JavaScript

Introduction to scripting

Learning Objectives

By the end of this lecture, you should be able to:

- Understand the basics of what a scripting language is and does
- List the three locations where you can place your JavaScripts
- Describe what is meant by "case sensitive"
- Demonstrate familiarity with the various ways the `alert()` function can be used including when and when not to use quotes
- Describe what a string is
- FROM MEMORY, be able to type out a short, but <u>complete</u> web document that includes some simple line(s) of JavaScript code

About Scripting Languages

- A scripting language provides a set of instructions for the computer to follow.
    - Do a calculation
    - Change an image
    - Store some information
- JavaScript is by far the most widely-used client-side scripting language on the web.
    - We'll explain "client-side” later
- There are several other languages commonly used for web scripting. Some of the best known are PHP and Ruby.
- Full-fledged programming languages including Python, C#, Java and others are also used.

JavaScript is not Java

- This is not a particularly important point – I only bring it up here to avoid potential confusion.
- One is a scripting language, the other is a full programming language.
  - The distinction between a scripting vs a programming language is also not particularly important at this point.
- They do share quite a lot of syntax, however.
- Java is not easily embedded inside web pages. JavaScript is very easy to embed inside a web page.

Where do you put your scripts?

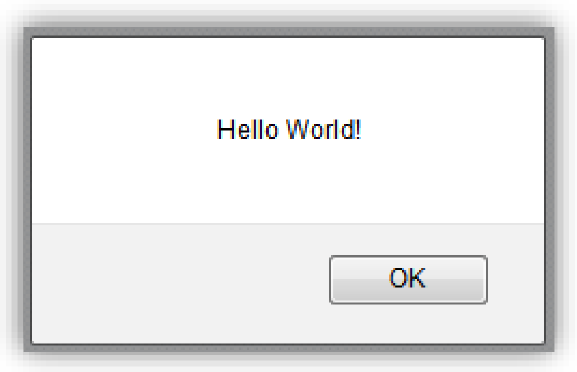
- In an HTML document, any script code *must* be placed inside a `<script>` tag

- You can place your scripting code anywhere inside the HTML document as long as it is inside the `<script>` tag.

- However, most scripting code is grouped together at the very end of the `<body>` section.

- Scripts may be placed **in an entirely separate file** (e.g. `my_favorite_scripts.js`). That file is then referenced from inside your HTML document.
  - This is analogous to external CSS files.
  - Many web programmers consider it good form to use external JavaScript documents.
  - We link to our external script in the same we that was demonstrated for external CSS files.
    - This link is also typically placed at the very end of the `<body>` section.

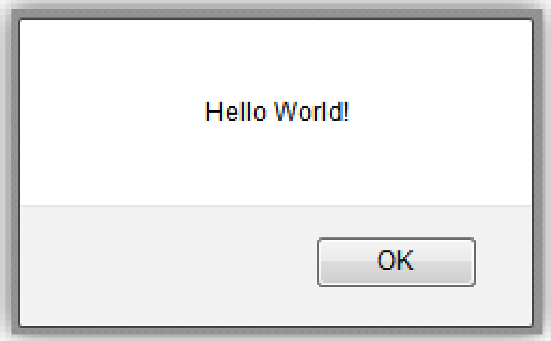
Your first JS function: `alert()`



- This is the tool we will use early on in this course to output information back to the user.
  - We will soon learn a far better way of outputting information to a web document.
- The `alert()` function can accept all kinds of information inside the parentheses.
- The `alert()` function displays information inside a dialog box that has an ‘OK’ button. When the user clicks this 'OK' button, the box goes away.

Your first JS

```
<!DOCTYPE html>
<html lang="en">
<head>
   <meta charset="utf-8">
   <title>JS Practice</title>
</head>
<body>
<h1>Welcome to my first JS Page!</h1>
Let me greet you:
 <script>
   alert( "Hello World!" );
</script>
</body>
</html>
```


Hello World!

Note the ; → Semicolons are required at the end of nearly ALL lines of JS code.

Your code will often work without them, but it is good practice to include them.
In this course, they are required.

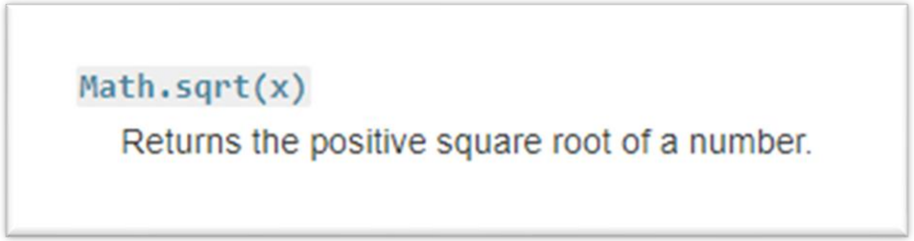
Case Sensitivity

- Many programming languages – including JavaScript— are "case sensitive".
- This means that upper case and lower case letters can not be interchanged – they are considered to be *completely* different letters!
- E.g.:
  - `alert("Hello World");` → no problem
  - `Alert("Hello World");` → **will not display!**

For this reason, whenever you look up code in a reference, be sure to pay attention to case!

Case Sensitivity

- Here is an example from the MDN reference page showing a function that returns the square root of a number.



`Math.sqrt(x)`

Returns the positive square root of a number.

- Note that the 'M' (in `Math`) is capitalized, while the 's' (in `sqrt`) is not.
  - In other words, when typing out this command, be sure to respect these distinctions. Your code will end up with errors if you do not!

Only script code should be present in the `<script>` tag

(i.e. no HTML or CSS code should be present)

```
<html>
<head>
  <title>First JS</title>
</head>
<body>
 <script>
        <h1>My First JavaScript</h
        alert("Hello World");
</script>
</body>
</html>
```

Bad!

Fun & Games with the `alert()` function

- We can place a variety of information inside the `alert()` function's parentheses. Here are some examples.
  - `alert("Hello");` → Will output the string: **Hello**
  - `alert( 5+5 );` → will output: **10** (discussed shortly)
  - `alert( "5+5" );` → will output the string: **5+5**
  - `alert( Date() );` → will output the date and time
  - `alert( Math.sqrt(25.0) );` → will output the square root of 25
  - `alert("5+5 equals " + (5+5) + ".");`

    → Uses a technique called 'concatenation'. We will discuss this later.

Note: The alert box is a JavaScript window, it is NOT a miniature web browser. For this reason, you can not place HTML code inside an alert function.

Strings

- If you put information inside quotation marks, we call that text a ‘String’.
  - You should know this term.

- A String gets output literally on the screen:
  - `alert("Hello World");` → Will output “Hello World”
  - `alert("5+5");` → will output “5+5”

- Now notice what happens when we remove the quotes:
  - `alert(5+5);` → Will output 10
  - Because it is no longer a String, the literal text will NOT be output to the screen. Instead, the JS engine will attempt to treat that text as a JS command. In this case, JS recognizes the ‘+’ symbol as an addition character and will respond accordingly. This will be discussed in more detail later.

