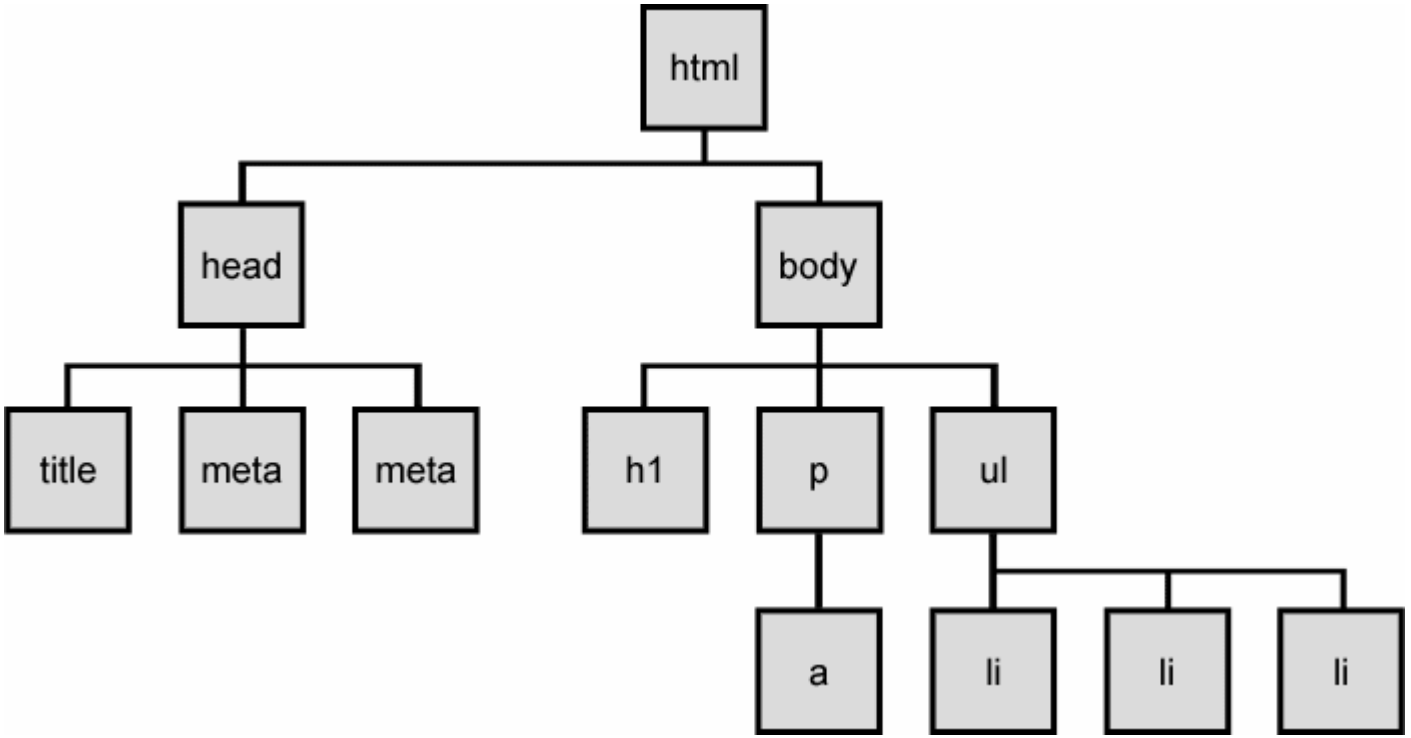
Exercise 2: Bouncing Ball

Create a page which contains an animated bouncing ball. You are given BouncingBall.html and ball.css and you will write ball.js.

The bounce area is 600px by 400px, and the ball takes up 40px by 40px of space.

- Every frame of animation (every 20ms), apply a "gravity" ot the ball and increase its downward speed by 1.
- If the ball hits the ground, make it "bounce" up to 3/4 the velocity it previously had.
- Center the ball within the ball area and use that element's width/height as boundaries.
- Optional: Make the code generic enough to work with any size ball and any size bounce area (ie., don't hard code these numbers)

The DOM tree



- The elements of a page are nested into a tree-like structure of objects the DOM has properties and methods for traversing this tree

DOM versus innerHTML hacking

- Why not just code this way?

```
function slideClick() {
  document.getElementById("main").innerHTML += "<p>A
paragraph!</p>";
}
```

- Imagine that the new node is more complex:
  - ugly: bad style on many levels (e.g. JS code embedded within HTML)
  - error-prone: must carefully distinguish " and '

```
function slideClick() {
  document.getElementById("main").innerHTML += "<p
style='color: red; " +
      "margin-left: 50px;' " + "onclick='myOnClick();'>"
+
      "A paragraph!</p>";
}
```

Creating new node

name	description
document.createElement("*tag*")	creates and returns a new empty DOM node representing an element of that type
document.createTextNode("*text*")	creates and returns a text node containing given text

```
// create a new <h2> node
var newHeading = document.createElement("h2");
newHeading.innerHTML = "This is a heading";
newHeading.style.color = "green";
```

- merely creating a element does not add it to the page
- you must add the new element as a child of an existing element on the page...

