

# CSC435: Web Programming

## Lecture 16: more on DOM and events

Bei Xiao

American University

March 19, 2019

# Future lecture plan

Lectures	Content	homework
March 19	JS Events, review	CP 3 Out
March 22	Midterm exam	Homework 4 Due
March 26	More on JS events, Ajax/JSON	CP3 Due
March 30	Server side programming	Project 5 is out, CP4 out

---

# 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](http://www.w3schools.com/jsref/jsref_tofixed.asp)

# Last lecture

- Creating elements:
- **`document.createElement`**
- **`document.parentNode.appendChild`**

# How to create new rectangles?

```
# this refresh the rectangle area to empty
document.getElementById("rectanglearea").innerHTML = "";
# grab the count from dropdown menu
var count = document.getElementById("count").value;

    for(var i = 0; i < count; i++) {
        # create element 'div'
        var rect = document.createElement("div");
        rect.className = "rectangle";

#append to the parent div
        document.getElementById("rectanglearea").appendChild(rect
    );
    }
```

# How to move the rectangles?

```
function moveIt() {
    var rects = document.querySelectorAll("#rectanglearea
.rectangle");
    var area = document.getElementById("rectanglearea");
    for(var i = 0; i < rects.length; i++) {
        var maxHeight =
parseInt(window.getComputedStyle(area).height) -

        parseInt(window.getComputedStyle(rects[i]).height);
        //console.log(maxHeight);
        var maxWidth =
parseInt(window.getComputedStyle(area).width) -

        parseInt(window.getComputedStyle(rects[i]).width);
        rects[i].classList.add("movable");
        rects[i].style.top = Math.floor(Math.random() *
maxHeight + 1) + "px";
        rects[i].style.left = Math.floor(Math.random() *
maxWidth + 1) + "px";
    }
}
```



# Selecting groups of DOM object

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

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

# Common querySelectorAll 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.querySelector("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.querySelector(".gamebutton").style.color =  
"red";  
var gameButtons = document.querySelector(".gamebutton");  
for (var i = 0; i < gameButtons.length; i++) {  
    gameButtons[i].style.color = "red";  
}
```

# Example: rectangle coloring

```
# select all classes named "rectangle"
var rects = document.querySelectorAll(".rectangle");
# rects will be an array

# color the rects

for(var i = 0; i < rects.length; i++) {
    var r = Math.floor(Math.random() * 256);
    var g = Math.floor(Math.random() * 256);
    var b = Math.floor(Math.random() * 256);
    var opacity = Math.random();
    rects[i].style.backgroundColor = "rgba(" + r + ",
" + g + ", " + b + ", " + opacity + ")";
}
```

# Get Elements By Tag Name

```
// Example of document.getElementsByTagName

var myList = document.getElementsByTagName("p") # //get all
p element

myList.length; //show number of elements

myList[0].style.color = 'red'; //make the first element red
```

# Problems with reading/changing styles

```
<button id="clickme">Click Me</button>
```

html

```
window.onload = function() {  
    document.getElementById("clickme").onclick =  
    biggerFont;  
};  
function biggerFont() {  
    var button = document.getElementById("clickme");  
    var size = parseInt(button.style.fontSize);  
    button.style.fontSize = (size + 4) + "pt";  
}
```

JS

- style property **lets you set any CSS style** for an element
- problem: you cannot read existing styles with it  
(you can read ones you set using the DOM .style, but not ones that are set in the CSS file)

# Accessing element's existing style

```
window.getComputedStyle(element).propertyName
```

```
function biggerFont() {  
    // turn text yellow and make it bigger  
    var clickMe = document.getElementById("clickme");  
    var size =  
    parseInt(window.getComputedStyle(clickMe).fontSize);  
    clickMe.style.fontSize = (size + 4) + "pt";  
}
```

Click Me

- `getComputedStyle` method of global window object accesses existing styles
- <https://developer.mozilla.org/en-US/docs/Web/API/Window/getComputedStyle>

# Getting/setting CSS class

```
function highlightField() {  
    // turn text yellow and make it bigger  
    var text = document.getElementById("text");  
    if (!text.className) {  
        text.className = "highlight";  
    } else if (text.className.indexOf("invalid") < 0) {  
        text.className += " highlight";    // awkward  
    }  
}
```

- JS DOM's [className](#) property corresponds to HTML class attribute
- somewhat clunky when dealing with multiple space-separated classes as one big string

# Getting/setting CSS class with `classList`

