CS1520 Recitation

Week 4

Javascript 2
Array, Function, Object, DOM, and Event-Driven Programming

http://cs.pitt.edu/~jlee/cs1520

Jeongmin Lee, (jlee@cs.pitt.edu)

Today

- Array
- Function
- Object
- DOM
- Event Listener

Many examples are based on w3schools.com and tutorialspoint.com

Array

Arrays

- Array is an object with some special functionality
  - Its values can be primitive values
    or references to other objects
  - Length is dynamic

  - Create in (1) new or (2) by assigning an array literal []
  - `var myList = new Array(24, "bread", true );`
  - `var myList2 = [24, "bread", true ];`
  - `var myList3 = new Array(24);`

Functions

Function

- Return value is the parameter of return
    - If there is **no return**, or if the end of the function is reached, **undefined** is returned,
      If return has no parameter, **undefined** is returned

```
function myFunction(p1, p2) {
    return p1 * p2;
    // The function returns the product
}

var x = myFunction(4, 3);
// Function is called, return value will end
up in x
```

Object

Object

- Objects are used to **store keyed collections** of various data and more complex entities

- Each object can have properties (=keyed collections)
- It can even be functions

Object Creation

- Objects can be created with **new** or with just **{}**

```
var myComic = new Object();
//"object constructor syntax"

var myComic = {};
//"object literal syntax"
```

Object Properties

- Object can have properties (key-value storage)
- “Dot access”

```
myComic.publisher = "Image";
myComic.title = "Seven To Eternity";
```

- “Square bracket access”

```
myComic[publisher] = "Image";
myComic[title] = "Seven To Eternity";
```

- Or, it can be created with its properties:

```
var myComic = {
                   name: “A Title”,
                   published_in: 2019
              };
```

Object Properties

- Functions can be object’s property

```
function myf(num){return num*2}
myComic.fdouble = myf
myComic.fdouble(4)
```

- Another object can be an object’s property

```
mySubBook = {title: ‘smallbook’};
myComic.subbook = mySubBook;
alert(myComic.subbook.title)
```

Object Deletion

- Properties can be deleted:

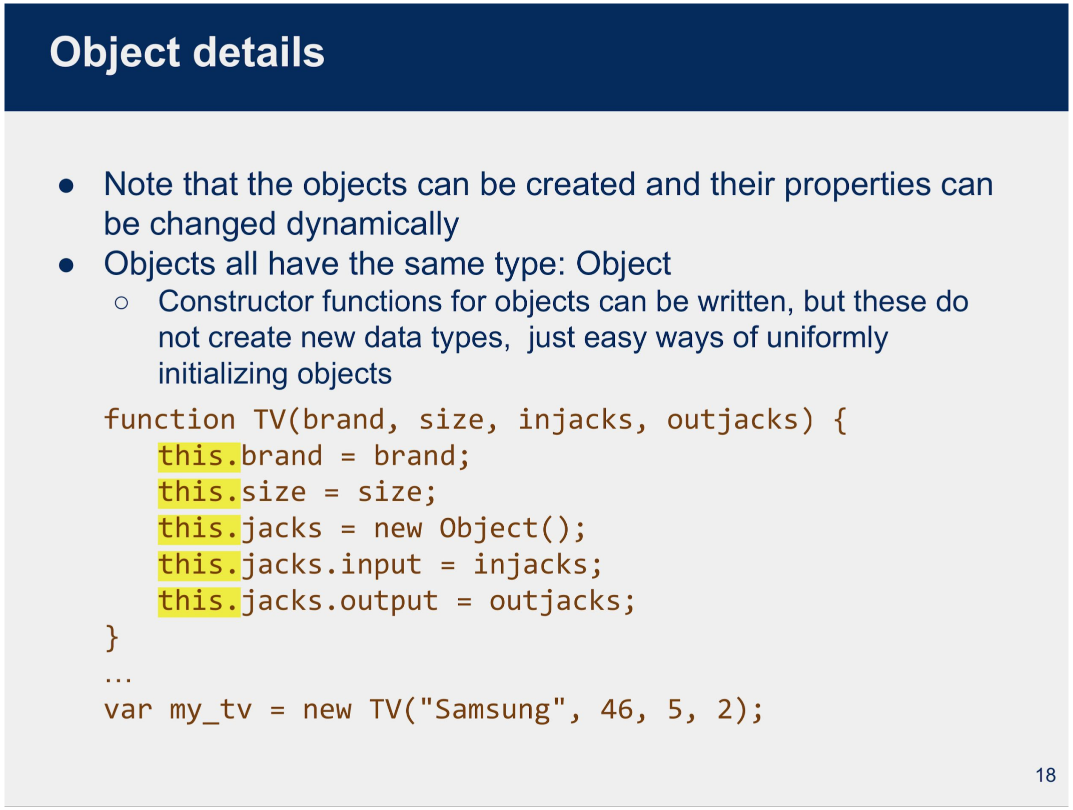
```
delete myComic.name
```

- Object itself can be deleted:

```
delete myComic
```

What is .this in object

- you might saw .this in this slide:



Object details

- Note that the objects can be created and their properties can be changed dynamically
- Objects all have the same type: Object
  - Constructor functions for objects can be written, but these do not create new data types, just easy ways of uniformly initializing objects

```
function TV(brand, size, injacks, outjacks) {
    this.brand = brand;
    this.size = size;
    this.jacks = new Object();
    this.jacks.input = injacks;
    this.jacks.output = outjacks;
}
...
var my_tv = new TV("Samsung", 46, 5, 2);
```

18

What is .this in object

- .this is the way to access a property of an object from a method in the object

```
var user = {
  name: "John",
  age: 30,

  sayHi() {
    // "this" is the
"current object"
    alert(this.name);
  }

};


user.sayHi(); // John
```

.this is not bounded

- .this can be used in any function.

```
function sayHi() {
  alert( this.name);
}
```

- .this is evaluated during the run-time.

```
var user = { name: "John" };
var admin = { name: "Admin" };

function sayHi() {
  alert( this.name );
}

// use the same function in two objects
user.f = sayHi;
admin.f = sayHi;

user.f(); // John  (this == user)
admin.f(); // Admin  (this == admin)
```

http://javascript.info/object-methods#this-in-methods

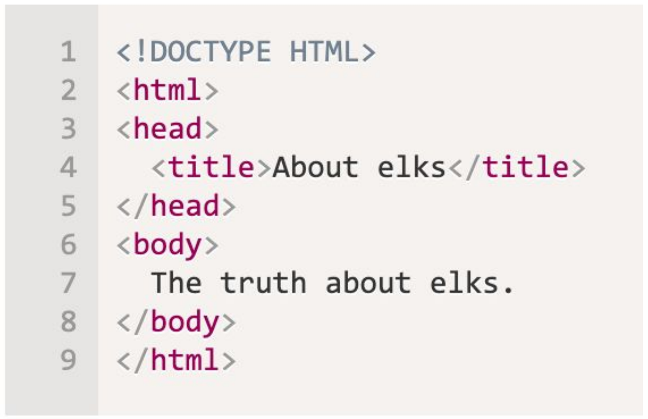
DOM :
Document Object Model

DOM

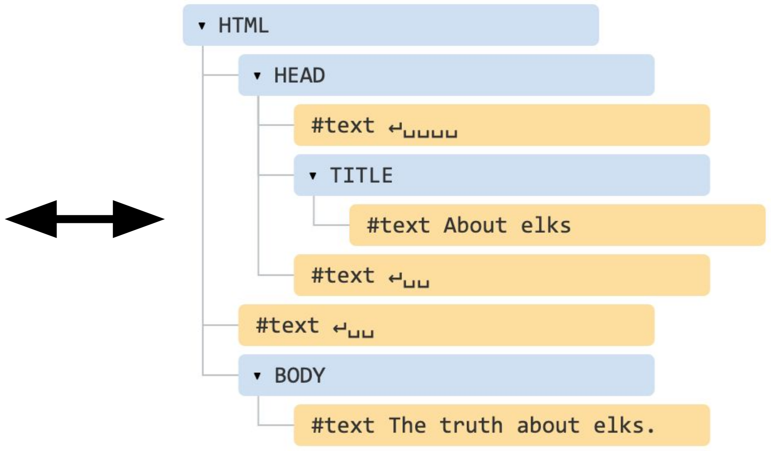
- DOM represents all HTML page contents as objects
- It can be modified (like any object in JS!!)

