JAVASCRIPT

Introduction to programming

Lecture 9: Jan. 25 2016

Interactivity

The Layers of Web Design

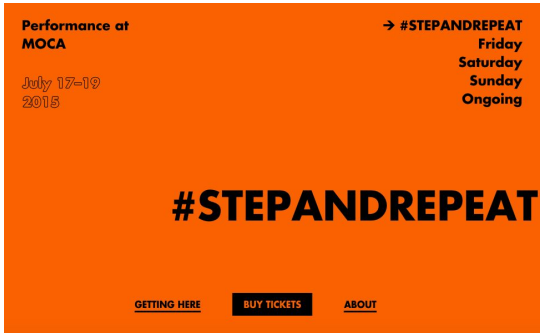
Content: HTML

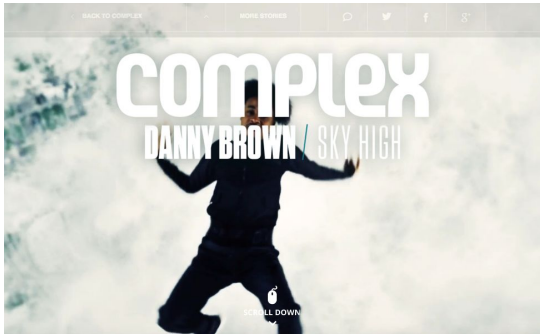
Form: CSS

Behavior: Javascript

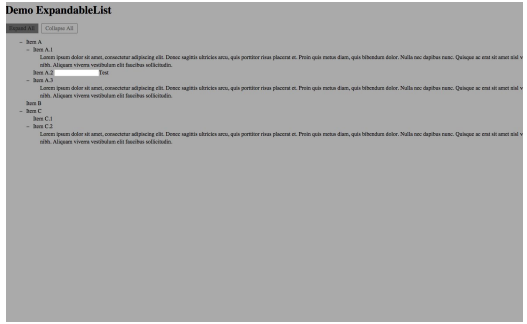
By using the way a browser recognizes user behavior, web designers can build functionality that accommodates and responds to how a user uses the site.

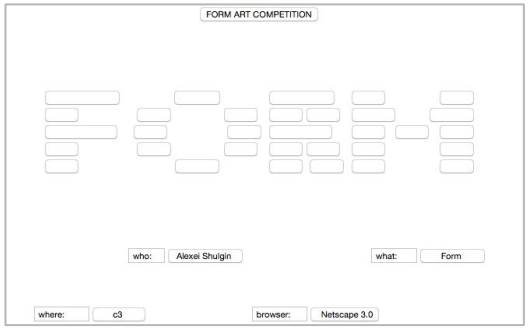
Using Scroll

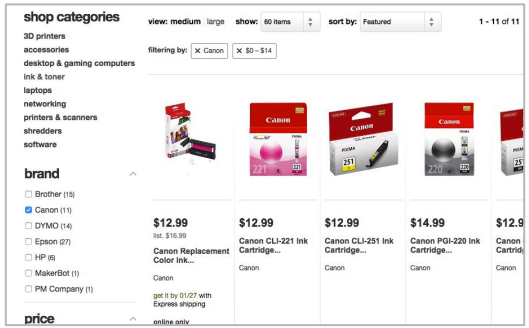




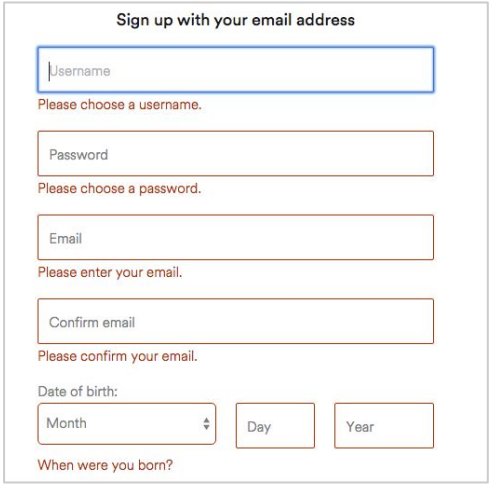
Using Mouse Behaviors (Pointing, Clicking)