```
function highlightField() {  
    // turn text yellow and make it bigger  
    var text = document.getElementById("text");  
    if (!text.classList.contains("invalid")) {  
        text.classList.add("highlight");  
    }  
}
```

- [classList](#) collection has methods `add`, `remove`, `contains`, `toggle` to manipulate CSS classes
- similar to existing `className` DOM property, but don't have to manually split by spaces



# Exercise

- Download classlistDemoStarter.html and when the user click. “Try it”. “I am a DIV element” will have the font/color style .mystyle

# Removing a node from the page

```
function slideClick() {  
    var bullet = document.getElementById("removeme");  
    bullet.parentNode.removeChild(bullet);  
}
```

- odd idiom: *obj*.parentNode.remove(*obj*);

[http://www.w3schools.com/jsref/met\\_node\\_removechild.asp](http://www.w3schools.com/jsref/met_node_removechild.asp)

<https://developer.mozilla.org/en-US/docs/Web/API/Node/removeChild>

# The keyword this

```
this.fieldName           // access field  
this.fieldName = value;  // modify field  
this.methodName(parameters); // call method
```

JS

- all JavaScript code actually runs inside of an object
- by default, code runs in the global window object (so this === window)
  - all global variables and functions you declare become part of window
- the this keyword refers to **the current object**




# Example: dragging the rectangle by mouse

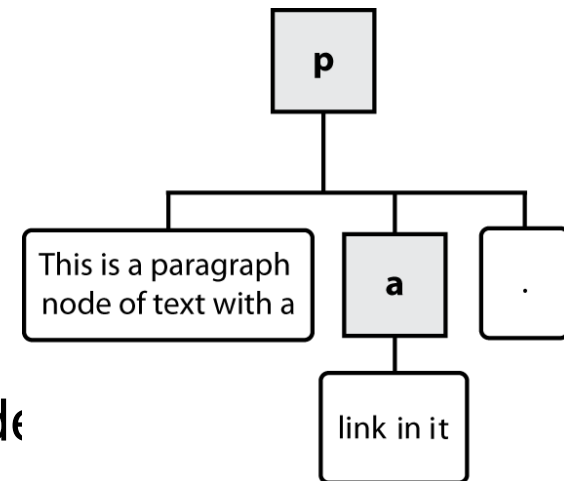
```
function mouseDown(event) {  
    this.style.zIndex = 10;  
    this.dragging = true;  
    this.prevX = event.clientX;  
    this.prevY = event.clientY;  
}
```

# Types of DOM nodes