- Another example:
  - `alert("Date()");  --> Can you figure out what will appear?`

    → **Answer:** The alert() function will output the *string* “Date()”
  - `alert(Date());`

    → JS will attempt to execute the *JS function called* Date()

JavaScript Part

Organizing JavaScript Code into Functions

Invoking JavaScript Functions

Learning Objectives

By the end of this lecture, you should be able to:

- Describe the difference between stand-alone code as opposed to code contained inside a function
- Explain what is meant by the term ‘invoke’
- Create a JavaScript function
- Connect your HTML code to a function via the `onclick` attribute
- Explain why it is important to refresh your page constantly
- **FROM MEMORY: Be able to create a very simple JS function, and invoke that function from a form**

Writing a JS Function

```
function greetTheUser()
{
    alert("Hello, user!");
    alert("I hope you have a nice day.");
}
```

1. All functions begin with the word: `function`
2. All functions must have a name (“identifier”)
   - Naming conventions (yup, another one) to use when naming a JS function:
     - First letter uncapitalized and all subsequent words should be capitalized (known as “camel case”)
     - No spaces between words
     - Avoid “reserved” words (function, if, return, etc, etc)
     - This is the same naming convention we are using when naming form elements
3. The function name is followed by parentheses: **`()`**
   - Later we will discuss what information can go inside these parentheses
4. The beginning and end of the body of the function must be delineated by braces: **`{`** and **`}`**

4. Note that while nearly all lines of JS code end with a semicolon, this does NOT apply to function <u>declarations</u> (i.e. the first line of the function).

Reminder: Clarity

```
function myFirstFunction( )
{
    alert("My first try at a function.");
    alert("I hope it works.");
}
```

- Note how each brace is placed on its own line
  - Many programmers like to place the first brace on the same line as the function declaration (the first line). This is perfectly acceptable. You will see both used in this course.
- Note how every statement inside the function is indented
- Note the ‘camel case’ convention for naming the function
- Note that there is NO semicolon at the end of the function declaration

Code inside a function
v.s.
Stand-Alone code

- The JS code below has not been placed inside a function.
- Therefore, this code will be automatically executed every single time the page runs.
- Also, when you execute this page, you may be surprised to see that even though the JS code is placed at the very end of the document, it is executed before any of the HTML code is displayed!

```
<!DOCTYPE html>
<html lang="en">
<head>
          <meta charset="utf-8">
          <title>Functions Example</title>
</head>
<body>
<h2>Some JS code...</h2>
Note how the JS code is executed automatically when the page loads.
<script>
  alert("Hello...");
</script>
</body>
</html>
```

Stand-Alone code
v.s.
Code inside a function

- If we place our JS code inside of a function, the code is NOT executed automatically.

- This allows us to control if and when the code is executed. This is almost always the desirable way of doing things.

- By placing JS code inside a function, It means that the code only gets executed when we want it to.
    - Recall that any stand-alone JS code is not only executed automatically, it is also executed before any of the HTML content on the page is displayed.

- It also means that we can execute the function multiple times if need be.

- It also means that we have the option of deciding that in a given situation the function should *never* be executed.

Code inside a function

The JavaScript code on this particular page will never be executed. The code is contained inside a function – but at no point on the page did we ever invoke the function.

```
<!DOCTYPE html>
<html lang="en">
<head>
	<meta charset="utf-8">
	<title>Functions Example</title>
</head>
<body>
<h2>Some JS code...</h2>
This JS code will never be executed!
<script>
function greetUser()
{
  alert("Hello, user!");
}
</script>
</body>
</html>
```

How to execute ("invoke") a JavaScript function

There are two ways of invoking functions that we will focus on during this course:

1. Via a JavaScript command (discussed later)
2. Via an HTML attribute called **`onclick`**

How to invoke a function:
--The `onclick` attribute --

1. The `onclick` attribute is most commonly applied to an HTML button.
   - It can be applied to other HTML tags as well, such as images, form elements, heading tags, etc. etc.
2. The value given to the `onclick` attribute must <u>match</u> the name of your function.
   - This includes the parentheses
3. To connect the function to a button we must include the `onclick` attribute inside the button's tag:

```
<input type="button"
       value="Greet the user"
       onclick="greetUser();" >
```

```
<!DOCTYPE html>
<html lang="en">
<head>
            <meta charset="utf-8">
            <title>Functions Example</title>
</head>
<body>
  <input type="button" value="Greet the user" onclick="greetUser();">
<script>
function greetUser()
{
  alert("Hello, user!");
}
</script>
</body>
</html>
```

Executing a script from a form

- In this course, we will nearly always execute our scripts in response to the user submitting an HTML form.
  - Specifically, we will apply the `onclick` attribute to an HTML button.

- Let’s go through the process of creating a button that will execute a JavaScript function.

- The steps are:
  1. Create the button
  2. Write the JavaScript function
  3. Connect the button to the function

```html
<form id="practiceForm">
  <input type="button"
         value="Greet Me!" >
</form>
```

```
<script>
  function greetUser()
  {
     alert("Hello, dear user!");
     alert("Have a great day!");
  }
</script>
```

Pop-Quiz: Where in your HTML document should this script be placed?

Answer: Just before `</body>`

```
<form id="practiceForm">
  <input type="button"
         value="Greet Me!"
         onclick="greetUser();">
</form>
```

- The value of the `onclick` attribute must precisely match the name of the JS function you wish to invoke (including the parentheses).
- The semicolon inside the attribute value is optional.

Complete Example

```
<!DOCTYPE html>
<html lang="en">
<head>
        <meta charset="utf-8">
        <title>Functions Example</title>
</head>
<body>
  <form id="practiceForm">
     <input type="button" value="Greet Me!" onclick="greetUser();" >
  </form>
<script>
function greetUser( ) {
   alert("Hello...");
}
</script>
</body>
</html>
```

Aside: Note that the opening brace of the function is on the same line as the function declaration. This is fine – in fact, many users prefer to place it there. The closing brace, however, should always be on its <u>own</u> line.

A common error:

- The function declaration should NEVER have a semicolon after it.
- Doing so will cause the body of your function to never execute!

```
function greetUser( );
{
   alert("Hello...");
}
```

Never place a semicolon after your function declaration.

JavaScript

Variables

Learning Objectives

By the end of this lecture, you should be able to:

- Understand what it means to "declare" a variable
- Appreciate the importance of giving a clear and useful identifiers to variables
- Apply a basic mathematical formula and assign the resulting value to a variable
- Become more comfortable with the use of strings
- **FROM MEMORY: Be able to create a simple conversion page such as a conversion between fahrenheit and celcius.**

Using “*variables*” to store information

What is a variable?

- Often (*very* often) in programming, we need to temporarily store information. For example, suppose you want to retrieve the user’s name and email address from a web form so that you can enter it into a database. Where will you store this information in the short term?
- You can store this information inside something called a ‘variable’. A variable is a tiny space in your computer’s memory that you reserve in order to store a piece of information.

How to we create a variable?

- In order to create a variable we simply “declare” it. Here is an example of declaring a variable called ‘userName’:

```
var userName;
```

  – That’s it!! You can now store any information in this variable that you want.

- Example:

```
var userName;           → Declaring the variable
userName = "Robert"; →Assigning a value to the variable
```

Variables cont.