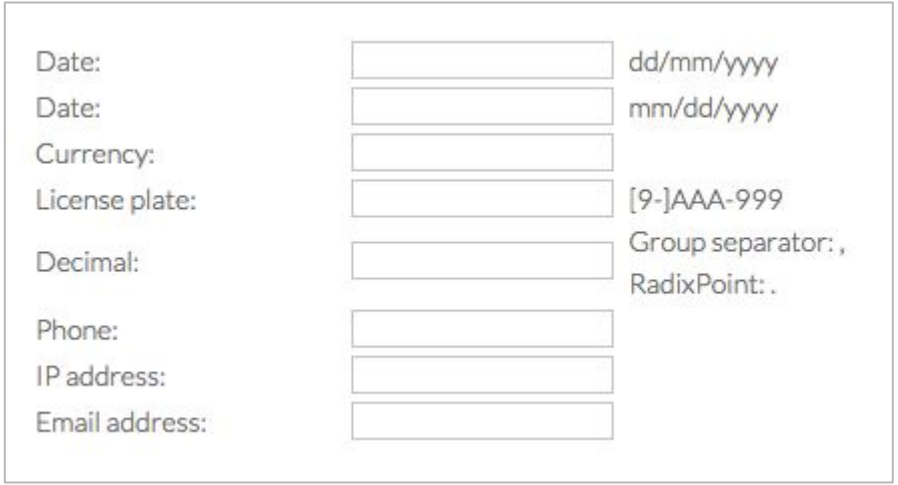






Form Validation and Input Masking





Scripting & Programming

Step 1

Step 2

Step 3

Step 1:

- Step 1a
- Step 1b

Step 1a:

- Step 1
- Step 2
- Step 3
- Step 4
- Step 5

Step 2:

- Step 2a
- Step 2b

Step 3:

- Step 3a
- Step 3b

On the most basic level, programming is a way of providing instructions to carry out a task. There are many ways to give instructions, some more efficient or helpful than others.

When programming, the primary move is to **decompose your problem:** break down the task into a series of steps. First, think generally and categorize the main steps required. Then, further break down each into logical sequences.

Breaking it down

```
Task: Hide circle

1. Create a circle with a diameter of 50px
in the center of the screen

     1a) Create a circle 50px wide

     1b) Center circle in the viewport

2.  When the user clicks on the circle, hide
it from view

     2a) Detect mouse click on circle

     2b) Hide circle
```

It’s useful to first write your program’s steps in English. This will allow you to clearly define your goal and plan out your steps and sub-steps before diving into the actual code. For complex functions, sketching out flowcharts of the decision tree may also be helpful.

Javascript

```
<script>
     function count_rabbits() {
          for(var i=1; i<=3; i++) {
            // operator + concatenates strings
             alert("Rabbit "+i+" out of the hat!")
          }
     }
</script>
```

Using `<script></script>` tags, JavaScript can be placed anywhere in an HTML document. Javascript code is read by the **syntax parser** (compiler / interpereter) top to bottom.

Javascript is a programming language used mainly for **client side** (i.e. in the browser) behaviors. Its main job is to add interactivity to a page by manipulating the **DOM** (Document Object Model.)

It has the ability to:

- **Access** the content of the page
- **Modify** the content of the page
- **Program** rules or instructions the browser can follow
- **React** to events triggered by the user or browser

Note: Javascript ≠ Java

Implementing Javascript

Your webpage can incorporate javascript in three ways.

1. Within the <head>

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8">
        <title>Page Title</title>

        <script>
        function count_rabbits() {
            for(var i=1; i<=3; i++) {
                // operator + concatenates strings
                alert("Rabbit "+i+" out of the hat!")
            }
        }
        </script>