DOM as Tree

- Each object (element) in HTML are in Tree-like structure



```html
<!DOCTYPE HTML>
<html>
<head>
  <title>About elks</title>
</head>
<body>
  The truth about elks.
</body>
</html>
```



```
▾ HTML
  ▾ HEAD
      #text ↵␣␣␣␣
    ▾ TITLE
        #text About elks
      #text ↵␣␣
    #text ↵␣␣
  ▾ BODY
      #text The truth about elks.
```

Figure: javascript.info

DOM as Tree

- That is, each element is in a relation with other elements

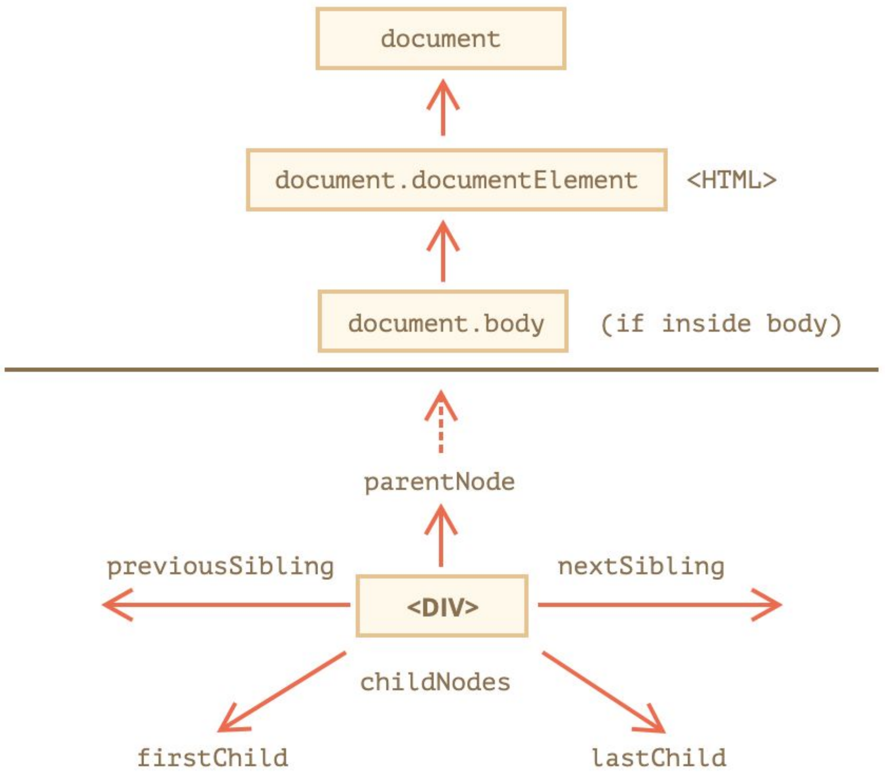


Figure: javascript.info

Selector (1): document.getElementById

- In your JS code, you can select a DOM object with this function: document.getElementById
- It is under document object. So it should be **document.**

```
<div id="elem">
  <div id="elem-content">Element</div>
</div>

<script>
  var elem = document.getElementById('elem');

  elem.style.background = 'red';
</script>
```

Selector (2): element.querySelectorAll

- More versatile method. It returns **all** elements under an element with a CSS selector as input:

```
<ul>
  <li>The</li>
  <li>test</li>
</ul>
<ul>
  <li>has</li>
  <li>passed</li>
</ul>

<script>
  let elements = document.querySelectorAll('ul >
li:last-child');

  for (let elem of elements) {
    alert(elem.innerHTML); // "test", "passed"
  }
</script>
```

DOM Node Property

- A DOM object has properties. You can access/manipulate them

- **.innerHTML** : returns contents

```
<body>
  <p>A paragraph</p>
  <div>A div</div>

  <script>
    alert( document.body.innerHTML ); // read the current contents
    document.body.innerHTML = 'The new BODY!'; // replace it
  </script>
</body>
```

DOM Node Property

- **.data** / **.nodevalue** : returns contents any type of nodes (besides html)

```
<body>
  Hello
  <!-- Comment -->
  <script>
    let text = document.body.firstChild;
    alert(text.data); // Hello

    let comment = text.nextSibling;
    alert(comment.data); // Comment
  </script>
</body>
```