Creating new element

```
document.body.onload = addElement;
‘function addElement () { // create a new div element ‘var newDiv =
document.createElement("div"); // and give it some content var newContent =
document.createTextNode("Hi there and greetings!"); // add the text node to the newly created
div
newDiv.appendChild(newContent); // add the newly created element and its content into the
DOM
 var currentDiv = document.getElementById("div1");
document.body.insertBefore(newDiv, currentDiv);
}
```

.JS

https://developer.mozilla.org/en-US/docs/Web/API/Document/createElement

- merely creating a element does not add it to the page
- you must add the new element as a child of an existing element on the page...

Creating new element

```
document.createELEMENT

document.createTextNode

element.appendChild
```

.JS

https://developer.mozilla.org/en-US/docs/Web/API/Document/createElement

- merely creating a element does not add it to the page
- you must add the new element as a child of an existing element on the page...

Modifying the DOM tree

- Every DOM element object has these methods:

name	description
appendChild(*node*)	places given node at end of this node's child list
insertBefore(*new*, *old*)	places the given new node in this node's child list just before old child
removeChild(*node*)	removes given node from this node's child list
replaceChild(*new*, *old*)	replaces given child with new node

```
var p = document.createElement("p");
p.innerHTML = "A paragraph!";
document.getElementById("main").appendChild(p);
```

A paragraph!

Traversing the DOM tree

name(s)	description
`firstChild`, `lastChild`	start/end of this node's list of children
`childNodes`	array of all this node's children
`nextSibling`, `previousSibling`	neighboring nodes with the same parent
`parentNode`	the element that contains this node

Example: replace child

```
<ul
id="myList"><li>Coffee</li><li>Tea</li><li>Milk</li></
ul>
```

```
<script>
function myFunction() {
    var textnode = document.createTextNode("Water");
    var item =
document.getElementById("myList").childNodes[0];
    item.replaceChild(textnode, item.childNodes[0]);
}
</script>
```

Example: remove child

```
<span>
Shotting
<img id= “fallenstar” src = “star_on.gif”>
</span>
```

```
var star = document.getElementById('fallenstar');
star.parentNode.removeChild(star);
```

Example: document.createTextNode

```
<span id = “math”
 6*7 =
</span>
```

```
document.getElementByID(‘math’).appendChild(document.c
reateTextNode(42));
```

Complex DOM manipulation problems

How would we do each of the following in JavaScript code? Each involves modifying each one of a group of elements ...

- When the Go button is clicked, reposition all the divs of class puzzle to random x/y locations.
- When the user hovers over the maze boundary, turn all maze walls red.
- Change every other item in the ul list with id of TAs to have a gray background.

Selecting groups of DOM object

- methods in document and other DOM objects (* = HTML5):

name	description
getElementsByTagName	returns array of descendents with the given tag, such as "div"
getElementsByName	returns array of descendents with the given name attribute (mostly useful for accessing form controls)
querySelector *	returns the first element that would be matched by the given CSS selector string
querySelectorAll *	returns an array of all elements that would be matched by the given CSS selector string

Getting all elements of a certain type

- highlight all paragraphs in the document:

```
var allParas = document.querySelectorAll("p");
for (var i = 0; i < allParas.length; i++) {
  allParas[i].style.backgroundColor = "yellow";
}
JS
```

```
<body>
  <p>This is the first paragraph</p>
  <p>This is the second paragraph</p>
  <p>You get the idea...</p>
</body>
HTML
```

Complex Selector

- highlight all paragraphs inside the section with ID “address”:

```
//
document.getElementById("address").getElementsByTagName("
p")
var addrParas = document.querySelectorAll("#address p");
for (var i = 0; i < addrParas.length; i++) {
  addrParas[i].style.backgroundColor = "yellow";
}
```

```
<p>This won't be returned!</p>
<div id="address">
  <p>1234 Street</p>
  <p>Atlanta, GA</p>
</div>
HTML
```

Common querySlectorAll issues

- many students forget to write . or # in front of a class or id

```
// get all buttons with a class of "control"
var gameButtons = document.querySelectorAll("control");
var gameButtons = document.querySelectorAll(".control");
JS
```

- querySelectorAll returns an array, not a single element; must loop over the results (document.querySelector returns just the first element that matches, if that's what you want)

```
// set all buttons with a class of "control" to have red
text
document.querySelectorAll(".gamebutton").style.color =
"red";
var gameButtons = document.querySelector(".gamebutton");
for (var i = 0; i < gameButtons.length; i++) {
  gameButtons[i].style.color = "red";
}
```