    </head>
    <body>
```

2. Within the <head>, linking to an external .js file

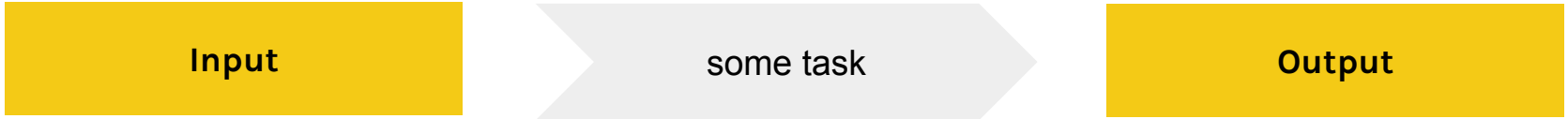
```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8">
        <title>Page Title</title>
        <script src="js/rabbits.js"></script>
    </head>
    <body>
```

3. Within the <body>

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8">
        <title>Page Title</title>
    </head>
    <body>
        <h1>Javascript Demo: Rabbits</h1>

        <script>
            function count_rabbits() {
                for(var i=1; i<=3; i++) {
                    // operator + concatenates strings
                    alert("Rabbit "+i+" out of the hat!")
                }
            }
        </script>
    </body>
</html>
```

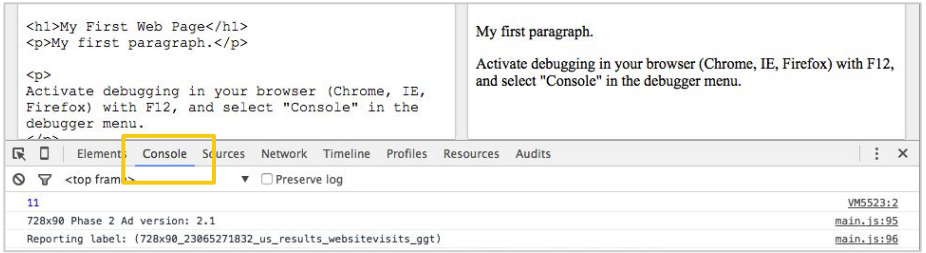
Output



When you instruct Javascript to perform a task, it will give you some output. However, it will not automatically display it anywhere. You can verify your result in one of the following ways:

- Writing into an alert box, using `window.alert()`
- Writing into the HTML output using `document.write()`
- Writing into an HTML element, using `innerHTML`
- Writing into the browser console, using `console.log()`

The Console



It is a good practice to print outputs in **the console**, accessible via your browser's Developer Tools. This will allow you to see your outputs without modifying document elements.

```
console.log(stuff);
```

method applied to the console object to print out your argument

what you are trying to display in the console

Components of Javascript

Introduction to key concepts

Statements

A script is a series of instructions that a computer can follow one-by-one. Each individual instruction or step is known as a **statement**. Statements should end with a semicolon (;) Statements are composed of values, operators, expressions, keywords, and comments.

```
log (cereal);
return ( "bicycle lane" );
alert ( "Happy Birthday" + outcome);
```

Syntax

type	description	example

values	**Fixed values** are called literals, such as numbers or texts (strings). **Variable values** are called variables.	`"Hello!"`
		`23`
		`count`

operators	**Assignment operators** are used to assign values to variables. **Arithmetic operators** are used to compute values. See list.	`var x = 5;` `(5 + 6) * 9;`

expression	An expression is a combination of values, variables, and operators, which computes to a value. The computation is called an **evaluation**.	`(5 + 6) * 9;`
		`"Web" + " " + "Design";`
		`x * 10;`

keywords	**Keywords** are used to identify actions to be performed.	`var x = 5;`

Variables

Variables are containers for information. They can store pretty much anything: numbers, text, html elements, etc.

```
var rabbitCount = 6;
```

semicolon
finishes your statement

var keyword
indicates that you are declaring a variable

variable name
name you give to the variable

value
the information to be stored in the variable

assignment operator
assigns the value (does NOT mean “equals”)

Naming variables

- The name must begin with a letter, dollar sign ($), or an underscore (_). It must NOT start with a number.
- The name can contain letters, numbers, dollar sign ($), or an underscore (_). Note that you must not use a space, dash(-) or a period (.) in a variable name.
- You cannot use keywords or reserved words.
- All variables are case sensitive
- Use a name that describes the kind of information that the variable stores
- If the variable name needs more than one word, it is usually written with the first word in lowercase, then all subsequent words capitalized (i.e. firstName, lastName). This practice is called Camel Case.