DOM Node Property

- **.value** : value of <input>, <select>
- **.href** : the address for hyperlink element
- **.id :** value of “id” attribute

```
<input type="text" id="elem" value="value">

<script>
  alert(elem.type); // "text"
  alert(elem.id); // "elem"
  alert(elem.value); // value
</script>
```

Example:
DOM with Object

Example: a car object

- Let's think about we have a car with these properties and methods.
  - Property
    - Name = Fiat
    - Model = 500
    - Weight = 1000 lbs
    - Color = white
  - What it can do
    - Start
    - Stop
    - Accelerate
    - Brake

Example: a car object

- And modeling these into JS properties and methods would be something like this ... (We will see how to define soon)
-
    - Property
        - Name = Fiat
        - Model = 500
        - Weight = 1000 lbs
        - Color = white
    - What it can do
        - Start
        - Stop
        - Accelerate
        - Brake

- Property
    - car.name = ‘Fiat’
    - car.model = 500
    - car.weight = 1000
    - car.color = ‘white’
- Methods
    - car.start()
    - car.stop()
    - car.accelerate()
    - car.brake()

Example: a car object

- First create a HTML page with paragraph element with id of ‘demo’.
- At this stage, nothing would be shown on webpage but h1 title.

```
<!DOCTYPE html>
<html>
<body>

<h1> Car object demo page </h1>
<p id="demo"></p>

</body>
</html>
```

Example: a car object

- Let’s add javascript that create variable car.
- Add script block first, and then create a variable within there.

```
<!DOCTYPE html>
<html>
<body>

<p id="demo"></p>

<script>
        var car = "Fiat";
</script>

</body>
</html>
```

Example: a car object

- Now, we will make the p element to have name of car object ('Fiat').

```
<!DOCTYPE html>
<html>
<body>

<p id="demo"></p>

<script>
        var car = "Fiat";
        document.getElementById("demo").innerHTML = car;
</script>

</body>
</html>
```

Example: a car object

- Another way to make an object: with its properties.
- And let’s see its type on html page.

```
<!DOCTYPE html>
<html>
<body>

<p id="demo"></p>

<script>
        var car = {type:"Fiat", model:"500", color:"white"};
        document.getElementById("demo").innerHTML = car.type;
</script>

</body>
</html>
```

Example: a car object

- Now, let’s work on **methods**!
- First, let’s have status and speed property.

```
<!DOCTYPE html>
<html>
<body>

<p id="demo"></p>

<script>
        var car = {type:"Fiat" model:"500", color:"white",
                        status: “stop”, speed: 0};
        document.getElementById("demo").innerHTML = car.status;
</script>

</body>
</html>
```

Example: a car object

- Let’s create a method of start. It changes status.

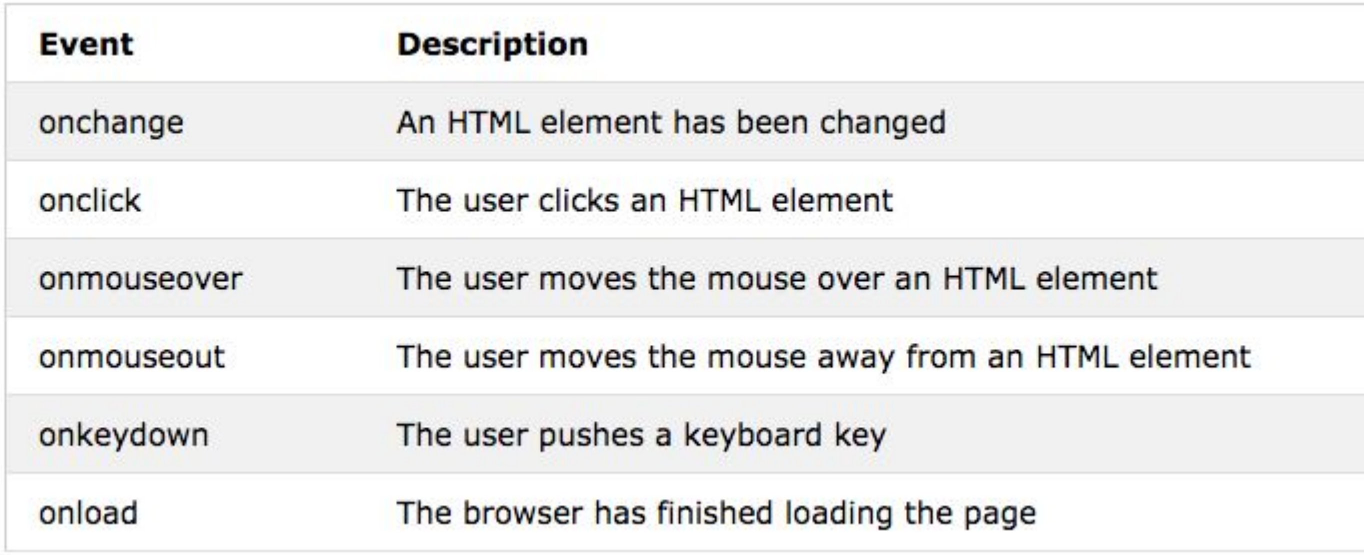
```
<!DOCTYPE html>
<html>
<body>

<p id="demo"></p>

<script>
        var car = {type:"Fiat", model:"500", color:"white",
                        status: “stop”, speed: 0, start(){this.status =”running”}};

        document.getElementById("demo").innerHTML = car.status;

        car.start();
        document.getElementById("demo").innerHTML = car.status;
</script>

</body>
</html>
```