```
<p>  
  This is a paragraph of text with a  
  <a href="/path/page.html">link in it</a>.  
</p>
```

HTML

- **element nodes** (HTML tag) 
  - can have children and/or attributes
- **text nodes** (text in a block element) 
- **attribute nodes** (attribute/value pair) 
  - text/attributes are children in an element node
  - cannot have children or attributes
  - not usually shown when drawing the DOM tree



# Traversing the DOM tree manually

- every node's DOM object has the following properties:

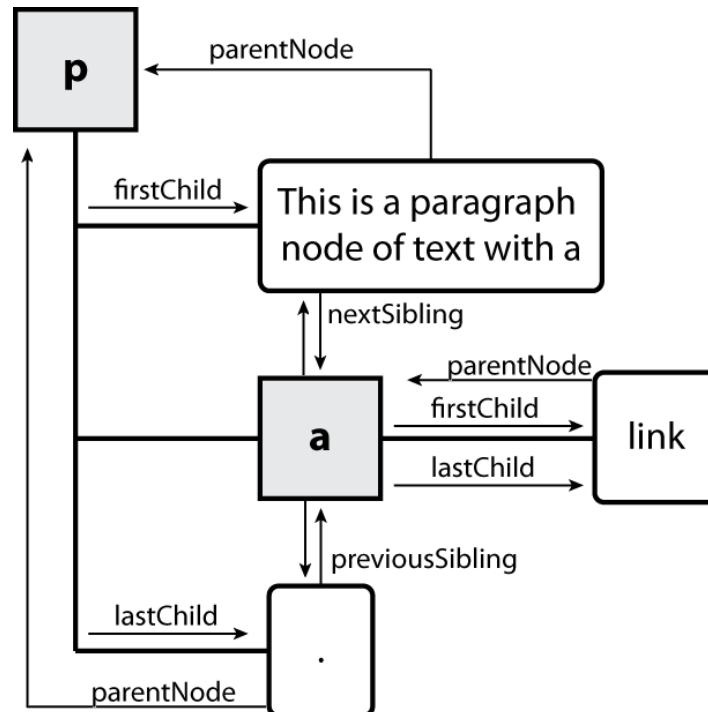
name(s)	description
firstChild, lastChild	start/end of this node's list of children
childNodes	array of all this node's children
nextSibling, previousSibling	neighboring nodes with the same parent
parentNode	the element that contains this node

- [complete list of DOM node properties](#)
- [browser incompatibility information](#)

# DOM tree traversal example

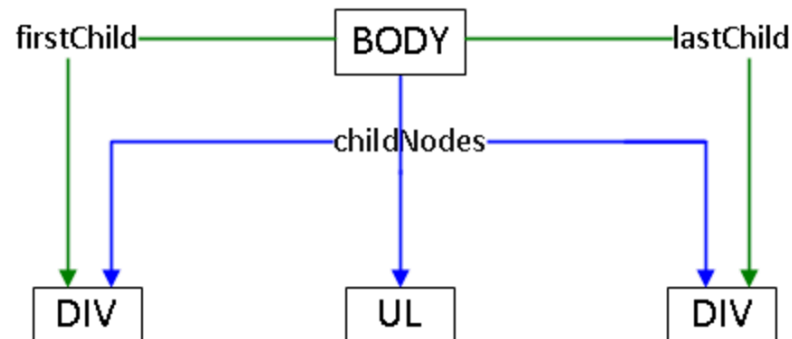
```
<p id="foo">This is a paragraph of text with a  
  <a href="/path/to/another/page.html">link</a>.</p>
```

HTML



# DOM tree traversal example

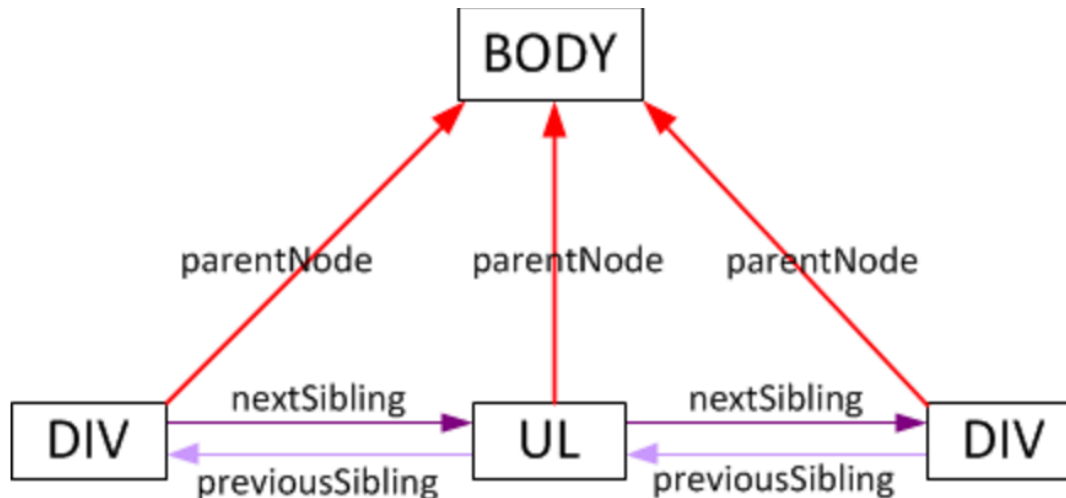
```
<html>
  <body>
    <div>Allowed readers:</div>
    <ul>
      <li>Bob</li>
      <li>Alice</li>
    </ul>
  </body>
</html>
```





# DOM tree traversal example

```
<html>
  <body>
    <div>Allowed readers:</div>
    <ul>
      <li>Bob</li>
      <li>Alice</li>
    </ul>
  </div></div>
</body>
</html>
```



# Exercise: color rectangles

Last lecture:

1. Create 1, 50, 100 rectangles
2. Color them randomly

Today's goal:

1. Random position the rectangles within the rectangle area (Hint: set maximum height and width).
2. Drag and drop by mouse. (extra)
3. Click and disappear (extra)

# Exercise: mouse over

<https://developer.mozilla.org/en-US/docs/Web/API/MouseEvent/clientX>

Event.ClientX

Event.ClientY

Shows the current mouse position

```
function mouseDown(event) {  
    this.style.zIndex = 10;  
    this.dragging = true;  
    this.prevX = event.clientX;  
    this.prevY = event.clientY;  
}
```

# JavaScript event

abort	blur	change	click	dblclick	error	focus
keydown	keypress	keyup	load	mousedown	mousemove	mouseout
mouseover	mouseup	reset	resize	select	submit	unload

- the `click` event (`onclick`) is just one of many events that can be handled

# The event object