Object: function construct with "this"

```
function  person(firstname,lastname,age,eyecolor)
{
    this.firstname=firstname;
    this.lastname=lastname;
    this.age=age;
    this.eyecolor=eyecolor;
}


// new instance
myFather=new person("John","Doe",50,"blue");
```

https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/this

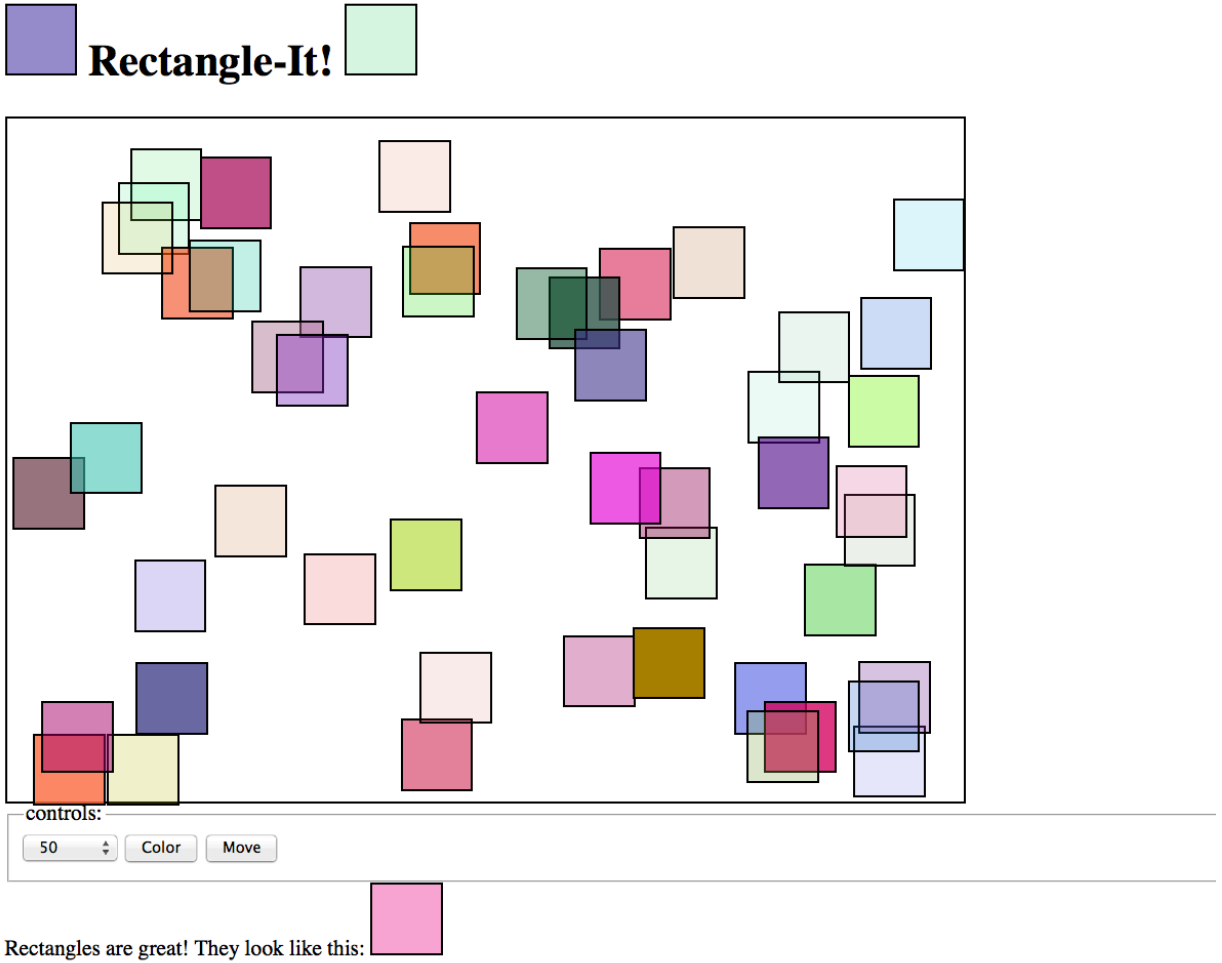
Create object using constructor function

- Constructor function to build an object. “this” refers to the function’s parent object.

```
function Employee (name, profession) {
this.name = name;
this.profession = profession;

} // Employee () is the constructor function because we use the new keyword below to invoke it.

var richard = new Employee (“Richard”, “Developer”) // richard is a new object we create from the Employee () constructor function.

console.log(richard.name); //richard

console.log(richard.profession); // Developer
```