Events

Event

- HTML events are **"things"** that happen to HTML elements.
- When JavaScript is used in HTML pages, JavaScript can **"react"** on these events.

- **Common HTML Events**



Event	Description
onchange	An HTML element has been changed
onclick	The user clicks an HTML element
onmouseover	The user moves the mouse over an HTML element
onmouseout	The user moves the mouse away from an HTML element
onkeydown	The user pushes a keyboard key
onload	The browser has finished loading the page

Event

- Example

```
<!DOCTYPE html>
<html>
<body>

<h1 onclick="changeText(this)">Click on this text!</h1>

<script>

function changeText(id) {

  id.innerHTML = "Ooops!";

}

</script>



</body>
</html>
```

Event

- Example

```
<!DOCTYPE html>
<html>
<body>

<h1 onclick="changeText(this)">Click on this text!</h1>

<script>

function changeText(id) {

  id.innerHTML = "Ooops!";

}

</script>



</body>
</html>
```

Event Handler function

The onload and onunload Events

- The **onload** and **onunload** events are triggered when the user enters or leaves the page.
- The **onload** event can be used to check the visitor's browser type and browser version, and load the proper version of the web page based on the information.
- The **onload** and **onunload** events can be used to deal with cookies.

The onload and onunload Events

```html
<!DOCTYPE html>
<html>
<body onload="checkCookies()">

<p id="demo"></p>

<script>
function checkCookies() {
  var text = "";
  if (navigator.cookieEnabled == true) {
    text = "Cookies are enabled.";
  } else {
    text = "Cookies are not enabled.";
  }
  document.getElementById("demo").innerHTML = text;
}
</script>

</body>
</html>
```

The onmouseover and onmouseout Events

- The **onmouseover** and **onmouseout** events can be used to trigger a function **when the user mouses over, or out of, an HTML element**:

```
<!DOCTYPE html>
<html>
<body>

<div onmouseover="mOver(this)" onmouseout="mOut(this)"
style="background-color:#D94A38;width:120px;height:20px;padding:40px;">
Mouse Over Me</div>

<script>
function mOver(obj) {
    obj.innerHTML = "Thank You"
}

function mOut(obj) {
    obj.innerHTML = "Mouse Over Me"
}
</script>

</body>
</html>
```

https://www.w3schools.com/js/tryit.asp?filename=tryjs_events_mouseover

Event Listener

EventListener

- **EventListener** is a function that **attached** to an **HTML element** and calls **a function** when **specified event** is triggered

```
document.getElementById("myBtn").addEventListener("click", displayDate);
```

HTML element **event** **function**

EventListener

- You can add **many event handlers** to **one element**.
- You can add many event handlers of the **same type to one element**, i.e two "click" events.
- You can add event listeners to **any DOM object** not only HTML elements. i.e the window object.

Syntax

```
element.addEventListener(event, function, useCapture);
```

- The first parameter is the **type of the event** (like "click" or "mousedown").
- The second parameter is **the function we want to call** when the event occurs.
- The third parameter is **a boolean value** specifying whether to use
  - **false: event bubbling (inner then outer)**
  - **true: event capturing (outer then inner)**.
- This parameter is optional.