```
function name(event) {  
    // an event handler function ...  
}
```

JS

- Event handlers can accept an optional parameter to represent the event that is occurring. Event objects have the following properties / methods:

property name	description
type	what kind of event, such as "click" or "mousedown"
target	the element on which the event occurred
timeStamp	when the event occurred

# Mouse events

<a href="#"><u>click</u></a>	user presses/releases mouse button on the element
<a href="#"><u>dblclick</u></a>	user presses/releases mouse button twice on the element
<a href="#"><u>mousedown</u></a>	user presses down mouse button on the element
<a href="#"><u>mouseup</u></a>	user releases mouse button on the element

clicking

<a href="#"><u>mouseover</u></a>	mouse cursor enters the element's box
<a href="#"><u>mouseout</u></a>	mouse cursor exits the element's box
<a href="#"><u>mousemove</u></a>	mouse cursor moves around within the element's box

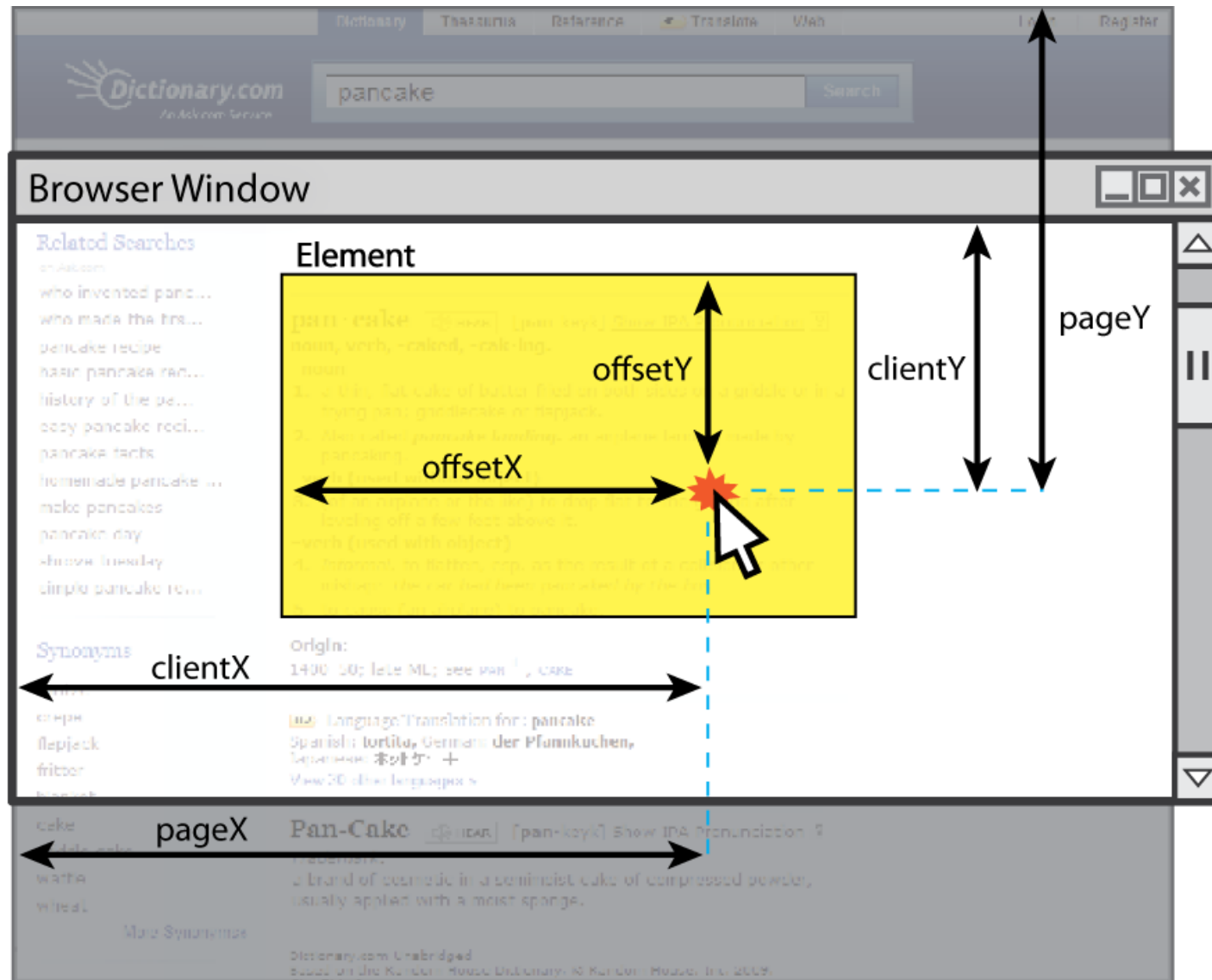
movement

# Mouse events

The `event` passed to a mouse handler has these properties:

property/method	description
<a href="#"><code>clientX</code></a> <code>clientY</code>	coordinates in <i>browser window</i>
<a href="#"><code>screenX</code></a> <code>screenY</code>	coordinates in <i>screen</i>
<code>offsetX</code> <code>offsetY</code>	coordinates in <i>element</i> (non-standard)
<code>button</code>	integer representing which button was pressed (0=Left, 1=Middle, 2=Right)

# Mouse events





# Mouse over

<https://developer.mozilla.org/en-US/docs/Web/API/MouseEvent/clientX>

Event.ClientX

Event.ClientY

Shows the current mouse position