- You can create *as many variables as you want* in a script.
- A variable must only be declared once in a function. In other words, once you've declared a variable, you do not declare it again.
- However, you can assign a different value to a variable over and over again.
  - Every time you assign a value to a variable, the previous value that was stored in that variable replaced by the new value.

- Here is an example of a simple script that declares two variables, one to store a quantity in U.S. Dollars, and second to store a value in Mexican Pesos. We will then do a very simple mathematical conversion and output the result in an alert box.
  - **Convention Alert!** Note the naming convention we use when we name our variables. We use the same camel-case naming that we used when naming functions.

Examine the following code. The full script is on the next slide.

```
var usDollars;
var mexicanPesos;

usDollars = 50;
mexicanPesos = usDollars * 12.87;

alert(mexicanPesos);
```

```
<!DOCTYPE html>
<html lang="en">
<head>
        <meta charset="utf-8">
        <title>Practice Makes Perfect</title>
</head>

<body>
<h1>It Takes Practice...</h1>
  <input type="button" value="Convert Dollars to Pesos"
         id="btnSubmit" onclick="dollarsToPesos()" >

<script>
function dollarsToPesos()
{
  var usDollars;
  var mexicanPesos;

  usDollars = 50;
  mexicanPesos = usDollars * 12.87;
  alert(mexicanPesos);
}
</script>

</body>
</html>
```

Some Important Notes

Naming convention: Note that we use the same naming convention for our variable identifiers as we've used for our function identifiers, i.e. camel-case.

Declaring variables: We typically declare one variable per line. However, if you have related variables, we often will group them together on the same line with commas in between:

```
var firstName, middleName, lastName;
```

Declare at the top: We typically declare our variables at the top of a function, even if we are only using them several lines below.

Assigning values to our variables

- Notice how we assigned a value to our variable:

```
usDollars=50;
```

- At the moment, the only way we can assign a value inside our variable is by putting it there ourselves. For example, in the example above, we manually typed in the value of '`50`' to the variable **`usDollars`**.
  - However, we will soon learn how to retrieve these values from our HTML forms. At this point, our code becomes much more powerful. When we reach that point, we will allow the user to enter the value via the form.
  - For example, we will create a form that asks the user to enter an amount in US Dollars. Our script will then retrieve that value from the form, and assign it to the variable **`usDollars`**.

Variables can store just about anything

Variables will be used in pretty much every single piece of code that you will ever write!

We can store any type of data into a variable such as numbers or strings (i.e. text).

Some examples:

```
var  temperatureCelcius, temperatureFarenheit;
var  userName, userAddress;

temperatureCelcius = 33;
temperatureFarenheit = (9 / 5 * temperatureCelcius)+32;
userName = "John Doe";
userAddress = "222 Memory Lane";
```

STRINGS

- *Strings are one of the very most important data types in programming.*
- A combination of letters / words / symbols, etc is called a “String”.

- Strings are <u>always placed in quotes</u>.

- Some examples of Strings – be sure to understand them!

– `"Robert Smith Johnson"`	→ a simple string
– `"273"`	→ a string of digits – ***NOT a number***!!
– `"#$@(~?%(*^%#$"`	→ a string of random characters
– `""`	→ an empty string
– `" "`	→ a string containing a single space

```
var userName, phoneNumber,  favoriteBand;

userName = "John Doe";
phoneNumber = "3124445555";
favoriteBand = "Spinal Tap";
```

Doing Math with JavaScript

```
var  temperatureCelcius, temperatureFahrenheit;
var  mathResult;

temperatureCelcius = 33;
temperatureFahrenheit = (9/5* temperatureCelcius )+32;
alert(temperatureFahrenheit);

mathResult = Math.sqrt(25) + 5;
alert(mathResult);
```

File: **`temperature_conversion.html`**

```
<!DOCTYPE html>
<html lang="en">
<head>
        <meta charset="utf-8">
        <title>Practice Makes Perfect</title>
</head>

<body>
<h1>It Takes Practice...</h1>
<hr>
        <input type="button" value="Convert 40 Celcius to Farhenheit"
                   id="btnSubmit" onclick="celciusToFahrenheit()">
<script>
  function celciusToFahrenheit()
  {
    var fahrTemperature;
    var celciusTemperature;

    celciusTemp = 40;
    fahrTemp = 9/5*celciusTemp +32;
    alert(fahrTemp);
  }
</script>
</body>
</html>
```

Examples of good and not-so-good identifiers

Let's look at some more examples using function names:

```
function convert()
```

- Not good! This identifier gives absolutely no idea of what the function is supposed to do.

```
function convertFarhenheitToCelcius()
```

- This is much better. A bit long perhaps, but there is nothing wrong with having a slightly long-ish identifier if that identifier really explains what is going on.
- Don't overdo it though!

```
function convertFahrToCel()
```

- Also good. Given the context, most programmers would agree that this is pretty clear.

```
function fToC()
```

- Hopefully we can all agree that this is also a terrible identifier!

Examples of good and not-so-good identifiers

And a few more… Suppose we are writing a function that will ask the user for their birthday:

`function getDate()`

- Not good. Are we looking up the current date? The date of the invasion of Normandy?!

`function userBirthday()`

- Definitely better, though the identifier we can't tell whether the function is reacting to a known birthday (e.g. determining if the user is 21 or over), or if it is planning on doing something based on their birthday such as reveal their horoscope, etc.

`function getUserBirthday()`

- Much better! It's not an overly long identifier, but gives a pretty good idea of what the function is supposed to do.

Some rules for identifiers

When choosing your identifier, keep the following in mind:

An identifier must be a sequence of
- letters
- digits

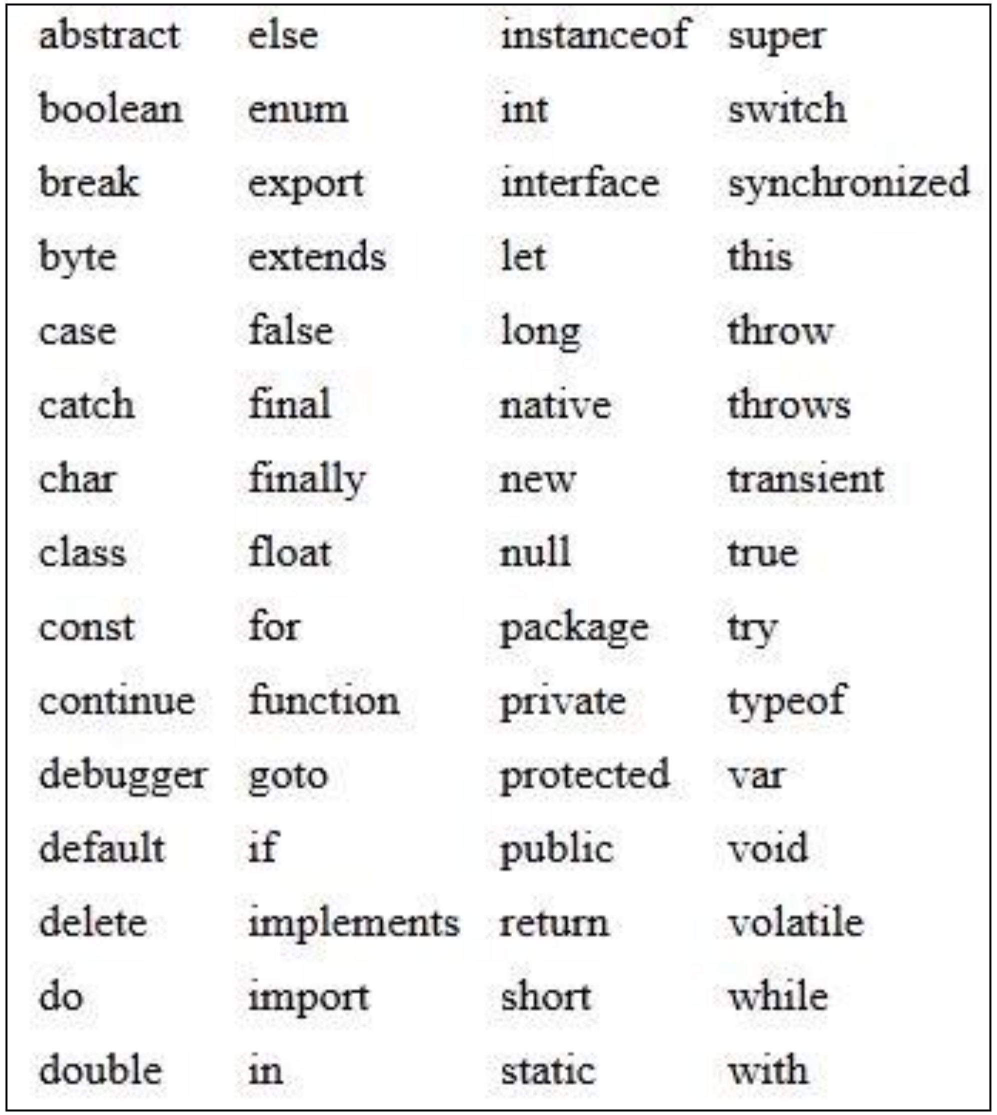
It should typically begin with a lower-case letter, and then use camel case for subsequent words.

An identifier can NOT:
- Begin with a digit
- Have a space
- Have mathematical operators such as: +, -, /, *, %, etc
- Be a ‘reserved’ word (discussed on the next slide)
- Have various other characters such as & **:** ^ #

Don't forget: JavaScript is *case sensitive* so “firstName” and “FirstName” are different variables.

Reserved Words

- Here is a list of words that can not be used as an identifier.
- We call these 'reserved keywords'.
- Most of these are words that 'mean something' in JavaScript, e.g. **`function`** or **`var`**.
- You do *not* need to memorize this list. It's just something you should keep in the back of your mind when you are choosing an identifier.



abstract	else	instanceof	super
boolean	enum	int	switch
break	export	interface	synchronized
byte	extends	let	this
case	false	long	throw
catch	final	native	throws
char	finally	new	transient
class	float	null	true
const	for	package	try
continue	function	private	typeof
debugger	goto	protected	var
default	if	public	void
delete	implements	return	volatile
do	import	short	while
double	in	static	with

JavaScript

Retrieving information from forms

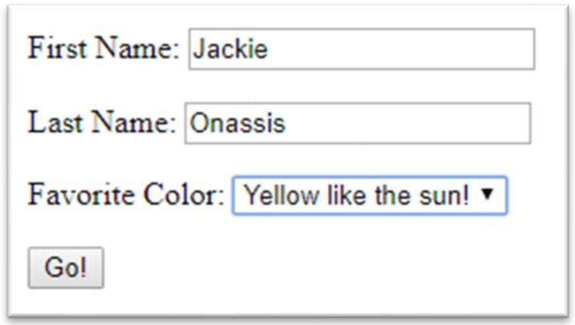
Learning Objectives

By the end of this lecture, you should be able to:

- Describe where the information comes from when retrieving from a text box or text area, as opposed to a form element that has a 'value' attribute.
- Comfortably retrieve values from textboxes, text areas, select boxes, and store those values inside variables
  - Do the above without looking at your notes

Retrieving values from a form using JavaScript

- Consider this simple form:



First Name:	Jackie
Last Name:	Onassis
Favorite Color:	Yellow like the sun! ▾

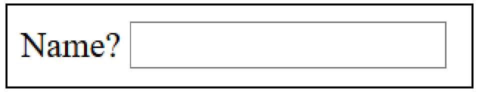
- As we have seen, creating this form using HTML code is pretty straight-forward.

- The next -- and key -- step is to learn how to retrieve the values that were entered into this form (e.g. the value of the two text fields, and the value form the select box) using JavaScript.

There are two ways to retrieve a value

In order to retrieve a value from a form, your form element *must* either:

1. Require the user to enter the value (e.g. via a text box or a text area)



In this case, the value we retrieve will be whatever the user has typed into the text box.

--- OR ---

2. Include a ‘`value`’ attribute, when you create the form element
   - In this case, the value comes from form elements in which you (the developer) have included an attribute called ‘`value`’, and then you – the programmer—chose a value for that attribute.
   - For example, recall that when we created the `<option>` tags in our select boxes, we assigned values to them:

```
What is your birth month?
<select id="selMonthBorn">
       <option value="jan">January</option>
       <option value="feb">Febuary</option>
       <option value="mar">March</option>
       etc...
</select>
```

How to retrieve a value from a form element using JavaScript

The syntax is:

```
document.getElementById("ID-NAME").value;
```

For example, suppose our form has a text box with an ID of 'txtFirstName'. We could retrieve whatever the user entered in that text box AND store that value inside a variable with the following line of code:

```
var name = document.getElementById('txtFirstName').value;
```

- This JavaScript command will:
  1. Look for a text field called ‘txtFirstName’ and will retrieve whatever value the user typed inside that text field.
  2. The code will then store that value inside the variable called ‘name’.

Important Notes:

- The word ‘document’ will not change during this course. It simply refers to the current web page.
- The value inside the parentheses must match with one of the ID names in your HTML document.
- The ‘value’ term says you wish to retrieve the value from that element.
  - If the element you are getting the value of is a text field or text area, then the information you get back will be whatever was *typed* by the user.
  - If your element has a *built-in* value attribute (e.g. select boxes), then that is the information that will be retrieved.

Example: Retrieving the value from a select box:

```
var monthBorn = document.getElementById('selMonthBorn').value;
```

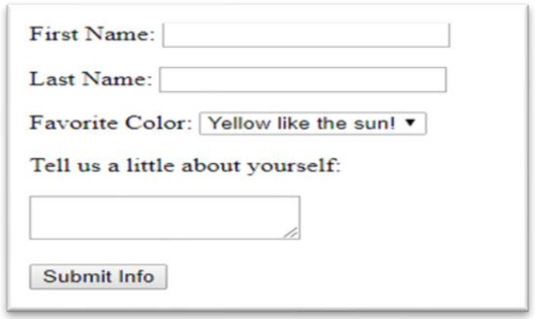
- Will retrieve the value selected by the user from a select box called ‘**selMonthBorn**’
- Suppose that one of the options in that select box was:

  `<option value="feb">Febuary</option>`
- If the user selects that option, then the variable monthBorn will be assigned "**feb**".