Basic Data Types

type	description	example

`string`	The string data type consists of letters and other characters. Strings must be enclosed in quotation marks. These can be single or double quotes, but the opening quote must match the closing quote.	`"Hello!"`
		`"50 cents"`
		`"Why?"`

`numeric`	The numeric data type handles numbers. In addition to calculating, numbers can be used for tasks such as determining the size of the screen, moving the position of an element on a page, or setting the amount of time an element takes to fade in.	`0.75`
		`-2500`
		`4`

`boolean`	Booleans can be either `true` or `false`. They can act as on/off switches for determining which part of a script should run	`true` `false`

Operators

```
(5 + 6) * 9;
```

```
99
```

arithmetic operators

+	Addition
-	Subtraction
*	Multiplication
/	Division
%	Modulus
++	Increment
--	Decrement

```
"Web" + " " + "Design";
```

```
Web Design
```

string operator

When used on strings, the + operator is called the concatenation operator.

Arrays

An **array** is a special type of variable that contains a group of values. This can be a list of anything (names, numbers, other variables, etc…) While an array can store all types of information, it's generally a good practice to store just one type of data in a single array.

```
var carNames = [
  "Civic",
  "Prius",
  "Wrangler",
  "Cayenne",
  "GranTurismo"];
```

var keyword, name, assignment

square brackets contain the list of items

commas separate each item

semicolon

Accessing Array Items

The position of an item in an array is called an **index.** Think of it as a numerical label. The position number starts at **0** (not 1.)

```
                    0          1          2          3          4
var carNames = ["Civic", "Prius","Wrangler","Cayenne", "GranTurismo"];
```

The index is useful to access specific items in your array. For example, you can access the third item in your list as follows:.

```
carNames[2] //This will return Wrangler.
```

You can also access the number of items an array contains through its length property.

```
carNames.length //This will return 5.
```

Functions

A **function** defines a set of actions to be performed. Every time a function is called, a new **execution context** is created and stacked on the previously created context.

function keyword
indicates that you are declaring a function

function name
name you give to your function

parentheses
lists any parameters (inputs) to your function.

curly brackets
contains the contents of the function

```
function sayHello() {
  document.write("Hello");
}
```

statements
compose the function

Conditionals

Conditionals allow you to declare different actions based on different conditions. You can use simple If/Then/Else logic to define your scenarios.

if keyword

indicates that you are declaring a conditional

condition

defines the rule fulfilled in the scenario

parentheses

contains the condition

curly brackets

contains the actions

```
if (rabbitCount < 4) {
  document.write("Less than 4");
}
```

action

executed when the specified condition is met

Comparison operators

Conditions are usually evaluated by comparing two values. The expression returns a boolean value, true or false.

syntax	description
`==`	checks that the values are the same
`===`	checks that both the data type and value are equal (strict)
`!==`	checks that the two values are NOT the same
`!===`	checks that both the data type and value are NOT equal (strict)
`< / >`	checks that the number on the left is less than / greater than the right
`<= / >=`	checks that the number on the left is less than or equal to / greater than or equal to the right

Objects

A collection of name value pairs

Objects

Objects are a collection of name-value pairs, such as variables and functions. Ideally, the components of an object are put together in such a way that the object represents both the attributes and behavior of some “thing” being modeled in the program. Each object can have its own **properties**, **methods**, and **events.**

Example Object: **Car**

Properties

Properties are variables belonging to an object. Think “characteristics.” Each property has a **name** and **value**. A value for a name may contain other name value pairs.

Car Properties

name · · · · make: MINI · · · · · · · · · · · · · · value

model: Cooper S

horsepower: 189 hp

color: orange

currentSpeed: 30mph

fuelLevel: 100

Methods

Methods are functions that belong to an object. Think of things people need to do with objects. Methods can tell you something about information stored in properties or change the value of properties.

Car Methods

- changeSpeed()
  a function that changes the value of the property currentSpeed
- hasFuel()
  a function that checks the value of the property fuelLevel

Events

In the same way cars are designed to respond differently when a driver interacts with the two pedals, websites are designed to react accordingly with a user interaction.