Quiz 2

Exercise: color rectangles



Object: function construct with "this"

```
function Rabbit(adjective) {
  this.adjective = adjective;
  this.speak = function(line) {
    print("The ", this.adjective, " rabbit
says '", line, "'");
  };
}

var killerRabbit = new Rabbit("killer");
killerRabbit.speak("GRAAAAAAAAAH!");
```

http://eloquentjavascript.net/1st_edition/chapter8.html

Extending objects with prototype

Create an object:

```
function Person(name)
{
    this.name = name;
}
// when we create a new object using:
 var zack = new Person ("Zack")
```

The function person has a prototype property. We can add properties to the function:

```
Person.prototpye.kind = 'person'
//in the new object we have access to

Person.prototype.person.kind  // person
```

Example: prototype (see Demo_Extending.html)

```
// constructor function that takes two properties, and one
method showInfo
function Person(personName) {
    this.name = personName;
    this.info = "This person is called" + this.name;
    this.showInfo = function(){
        alert(this.info);
    }
}
// create two objects
var person1 = new Person ("Adam");
Var person2 = new Person ("Eve");

// we can add a further method sayHello to the person object
Definition.
  Person.prototype.sayHello = function(){
    alert(this.name + "say hello");
  }
Person1.sayHello() //"Adam say hello"
Person2.showInfo()
```

Inheriting objects

- Inheritance is the ability to create one object type from another, the new object type inherits the properties and the methods of the old, as well as optionally having further properties and the methods of its own.

- object.prototype can be used to add new methods and properties, but it can also be utilized to add all of the methods and properties of an existing constructor function to your new object.

Demo: cat and dog

1. Create a function called Pet() that can have the property of “animal” and “name”
2. Create methods setAnimal and setName. Such as

```
var myCat = new Pet();
myCat.setAnimal("cat");
myCat.setName("Garfield");
```

3. Now create a function called Dog() that has an additional method called breed.
4. Let the Dog inherit all the properties from Pet()

```
var myDog = new Dog();
myDog.setAnimal("Dog");
myDog.setName("Odie");
myDog.setBreed("Greyhound");
```

Step 1: Pet()

```
function Pet(){
    this.animal = "";
    this.name = "";
    this.setAnimal = function(newAnimal){
        this.animal = newAnimal;
    }
    this.setName = function(newName){
        this.name = newName;
    }
}
```

Step 2: cat object

```
// A Pet object has properties that contain the type of animal and the name of the pet and methods to set these values.

var myCat = new Pet();

myCat.setAnimal("cat");

myCat.setName("Garfield");


alert(" " + myCat.animal + " is called " + myCat.name);
```

Step 3: create dog and make it breed

```
//Now we create a function for Dog. We want to add a property breed and a method setBreed.

function Dog(){
    this.breed = "";
    this.setBreed = function(newBreed){
        this.breed = newBreed;
    }
}
// we want the dog has the same property as Pet, to do so we simply inherit the properties and methods of Pet.

Dog.prototype = new Pet();
```

Step 4: Make dog inherit pet()

```
// we want the dog has the same property as Pet, to do so we simply inherit the properties and methods of Pet.


Dog.prototype = new Pet();


//Now, we can access to the properties and methods of Pet in addition to those of Dog.
var myDog = new Dog();


myDog.setAnimal("Dog");
myDog.setName("Odie");
myDog.setBreed("Greyhound");
alert(" My " + myDog.animal + " " + myDog.name+" is a " + myDog.breed);
```

Now, please do the take-home exercises of OOP.

Will be used in the mid-term

Exercise 1: using prototype to extend existing JavaScript object

You can implement a String.prototype.backwards methods that returns a reversed version of any string you supply.

Here is your starter .js code:

```
<script type="text/javascript">
var inString = prompt("Enter your test string");
document.write(inString.backwards());
</script>
```

Exercise 2

- Write a constructor function for a card object with properties suit (diamonds, hearts, spades, or clubs) and face (ace, 1, 2,…king).
- Add methods to set the values of suit and face.
- Hint: alert the suitName and the faceName into the screen when you set the new values of suit and face.

Inside the constructor:

```
    this.setValues = function(suitName, faceName) {
}
```

```
var myCard1 = new Card(“space”, “ace”)
myCard.setValues (“Hearts”, “12”)
```

Exercise 3

Write a JavaScript program to get the volume of a Cylinder with four decimal places using object class.

Volume of a cylinder is V = pi r ^2 h

Hint: create a Cylinder class that takes height and radius and use prototype to add the Volume function.

//You shall create a new Cylinder object by:
var cyl = new Cylinder (7,4);

//And output:
Document.write(cyl.Volume)

num.toFixed(4)

http://www.w3schools.com/jsref/jsref_tofixed.asp