Retreiving and storing information from a form element

- Once we retrieve a value from a form, where should we put it?
- This is one of the countless situations in programming where we use variables.

Imagine you have the form shown here:



First Name:

Last Name:

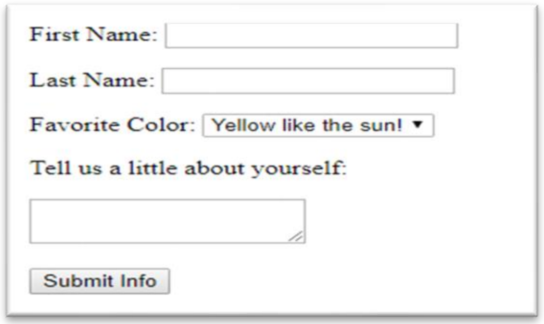
Favorite Color: Yellow like the sun!

Tell us a little about yourself:

Can you predict what the lines of code are supposed do? Try to figure it out before moving to the next slide.

```
var firstName, lastName;
var nationality;
var favoriteColor;
var aboutUser;

firstName = document.getElementById('txtFirstName').value;
lastName = document. getElementById('txtLastName').value;
favoriteColor = document.getElementById('selFavoriteColor').value;
aboutUser = document.getElementById('textarAboutUser').value;
```

Retreiving and storing information from a form element


First Name:

Last Name:

Favorite Color: Yellow like the sun!

Tell us a little about yourself:

Submit Info

```
1. var firstName, lastName;
2. var nationality;
3. var favoriteColor;
4. var aboutUser;

5. firstName = document.getElementById('txtFirstName').value;
6. lastName = document. getElementById('txtLastName').value;
7. favoriteColor = document.getElementById('selFavoriteColor').value;
8. aboutUser = document.getElementById('textarAboutUser').value;
```

- Lines 1-4 are, of course, our variable declarations.
- Lines 5 and 6 are retrieving the values of text fields. Therefore, the information that gets retrieved will be whatever the <u>user</u> typed into those text fields.
- Line #7 is retrieving the value of a select box. In *this* case, the text stored inside the variable `favoriteColor` will be the information entered inside the `value` attribute of the particular option tag that was selected by the user.
    - Recall that in this case, the `value` was set by the <u>web page programmer</u> – NOT the user who is completing the form!
- Line #8 retrieves whatever text the user typed in a text area.
    - I know it is a text area without looking at the HTML code since, when creating the form, I used the `textar` prefix. This is just one way in which naming conventions can make it easier to interpret what the code is trying to do.

JavaScript

Adios `alert()`

Using `innerHTML`

Using an empty section

Learning Objectives

By the end of this lecture, you should be able to:

- Learn a <u>much</u> better technique than `alert()` to output information to a web page
- Learn how to use a blank section as a placeholder for information you wish to output

Bye-bye `alert()`

It is time to say goodbye to an old friend. Though the `alert()` function has served us well, it should not typically be used in the "real world". At the very least, `alert()` should be reserved for only a few particular situations.

Some limitations of `alert()` include – but are not limited to:

- The alert box can not display HTML markup.
- The alert box can not display images.
- The alert box disappears (along with any information inside) when the user clicks the 'OK' button.
- The user must click 'OK' in order to make the alert box disappear.
- Information in an existing alert box can not be modified.

Therefore, we are going to pretty much retire alert boxes, we are now going to learn how to use a different JavaScript command to output content directly into an existing web page.

Outputting using `innerHTML`

The following JavaScript code will output the words: **Look at me!** (in `<h1>` markup) into a section on the page that has a ID named '`output`':

```
var someText = "<h1>Hello World!</h1>";
document.getElementById("output").innerHTML = someText;
```

This `innerHTML` command is extremely helpful and widely used.

IMPORTANT: It is very important to note that the `innerHTML` command replaces everything that was in that section before. So in this case, the above command would replace everything that was previously inside '`output`' with the words *Hello World!*.

Imagine if that `output` section contained a whole bunch of important information that was there earlier. If you then issued the innerHTML command, all of that existing content would be replaced with the words "*Hello World!*". This is clearly not a desirable result.

Fortunately, there are various easy fixes. We will discuss one option next.

Creating an empty section

We will place an empty section in our document. The one and only job of this section is to hold the output of the `innerHTML` command:

```
<div id="results">
</div>
```

The following JavaScript code will output the words: *Look at me!* in `<h1>` markup into our `results` div section:

```
var someText = "<h1>Look at me!</h1>";
document.getElementById("results").innerHTML = someText;
```

1. The first line of code simply creates a variable that holds a string containing the content we want to insert into our web document.
2. The second line inserts the string into the '`results`' section.

innerHTML_greeting.html

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="utf-8">
  <title>Greeting with innerHTML</title>
</head>
<body>
<h1>innerHTML Example</h1>
<p>What is your name?
<input type="text" id="txtName">

<p><input type="button" value="Say Hello to Me!" onclick="greetUser()" >

<div id="greetingOutput">
</div> <!-- end of greetingOutput div -->

<script>
function greetUser()
{
    var userName = document.getElementById("txtName").value;

    var greeting = "Hello, " + userName;
    //Recall that you can join 2 strings together with the + operator

    document.getElementById("greetingOutput").innerHTML = greeting;
}
</script>
</body>
</html>
```

Concatenation of Strings

JavaScript Comments

Learning Objectives

By the end of this lecture, you should be able to:

Concatenation:

- Describe the two different operations carried out by the '+' operator.
- Describe the circumstances under which the operator does addition, and when it does concatenation.
- Join variables, literal text, and even numbers into a single string using concatenation.

Comments:

- Be comfortable inserting comments in your code.
- Demonstrate how comments can be used to temporarily remove code for debugging and testing purposes.

The lowly '+' operator

We are all familiar with this operator… we use it for addition all the time.

- `alert(5+2);` → outputs 7
- `alert(5+5);` → outputs 10

However, this '+' operator is unusual in that it has *two* different uses. In addition to 'adding', the + operator, if used with strings, can "concatenate" (join) those strings together.

- `alert("H" + "e" + "l" + "l" + "o" + ".");` →ouputs: *Hello.*
- `alert("Hello, " + "How are you?");` → outputs: *Hello, How are you?*
- `alert("5" + "5");` → ***what do you think??***

→ ***Answer:*** *outputs the string "55"*

Form values are retrieved as strings

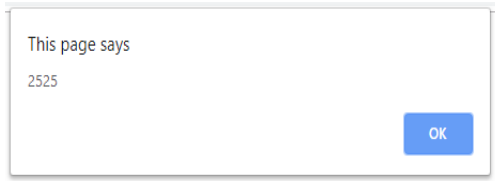
The following is a very very important point

Any time you retrieve a value from a form, that value comes back as a <u>string</u>.

Even if the value looks *exactly* like a number, that "number" is in fact, a string!

Concept check: Can you predict what will be output by the alert box here?