Car Events

- brake
  happens when a driver slows down
- accelerate
  happens when driver speeds up

Objects + Events + Methods + Properties

Events trigger methods → Methods retrieve or update an object's properties

Object: Car

1	**Event**	happens when:	method called:	**Properties**		
	`accelerate`	driver presses right pedal	`changeSpeed()`	make:	MINI	
2	**Method**	what it does:		`currentSpeed:`	45mph	3
	`changeSpeed()`	increases or decreases value of currentSpeed property		color:	orange	

Object Syntax

object name

curly brackets, using Object-Literal syntax

object properties

name value pairs

object method

```
var car = {
    make: "MINI",
    model: "Cooper S",
    horsepower: 189,
    color: "Orange",
    currentSpeed: 30,
    fuelLevel: 100,
    changeSpeed: function() {
        console.log(this.currentSpeed + 40 + "mph");
    }
};
```

Accessing Object Properties

There two different ways in which you can access the properties of an object.

```
someObj.propName;
```

dot notation

```
someObj["propName"];
```

bracket notation

unlike dot notation, you can also use variables that have the value of the desired property name

Objects and methods

By **calling the method** of an object, you can use objects and their methods.

method

```
document.write("Hello from the other side");
```

object

member operator

methods and properties of an object are called members, and are accessed with the dot

parameters

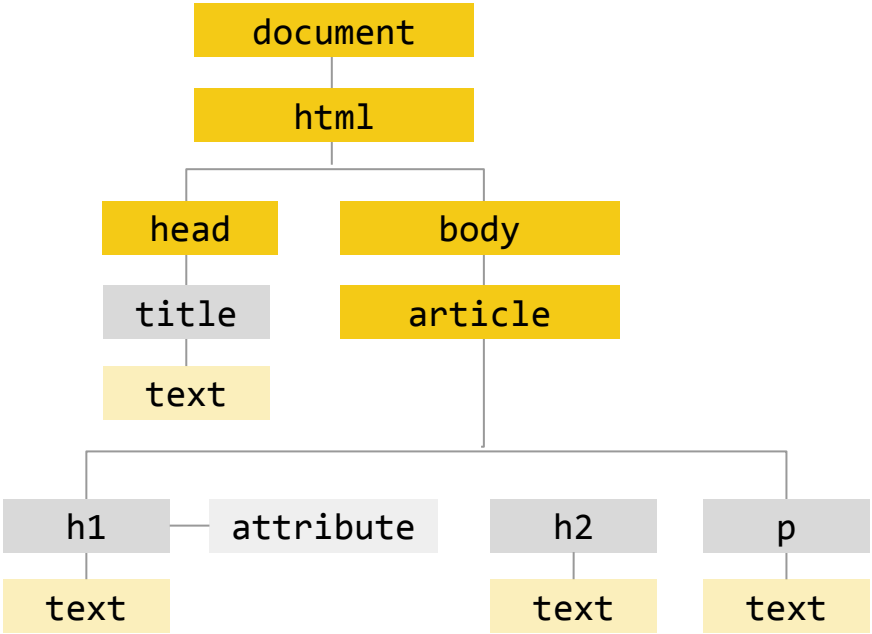
information required by the method are provided in parentheses

The Document Object

The most important document in javascript is the **Document Object**. It represents an HTML web page.

- **Properties** describe characteristics of the current web page (such as the title of the page).
- **Methods** perform tasks associated with the document currently loaded in the browser (such as getting information from a specified element or adding new content).
- **Events**, such as a user clicking or tapping on an element, can be used as triggers for methods

Document Object Model



As a browser loads a web page, it creates a model of that page. Not only does it create a document object, but it also creates an object for each element on the page. Together, the objects compose what is called a DOM tree, and it is stored in the browser's memory. With Javascript, you can access and manipulate any of the DOM tree elements.

Working with the DOM

The Javascript method

Ways to Access the DOM

The following **methods** are ways to access **nodes** of the document object model.

	method	description
selects individual elements	`getElementbyID('id')`	selects an individual element given the value of its `id` attribute
	`querySelector('css selector')`	uses CSS selector syntax, returning **only the first** of the matching elements
returns more multiple elements in a **NodeList**	`getElementsByClassName('class')`	selects one or more elements given the value of their `class` attribute
	`getElementsByTagName('tagname')`	selects all elements on the page with the specified tag name
	`querySelectorAll('css selector')`	uses CSS selector syntax to select one or more elements and returns all those that match