Examples

- Example1: **Click event attached to a button**
- https://www.w3schools.com/js/tryit.asp?filename=tryjs_addeventlistener_add

- Example2: **Attach many listener to an object**
- https://www.w3schools.com/js/tryit.asp?filename=tryjs_addeventlistener_add_many

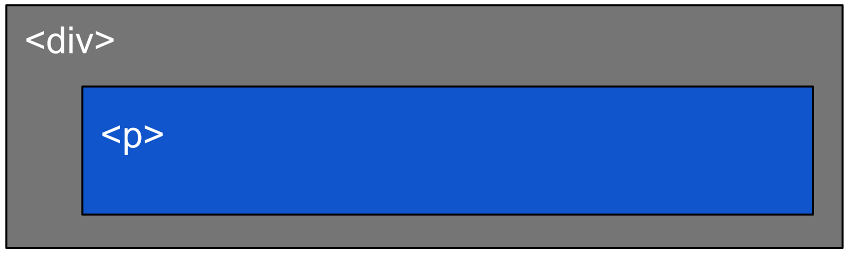
- Example3: **Attach to window object (not HTML object)**
- https://www.w3schools.com/js/tryit.asp?filename=tryjs_addeventlistener_dom

- Example4: **Passing a parameter**
- https://www.w3schools.com/js/tryit.asp?filename=tryjs_addeventlistener_parameters

Event Bubbling and Capturing

These are the way you order propagate multiple events in multiple HTML DOM.

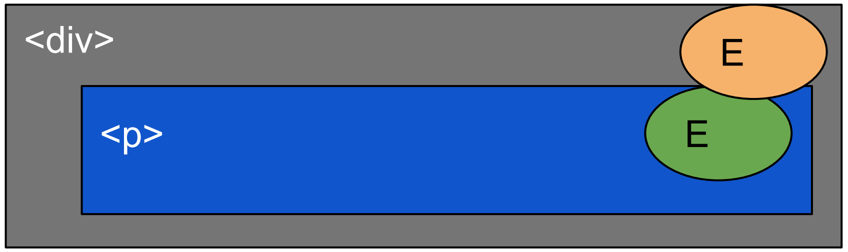
E.g: <p> inside <div>



Event Bubbling and Capturing

These are the way you order propagate multiple events in multiple HTML DOM.

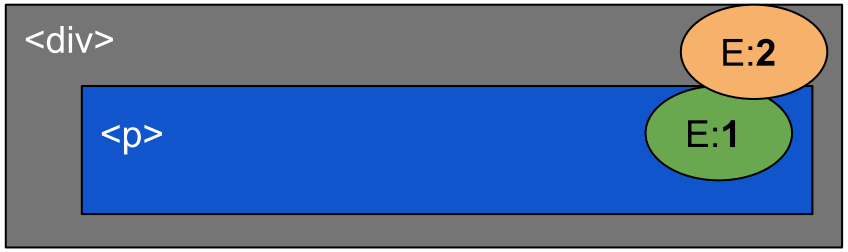
E.g: <p> inside <div> and
attached two pop-up events on each box



Event Bubbling and Capturing

Event Bubbling

the **inner most** element's event is handled **first** and then the outer



the <p> element's click event is handled first, then the <div> element's click event.

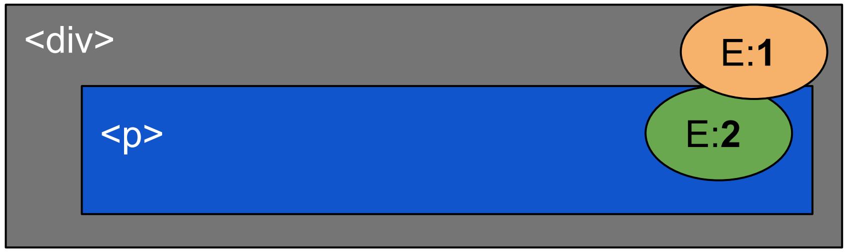
```
element.addEventListener(event, function, useCapture=false);
```

https://www.w3schools.com/js/tryit.asp?filename=tryjs_addeventlistener_usecapture

Event Bubbling and Capturing

Event Capturing

the **outer most** element's event is handled **first** and then the inner



the <div> element's click event will be handled first, then the <p> element's click event.

```
element.addEventListener(event, function, useCapture=true);
```

Questions?