```
function mouseDown(event) {  
    this.style.zIndex = 10;  
    this.dragging = true;  
    this.prevX = event.clientX;  
    this.prevY = event.clientY;  
}
```

# Mouse event example

```
<pre id="target">Move the mouse over me!</pre>
```

```
window.onload = function() {  
    var target = document.getElementById("target");  
    target.onmousemove = target.onmousedown = showCoords;  
};  
  
function showCoords(event) {  
    document.getElementById("target").innerHTML =  
        "screen : (" + event.screenX + ", " + event.screenY +  
        ") \n"  
        + "client : (" + event.clientX + ", " + event.clientY +  
        ") \n"  
        + "button : " + event.button;  
}
```

```
screen : (333, 782)  
client : (222, 460)  
button : 0
```

# Event Handlers

```
<button>Click me any way you want</button>
```

<script>

```
let button = document.querySelector("button");
button.addEventListener("mousedown", event => {
  if (event.button == 0) {
    console.log("Left button");      }

  else if (event.button == 1) {
    console.log("Middle button");    }

  else if (event.button == 2) {
    console.log("Right button");    }  });
```

&lt;/script&gt;

Click me any way you want

```
Left button
Left button
Left button
Left button
Left button
Left button
Right button
Right button
```

# Event object: Propagation

```
<p> A paragraph with a <button> button</button>.</p>
```

```
<script>
```

```
  let para = document.querySelector("p");  
  let button = document.querySelector("button");  
  para.addEventListener("mousedown", () => {  
    console.log("Handler for paragraph.");  });
```

```
  
  button.addEventListener("mousedown", event => {  
    console.log("Handler for button.");  
    if (event.button == 2)  
      event.stopPropagation();  
  });
```

```
</script>
```

A paragraph with a .

Handler for button.  
Handler for paragraph.  
Handler for paragraph.

If you click on the button, it will say “button” instead of “paragraph”, though button is inside the paragraph. This is called “Propagation”. If both the paragraph and the button have a handler, the more specific handler—the one on the button—gets to go first.

# Exercise Drag and Drop exercise

Click the ball and drag to move it.



Hint: use onmousedown onmouseup

```
var ball = document.getElementById("ball");
```

```
ball.onmousedown = mouseMove;
```

[https://eloquentjavascript.net/15\\_event.html](https://eloquentjavascript.net/15_event.html)

# Exercise Drag and Drop exercise

1. Catch mousedown on a draggable element.
2. Prepare the element to moving (maybe create a copy of it or whatever).
3. Then on mousemove move it by changing left/top and position:absolute.
4. On mouseup (button release) – perform all actions related to a finished Drag'n'Drop.

This exercises is explained very well here. But don't read it before you try it out yourself.

<https://javascript.info/mouse-drag-and-drop>

# Keyboard and text events

| name                            | description   |
|---------------------------------|---|
| <a href="#"><u>focus</u></a>    | this element gains keyboard <b>focus</b> (attention of user's keyboard) |
| <a href="#"><u>blur</u></a>     | this element loses keyboard focus                                       |
| <a href="#"><u>keydown</u></a>  | user presses a key while this element has keyboard focus                |
| <a href="#"><u>keyup</u></a>    | user releases a key while this element has keyboard focus               |
| <a href="#"><u>keypress</u></a> | user presses and releases a key while this element has keyboard focus   |
| <a href="#"><u>select</u></a>   | this element's text is selected or deselected                           |

# Key event objects

| property name             | description  |
|---------------------------|--|
| keyCode                   | ASCII integer value of key that was pressed<br>(convert to char with <a href="#">String.fromCharCode</a> ) |
| altKey, ctrlKey, shiftKey | true if Alt/Ctrl/Shift key is being held   |

- issue: if the event you attach your listener to doesn't have the focus, you won't hear the event
  - possible solution: attach key listener to entire page body, `document`, an outer element, etc.



# Key event example