```
var age;
age = document.getElementById('txtAge').value;
//Let's suppose that the user entered 25 for their age

alert(age+age);
```



```
This page says
2525
```

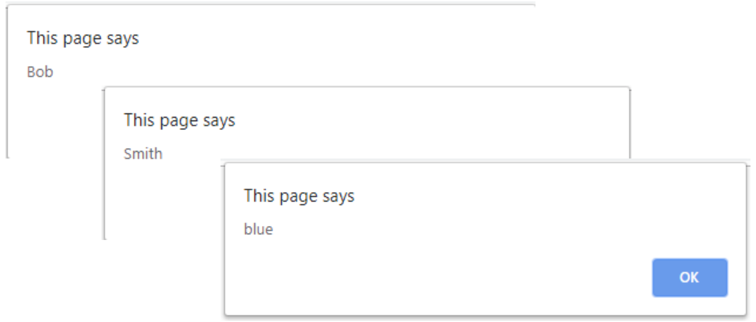
- Because ‘age’ is holding a string (as opposed to a number), the ‘+’ operator will do concatenation as opposed to addition.
- This type of thing comes up <u>a lot!</u> And if we don't learn how to "fix" it, all kinds of errors and bugs in your code will result.

- **Be *absolutely certain* that you understand this situation. It will be vital in upcoming assignments, and I can *guarantee* exam questions relating to it!**

Concatenation Example

Concatenation becomes extremely useful when we want to output the value of different strings and variables together. Consider this situation:

```
var firstName = "Bob";
var lastName = "Smith";
var favColor = "blue";
alert(firstName);
alert(lastName);
alert(favColor);
```



```
This page says
Bob
```

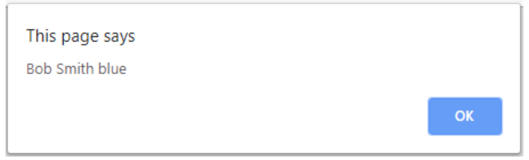
```
This page says
Smith
```

```
This page says
blue
```

Hopefully you will agree that it is hardly good scripting!

Now compare it with:

```
alert(firstName + " " + lastName + " " + favColor);
```



```
This page says
Bob Smith blue
```

By concatenating the five strings together, this line of code will output all three variables in only one alert box.

Note the need to put spaces between each string. If we did not, we would end up with one long string.

Concatenation example

Study this example to ensure that you understand it.

```
var firstName, lastName, age;
firstName = document.getElementById('txtFirstName').value;
lastName = document.getElementById('txtLastName').value;
age = document. getElementById('txtAge').value;

alert("Dear " +  firstName  +  " "  +  lastName +  ".");
alert("You are "  +  age  +   " years old.");
```

As always, be sure to type out the example yourself and to experiment with it.

Incidentally, note how I typed a space before and after certain strings in our alert statements. If I neglected those spaces, I hope you can see that the words would all be scrunched up against each other.

Next, experiment by placing the entire string in just one `alert()` window instead of two.

File: concat_ex1.html

```
<!DOCTYPE html>
<html lang="en">
<head>
        <meta charset="utf-8">
        <title>Practice Makes Perfect</title>
        <script>
        function testConcatenation()
        {
          var firstName, lastName, age;
          firstName = document.getElementById('txtFirstName').value;
          lastName = document.getElementById('txtLastName').value;
          age = document.getElementById('txtAge').value;

          alert( "Hello " +  firstName  +  " "  +  lastName +  "! "
                + "You are "  +  age  +   " years old.");
        }
        </script>
</head>
<body>
<h1>It Takes Practice...</h1>
<form id="userInfo">
        <p>First Name:   <input type="text" id="txtFirstName">
        <p>Last Name:    <input type="text" id="txtLastName">
        <p>Age:          <input type="text" id="txtAge">

        <p><button onclick="testConcatenation()">Submit</button>
</form>
</body>
</html>
```

When Form Elements Go Rogue

File: age_next_year_no_parse.html

Be sure to study this example so that you can appreciate the problem.
(Explanation on the next slide)

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="utf-8">
  <title>Form Elements are Strings</title>
  <script>
  function calcNextYear()
  {
    var ageThisYear, ageNextYear;
    ageThisYear = document.getElementById('txtAge').value;
    ageNextYear = ageThisYear + 1;

    alert("Next year, you will be: " + ageNextYear + " years old!");
  }
</script>
</head>
<body>
  <h1>How old???!!</h1>
  <p>How old are you now? <input type="text" id="txtAge">
  <p><button onclick="calcNextYear()" >How old will I be next year?</button>
</body>
</html>
```

New Topic: “*Comments*”

Recall that comments are very important in programming. As your code becomes increasingly complex, you will find that including comments in yrou code makes life far, far more pleasant for other programmers working with you on your code.

We have already discussed how to comment in HTML. Recall that placing text inside `<!--` and `-->` will case that text to be ignored by the web browser. That is, the text will be ignored by the interpreter that is displaying content on the web browser.

Similarly, the JavaScript also has a syntax for including comments. In fact, it has two methods:

1. Placing two forward slashes `//` anywhere on a line of JS code tells the interpreter that anything that follows is a comment and should be ignored.
   - ➢ If you have multiple lines you wish to “comment out”, then you must put the `//` characters before each and every one of those lines.

2. If you wish to write several lines of comments (a common occurrence), it might be tedious to type double slashes at the beginning of each line. In this situation, we can use "multi-line comments". To do so, type `/*` Everything that follows until you type a closing `*/` will be treated as a comment.

JavaScript

Data Types

Parsing Data

Learning Objectives

By the end of this lecture, you should be able to:

- Explain what is meant by the term "argument".
- Identify the three "data types" that we will discuss through the remainder of the course.
- Recognize how and when to apply the `parseInt()` and `parseFloat()` functions.
- Recognizing when not to use the parse functions.
- Learning about programming by studying other people’s code.

"Arguments"

In programming, one term with which we should be familiar is: *argument*.

An argument is the information you provide to a function when you invoke it. Some functions require 0 arguments, others may require 1 argument, and others can require multiple arguments.

This is a very simple term, but it is important that you are comfortable using it.

For example:

- `alert('Hi');` --> the string 'hi' is the argument
- `Math.sqrt(23.7);` --> 23.7 is the argument
- `Math.pow(3,4);` --> this function has two arguments: 3 and 4
- `Date();` --> this function has 0 arguments

Data Types

- In many programming languages, every piece of data that we work with has a “**data type**”.

- There are many different **data types** out there, but in this course, we will focus on three.

- **The three data types you should be able to name:**
  - Strings → you are already familiar with these!
  - Integers → numbers that are ‘whole’ (i.e. without a decimal)
  - Floats → numbers that have a decimal

- Examples:
  - x = "25" → x is holding a *String*
  - x = 25 → x is holding an *Integer*
  - x = 25.0 → x is holding a *Float*

Examples

In each of the four assignment statements below, identify the value and data type that will be stored inside the variable `number`:

```
var x1 = "25";
var x2 = 10;
var x3 = 5.3;
var number;

number = x1 + x2;
       → Value: "2510"  Data type: string
number = x1 + x3;
       → Value: "255.3"  Data type: string
number = x2 + x3;
       → Value: 15.3  Data type: float
number = x2 + 4;
       → Value: 14  Data type: int
```

- In the first two cases, we have a string on one side of the '+' operator. Recall that the moment a string is present on either side, we will get concatenation.
- The third and fourth examples involve two numbers. In these case, then, the '+' operator will do addition.

