today

view: topic brainstorms

topic: review functions, html and css manipulation, error detection

due: book exercise—ch 2, looping exercise

due: studio 2 sketch (studio 2 is due a week from today at the start of class)

thursday

read: eloquent js, ch 3 (ch 18 optional)

due: book exercise—ch 3, minimum

due: project—comparative analysis

studio

share: topic brainstorms

add: repository link in portal footer to the left of the links

javascript functions

organize & reuse code

```
var n="";

logNums();

function logNums(){
   for (var i=1; i<=7; i++){
      n+=i;
      console.log(n);
   }
}
```

one function

can have many statements

```
function getGroceries
   go to the store and buy eggplant


function getGroceries
   go to the store and buy eggplant
   while at the store buy chocolate
```

can make changes to multiple elements

```
function changePage
   change the background color

function changePage
   add padding

function changePage
   change the background color
   add padding
```

functions with arguments

```
   addNums(10,20);


   function addNums(num1, num2){
       var sum = num1 + num2;
       console.log ("the sum of " + num1 + " and " + num2 + " is: " + sum);
}
```

functions that return values

```
   var newSum = addNums(10,20);
   console.log("newSum: " + newSum);

   function addNums(num1, num2){
       var sum = num1 + num2;
       return sum;
}
```

practice: using the homework template file
create a script for the following pseudo code:

1. create a function called addNums that accepts two arguments (numbers)
   a. create a variable to hold the sum of the two arguments
   b. return the sum

2. call the function from a new variable
3. print the sum to the console

best practice tip: let the pseudo code become your comments

but what we care about is the dom

document object model (DOM)

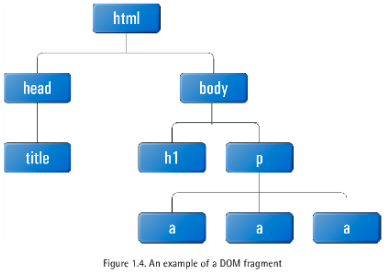


Figure 1.4. An example of a DOM fragment

From jQuery: Novice to Ninja

external javascript, check for DOMContentLoaded (best practice)

index.html

```
<head>
<script type="text/javascript" src="script.js">
</script>
</head>
```

script.js

```
document.addEventListener("DOMContentLoaded", function(event) {
    console.log("DOM fully loaded and parsed");

    // all other js here

    // call to popUpAlert
    popUpAlert();

    function popUpAlert() {
        alert("Take me to New York, I'd love to see LA");
    }
});
```

back to <form> for a second

html

```
<form id="f" name="f">
    <label>name</label>
    <input type="text" name="userName"><br>
</form>
```

back to <form> for a second

html

```
<form id="f" name="f">
   <label>name</label>
   <input type="text" name="userName"><br>
</form>
```

back to <form> for a second

html

```
<form id="f" name="f">
   <label>name</label>
   <input type="text" name="userName"><br>
</form>
```

js

```
var userName=document.f.userName.value;
```

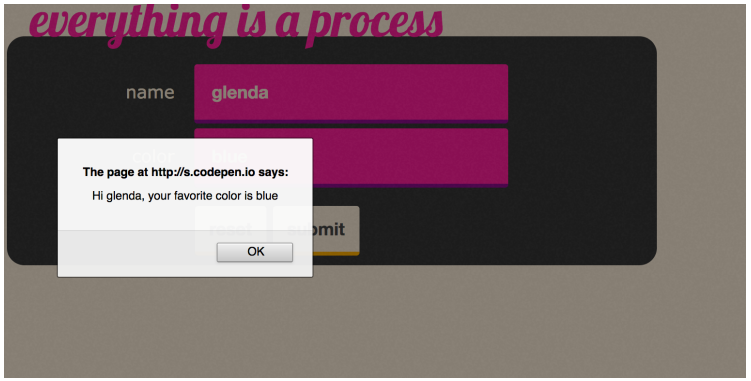
javascript dot syntax + properties

dot syntax: way to target objects or connect objects, properties and methods with dots to describe the object (or process)

```
document.f.userName.value
```

return false;
prevents the page from reloading

codepen: input process output



javascript targets html

js variables are often used to target html element
(by name, id, class, attribute, etc)

html

```
<input type="text" name="userName">

<p id="myMsg"></p>
```

js

```
//get the value from a form field
var userName=document.f.userName.value;

//target the html element
var myMsg=document.getElementById("myMsg");

myMsg.innerHTML="Hi, " + userName + "! Have a great day!";
```

javascript targets html

js variables are often used to target html element
(by name, id, class, attribute, etc)

html

```
<input type="text" name="userName">

<p id="myMsg">
```

js

```
//get the value from a form field
var userName=document.f.userName.value;

//target the html element
var myMsg=document.getElementById("myMsg");

myMsg.innerHTML="Hi, " + userName + "! Have a great day!";
```

innerHTML

js that changes the content of an html element

html

```
<section id="results">
  <p id="myMsg"></p>
</section>
```

js

```
//store myMsg element in variable
var myMsg=document.getElementById("myMsg");

...
// this code would be part of a function
myMsg.innerHTML="Hi, " + userName + "! You love the color <em>" + userColor +
"</em>!  Have a great day!";
```

innerHTML

js that changes the content of an html element

html

```
<section id="results">
  <p id="myMsg"></p>
</section>
```

js

```
//store myMsg element in variable
var myMsg=document.getElementById("myMsg");

...
// this code would be part of a function
myMsg.innerHTML="Hi, " + userName + "! You love the color <em>" + userColor +
"</em>!  Have a great day!";
```

codepen: input process output with innerHTML

studio 2 strategies

sketch first on paper!

studio: get your html and js to work,
then work on your css

1. html

form elements with name attributes
add submit and reset (self-closing)

```
<label>name</label>
<input type="text" name="userName">

<input type="submit">
<input type="reset">
```

2. js

write pseudo code first...(use comments!)
capture all user input into individual variables
add an event when the user clicks "submit" that calls a custom function
define the custom function so that it concatenates the output message and displays it
    using .innerHTML
use return false so that the page does not refresh unless you want it to

pseudocode

```
   :
IF (a > 10) AND (b = 5)
   THEN PRINT "Hello there!"
   ELSE IF (c = d) OR (a = d)
           THEN PRINT "Goodbye"
           ELSE PRINT "My head hurts"
        ENDIF
ENDIF
  :
 etc.
```

best practice tip: let the pseudo code become your comments

3. challenge yo'self

add a condition to check for form validation

```
function processForm(){
   if (userName == "" || userColor == "") {
      alert("please fill out the form!");
    }
  ...
}
```

have other css properties change (visibility, or change colors)

```
.style.property name using camel case (not dashes)

.className=""
```

interaction design concept: error detection

controlling flow with conditions

```
if (condition) {
    truePart;
}
else {
    falsePart;
}
```

or can be written in shorthand:

```
(condition)? truePart : falsePart;
```

for the purpose of evaluation…

relational operators

```
>   //greater than
<   //less than
>=  //greater than or equal to
<=  //less than or equal to
```

logical operators

```
==   //checks for equality
!=   //checks for inequality
||   //logical or
&&   //logical and
```

js can change the css!

you've got **style**

how to change style and other common tasks such as className

codepen practice: hiding and showing results

```
document.body.style.backgroundColor = "#111";

myMsg.className = "hide";
```