```
document.getElementById("textbox").onkeydown =  
textKeyDown;  
...  
function textKeyDown(event) {  
    var key = String.fromCharCode(event.keyCode);  
    if (key == 's' && event.altKey) {  
        alert("Save the document!");  
        this.value = this.value.split("").join("-");  
    }  
}
```

JS

- each time you push down any key, even a modifier such as Alt or Ctrl, the **keydown** event fires
- if you hold down the key, the **keydown** event fires repeatedly
- **keypress** event is a bit flakier and inconsistent across browsers

# Useful Keycodes

| Keyboard Key                  | Event Keycode  |
|-------------------------------|----------------|
| Backspace                     | 8              |
| Tab                           | 9              |
| Enter                         | 13             |
| Escape                        | 27             |
| Page Up, Page Down, End, Home | 33, 34, 35, 36 |
| Left, Up, Right, Down         | 37, 38, 39, 40 |
| Insert, Delete                | 45, 46         |
| Window/Command                | 91             |
| F1-F12                        | 112-123        |

# Page/Window Events

| Name                               | Description   |
|------------------------------------|---|
| <a href="#"><u>contextmenu</u></a> | the user right-clicks to pop up a context menu                    |
| <a href="#"><u>error</u></a>       | an error occurs when loading a document or an image               |
| <a href="#"><u>load</u></a>        | the browser loads the page  |
| <a href="#"><u>resize</u></a>      | the browser window is resized                                     |
| <a href="#"><u>scroll</u></a>      | the user scrolls the viewable part of the page up/down/left/right |
| <a href="#"><u>unload</u></a>      | the browser exits/leaves the page                                 |

# Stopping an event

| event method name                      | description   |
|--|---|
| <a href="#"><u>preventDefault</u></a>  | stops the browser from doing its normal action on an event; for example, stops the browser from following a link when <a> tag is clicked, or stops browser from submitting a form when submit button is clicked |
| <a href="#"><u>stopPropagation</u></a> | stops the browser from showing this event to any other objects that may be listening for it   |

# Stopping an event

```
<form id="exampleform"
action="http://foo.com/foo.php">...</form>
```

```
window.onload = function() {
    var form = document.getElementById("exampleform");
    form.onsubmit = checkData;
};

function checkData(event) {
    if (document.getElementById("state").length != 2)
    {
        alert("Error, invalid city/state."); // show
error message
        event.preventDefault();
        return false; // stop form
submission
    }
}
```

# Take home read

Mouse event:

<https://developer.mozilla.org/en-US/docs/Web/API/MouseEvent>

JavaScript Document:

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

Introduction to OOP in JavaScript

[https://developer.mozilla.org/en-US/docs/Web/JavaScript/Introduction to Object-Oriented JavaScript](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Introduction_to_Object-Oriented_JavaScript)

# Setting a timer

method	description
<code>setTimeout(<i>function</i>, <i>delayMS</i>);</code>	arranges to call given function after given delay in ms
<code>setInterval(<i>function</i>, <i>delayMS</i>);</code>	arranges to call function repeatedly every <i>delayMS</i> ms
<code>clearTimeout(<i>timerID</i>);</code> <code>clearInterval(<i>timerID</i>);</code>	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
<code>setTimeout(<i>function</i>, <i>delayMS</i>);</code>	arranges to call given function after given delay in ms
<code>setInterval(<i>function</i>, <i>delayMS</i>);</code>	arranges to call function repeatedly every <i>delayMS</i> ms
<code>clearTimeout(<i>timerID</i>);</code> <code>clearInterval(<i>timerID</i>);</code>	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





# setInterval

```
var timer = null; // stores ID of interval timer

function delayMsg2() {
    if (timer === null) {
        timer = setInterval(rudy, 1000);
    } else {
        clearInterval(timer);
        timer = null;
    }
}

function rudy() { // called each time the timer goes off
    document.getElementById("output").innerHTML += " Rudy!";
}
```

JS

Click me!

output