The `parseInt()` function

- JavaScript includes a very useful function that specializes in converting strings into integers. This function is called **`parseInt()`**.

- `parseInt()` accepts a piece of information (typically a string) inside its parentheses. (This piece of information is the "argument" that we discussed earlier).

- The `parseInt` function then attempts to convert that information into an integer. If the information inside the parentheses "looks" like an number, then great!

- If the information inside the parentheses does *not* look like a string, however, your script will generate an error and your page will not display properly.

`parseInt` Examples

Can you predict what will be stored inside the variable ‘**`temp`**’ in each of the following examples?

```
1.    var temp;
2.    temp = parseInt("352");
3.    temp = parseInt(49.99);
4.    temp = parseInt("ten");
```

- Line 2?
  - **`temp`** will store the integer 352
  - The `parseInt` function has no difficulty converting this particular string to an int
- Line 3?
  - **`temp`** will be assigned the integer 49
  - The `parseInt` function has no difficulty converting this float to an int
  - ***Important:*** Note that `parseInt()` does not round numbers up or down. Rather, the function simply chops off the decimal. We have a fancy word for this too (sorry): “truncation”.
- Line 4?
  - **Error**: As humans, we can easily figure out what is *supposed* to happen. Sadly, computers are unable to make that leap.
  - **IMPORTANT:** A key point to know is that the value being converted by the `parseInt` function must "resemble" a number in order for `parseInt()` to be able to figure out the conversion.

parseFloat()

What if the number you are planning to retrieve may contain decimals?

- E.g. The time to complete a 100 meter dash, the cost of a gallon of gas, etc

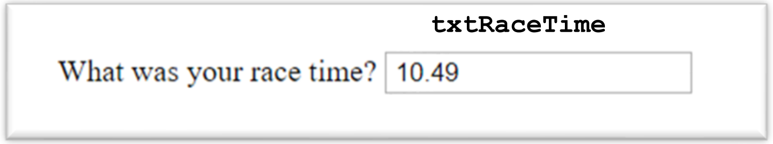
Recall that `parseInt()` *stops* parsing the instant it encounters any non numeric character – such as a decimal.

Therefore, if you want to keep your decimals, you would *not* want to use `parseInt()`.

Instead, you must use **`parseFloat()`**

For example, suppose you are recording the race time for a 100 meter sprint where the times are often recorded to hundredths of a second. In this case, we would absolutely need to keep the decimal places. Therefore, we *must* use **`parseFloat()`** instead of `parseInt()`

```
var raceTime = document.getElementById('txtRaceTime').value;
raceTime = parseFloat(raceTime);
```



What was your race time? 10.49

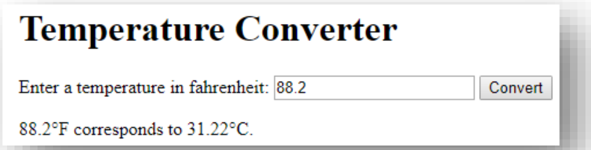
File: parse_example.html

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="utf-8">
  <title>Parse Example</title>
</head>
<body>
  <h1>Parse Example</h1>
  <p>How old are you now? <input type="text" id="txtAge">

  <p><button onclick="calcNextYear()">How old will I be next year?</button>

<script>
function calcNextYear() {
  var ageThisYear, ageNextYear;

  ageThisYear = document.getElementById("txtAge").value;
  ageThisYear = parseInt(ageThisYear);

  ageNextYear = ageThisYear + 1;

  alert("Next year, you will be: " + ageNextYear + " years old!");
}
</script>
</body>
</html>
```

Reminder: We are only using `alert()` functions in these examples to minimize unrelated code. In the real world, we would use **`innerHTML`**.

File: `fahrenheit_celcius_converter.html`

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="utf-8">
  <title>Temperature Converter</title>
</head>
<body>
<h1>Temperature Converter</h1>
<form id="conversionForm">
  <p>Enter a temperature in fahrenheit:
    <input type="text" id="txtFahrenheit">
    <button type="button" onclick="convertToCelcius()">Convert</button>
    <!- We *need* the type attribute above! Otherwise our innerHTML won't work. -->
</form>

<div id="results">
</div><!-- end of results div -->

  <script>
  function convertToCelcius() {
    var fahrenheit, celcius;
    var resultsString;

    fahrenheit = document.getElementById('txtFahrenheit').value;
    fahrenheit = parseFloat(fahrenheit);
    //people may well enter a decimal here, so use parseFloat

    celcius = (fahrenheit-32)*5/9;  //Should we parse 'celcius' ??
    //No!  celcius is holding a number -- NOT a string
    //therefore we do not need to parse it.

    celcius = celcius.toFixed(2);   //limit to 2 decimal places

    resultsString = fahrenheit + "&#176;F corresponds to " +
                    celcius + "&#176;C."; //Note the entity code!

    document.getElementById("results").innerHTML = resultsString;
  }
</script>
</body>
</html>
```


Temperature Converter

Enter a temperature in fahrenheit: 88.2 Convert

88.2°F corresponds to 31.22°C.

IMPORTANT: One of the best ways to finesse your programming skills and learn new ones is by studying existing code. We will experiment with this approach here by introducing a couple of new points without several slides of formal "PowerPoint" discussion. However, I will provide some explanation.

There are some important concepts discussed in this example – **be sure to study it closely, and review periodically!**

For reasons beyond the scope of this discussion, the `innerHTML` command won't work unless we include the `type="button"` attribute. So, if ANY button is invoking a function that includes `innerHTML`, be sure to include this attribute.

To be on the safe side, feel free to include this attribute in all of your buttons if you like. (I've also included this on the assignment checklist).

Remember to close `<div>` sections with a comment indicating the name of the section that was closed.

Remember that we should NOT parse items that do not require it! In this case, the variable **`celcius`** is ALREADY holding a number. It is not a string, so we do not need to parse it!

The `toFixed()` function is quite useful! Note that we must reassign the value returned by this function to the same variable.

We create a string variable to hold our rather long concatenated string.

We can then assign that string to our `innerHTML` command.

When dealing with a very long string, this technique makes the code easier to write and follow.

JavaScript

Controlling the flow of your programs with ‘if’ statements

```
if (something is true)
{
  Do this stuff here
}
else
{
  Do this other stuff
}
```

Learning Objectives

By the end of this lecture, you should be able to:

- Describe what is meant by a ‘conditional expression ‘ (a.k.a. boolean expression, a.k.a. logical expression) and how they are evaluated
- Understand how the order (a.k.a. “flow”) of JavaScript commands is executed based on whether or not a conditional evaluates to true
- Write simple scripts demonstrating *comfort* with if and if-else statements
- Interpret – and write – slightly more involved web page using if-else statements such as the one demonstrated in `salary_calculator.html`

Syntax: The if statement

```
if (conditional)
{
  Block A statements…
}
else
{
  Block B statements…
}
```

$\leftarrow$ **NOTE: There is NO semicolon placed at the end of a conditional.**

If the conditional inside the parentheses is true, the code inside the braces labeled 'Block A' will be executed. If the conditional is false, then the flow skips the block. In that case, the code inside the braces after the **`else`** statement will be executed.