Accessing and Updating Element Content

The following are examples of properties are useful for getting or setting the content of elements.

property	description
`innerHTML`	allows you to retrieve / replace text & markup within the element.
`textContent`	allows you to retrieve / replace just the text that is in the containing element and its children.

```
var element = document.getElementById("demo").innerHTML;
```

```
document.getElementById("demo").innerHTML = "some html text";
```

Adding Elements to the DOM (without innerHTML)

1. Create the element

   `var newListItem document.createElement("li");`

2. Give it content

   `var newText document.createTextNode("some text");` or
   `newListItem.innerHTML = "some text";`

3. Add it to the Dom

   `document.getElementById("targetList").appendChild(newListItem);`

Events

Triggers for your scripts

Events

Events are actions that happen on a webpage. See a list of DOM events.

event	description
`change`	an HTML Element has been changed
`click`	user clicks an HTML element
`mouseover`	user moves mouse over HTML element
`mouseout`	user moves mouse away from an HTML element
`keydown`	user pushes a keyboard key
`load`	browser finishes loading page

How Events trigger Javascript Code

1. Select the element node(s) to which the script should respond
2. Indicate which event on the selected node will trigger the response
3. Code the function that defines the response

This is achieved with either **event handlers** or **event listeners**.

Event Handlers

An **event handler** binds an event to an element and provides a set of instructions about how that particular event should be dealt with. It indicates the reaction to the event. Event handlers can only handle a single function to be bound to an event.

```
document.onclick = myFunction;
```

element

dot

on precedes event name

event

the response to the event

Event Listeners

An **event listener** allows multiple function to be bound to an event (but are not supported with older browsers.)

```
document.addEventListener('click', myFunction, false);
```

element

dot

addEventListener method

event

the response to the event

name of function

set to false (related to the way an event flows)

Loops

Repeating tasks

Loops

Loops allow you to run the same code over and over again, each time with a different value. There are two types of loops, “for” and “while.” Loops are useful especially when working with arrays, when you want to run the same task with each item in the array. **Note:** Be sure to set a condition to **break** your loop — otherwise, you will crash your browser.

```
for (var i = 0; i < 10; i++) {
     /**execute this code each time**/

}

while (n < 3) {
     /**execute this code each time**/
}
```

For Loop

Loops allow you to run the same code over and over again, each time with a different value. For loops allow you to run a function by specifying the number of iterations you want it to repeat; it uses a **counter** as its condition to run.

for keyword

initialization
create and set the counter variable i

condition
the loop continues until the counter reaches this number

update
at each pass of the loop, the counter adds 1 (`i = i+1);`

```
for(var i = 0; i < 10; i++){
   console.log("Hello");
}
```

brackets

code to execute during loop

While Loop

While loops are useful when you want to repeat a function but don't know beforehand the number of times to loop the task.

```
var n = 0;
while(n < someArray.length){
    console.log(someArray[n]);
    n++;
}
```

initialization: create and set the counter variable n

condition: the loop continues until the counter reaches this number

while keyword

brackets

code to execute during loop

update
at each pass of the loop, the counter adds 1 (`n = n+1;`)

Commenting

Like in HTML and CSS, we can insert comments in javascript to help explain what your code does. This makes it easier for you and others to read the code.

/*	*/	for more than one line of code
//		for a single line of code

The importance of contexts

Lexical environment, or where the code is written, is important as it determines the scope of your javascript variables and functions.

Multiple lexical environments are managed by the **execution context**, which is a wrapper to help manage the code that is running. This can contain things beyond what is in your code, as it reflects the way javascript translates your code.

An execution context creates an object, and a special variable “this.” The base execution context is **global** and is the browser window. An execution context is then created for each function in your javascript.

The execution context parses through the javascript in two phases: the creation phase and execution phase. It first runs through the code and sets apart memory space for the variables and functions; then it assign variables values and runs the code.

Introduction to Javascript: Review

You should now have an understanding of how to:

- ❑ Break down a problem into steps
- ❑ Place javascript on a webpage
- ❑ Use the console to verify outputs
- ❑ Use variables
- ❑ Distinguish data types
- ❑ Use operators
- ❑ Build arrays
- ❑ Use conditionals
- ❑ Build objects
- ❑ Program repeating actions