However, if the 'if' condition is true, then once that "true" block has been executed, the **`else`** block will be skipped.

Blocks can be as long/short as you like

```
if (conditional)
{
  Block A statements
  Block A statements
  Block A statements
  Block A statements
  Block A statements
  Block A statements
  Block A statements
  Block A statements
}
else
{
  Block B statements
}
```

The code inside these blocks can be as short (e.g. one line) or as long (e.g. 500 lines) as you wish.

Example – Movie Ticket

Example: We retrieve the value from a form in which we ask the user their age.

Our business requirements specify that a regular ticket costs $10, while a ticket for people 65 or over costs $5.

```
var age;
age = document.getElementById('txtAge').value;
age = parseInt(age);

if (age >= 65)
{
     alert("Your ticket costs $5.00");
     //We would use innerHTML in the “real world”
}
else
{
     alert("Your ticket costs $10.00");
}
```

Complete file: `movie_ticket_simple.html`

What happens after the if/else block?

Once an if/else block has completed, the program simply <u>continues</u> on through to the end of the function.

In this example, <u>regardless</u> of whether or not the if block gets executed, the program will simply continue on and complete the rest of the function. In this case, the program will always alert “Goodbye”.

```
function determineBonus()
{
  var hours;
  hours = document.getElementById('txtHours').value;
  hours = parseFloat(hours);

  if (hours > 40)
  {
     alert("You qualify for overtime pay!");
  }

  alert("Goodbye!");
}
```

Another example

Let’s write a script in which we ask the user for their age. If their age is 18 or over, print: “You can rent a car!”

We will do it with and without an else block.

Take a moment to try and write the relevant code your own before looking at my version. Start with a version that does not have an else block. Simply output “You can rent a car” if the user is 18 or over. Assume the text field is called “`txtAge`”. You can use `alert()` to keep things simple for the exercise.

```
var age;
age = document.getElementById('txtAge').value;
age = parseInt(age);

if (age >= 18)
{
   alert("You can rent a car!");
}
```

Exercise

1. Ask the user how many hours they worked in the last week. If they worked more than 50 hours, output "Wow, you're a hard worker". If they did not work over 50 hours, don't say anything. Assume the text field is called "`txtHours`".
   - ***NOTE: Do NOT confuse 'more than 50 hours' with '50 or more hours'… One uses the '>' operator, the other uses the '>=' operator.***

```
var hours;

hours = document.getElementById('txtHours').value;
hours = parseFloat(hours);

if (hours > 50)
{
  alert("Wow, you're a hard worker!");
}
```

Exercise

Ask their age: If the user's age is 65 or over, say "You are eligible for retirement benefits", otherwise, say "Sorry, no benefits yet!" Assume the text field is called "`txtAge`".

Again, feel free to use `alert()` since we are just practicing.

```
var age = document.getElementById('txtAge').value;
age = parseInt(age);

if (age >= 65)
{
        alert("Eligible for retirement benefits");
}
else
{
        alert("Wait a few years, sorry.");
}
```

The conditional:

if (………)

- The conditional is <u>the</u> key component of ‘if’ statements.
- The conditional asks a question that must ultimately be evaluated as either ‘true’ or ‘false’

- Is the user less than 21 years old?
  - `if( age < 21)…`

- Is the due date later than today’s date?
  - `if (dueDate > todayDate)…`

JavaScript

Precedence

Operators

Error Types

Learning Objectives

By the end of this lecture, you should be able to:

- Identify the ‘operators’ on a given line of code.
- Identify the precendence (order) in which each of those operations will be carried out.
- Recognize situations in which you may encounter an ‘NaN’ error code.

Operator “Precedence”

```
var num1, num2, result;
num1 = 5;
num2 = 20;
result  =  num1 + num2 / 5 - 1;   //What is the value of result?!
```

Some key precedence rules:

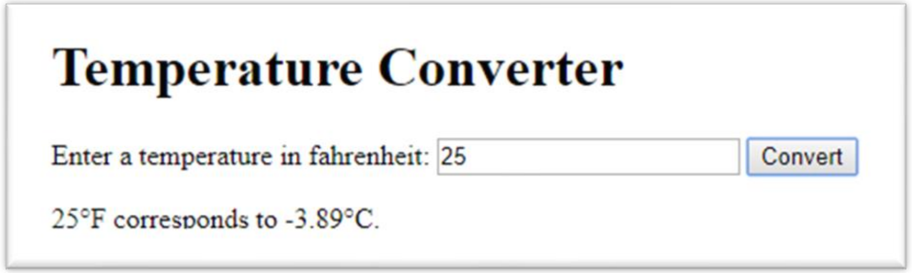
- Multiplication and division have higher “precedence” than addition and subtraction. By precedence we mean that, for example, on a given line, a multiplication operation will be done before, say, a subtraction – regardless of which one appears first on the line.
- Operations that have the same precedence (e.g. addition and subtraction, or multiplicaton and division) are evaluated left to right.
- Parentheses around operations raise the precendence of those operations.

Assignment Operator: Note that the ‘=‘ is an operation! It is called the “assignment’ operator” as we use it to assign a value to a variable. However, this operator has very low precedence. It is almost always the last operation on the line to be executed.

Parentheses: You can use parentheses to force the evaluation order: the innermost parentheses are evaluated first. If you have any doubt about how things will be executed when writing a detailed expression, it often helps to use parentheses.

File: `fahrenheit_celcius_converter.html`

```
<body>
<h1>Temperature Converter</h1>
<form id="conversionForm">
  <p><label for="txtFahrenheit">Enter a temperature in fahrenheit:</label>
    <input type="text" id="txtFahrenheit">
    <button type="button" onclick="convertToCelcius()">Convert</button>
    <!- We *need* the type attribute above! Otherwise our innerHTML won't work. -->
</form>

<div id="results">
</div><!-- end of results div -->

 <script>
  function convertToCelcius() {
    var fahrenheit, celcius;
    var resultsString;

    fahrenheit = document.getElementById('txtFahrenheit').value;
    fahrenheit = parseFloat(fahrenheit);
    //people may well enter a decimal here, so use parseFloat

    celcius = (fahrenheit-32)*5/9;  //Should we parse 'celcius' ??
    //No!  celcius is holding a NUMBER. It is NOT holding a string!
    //Therefore we do not need to parse it.

    celcius = celcius.toFixed(2);   //limit to 2 decimal places

    resultsString = fahrenheit + "&#176;F corresponds to " +
                celcius + "&#176;C.";
    //Note the entity code for the degrees character!

    document.getElementById("results").innerHTML = resultsString;
  }
</script>
</body>
```



Temperature Converter

Enter a temperature in fahrenheit: 25 Convert

25°F corresponds to -3.89°C.

Another Lab Exercise:

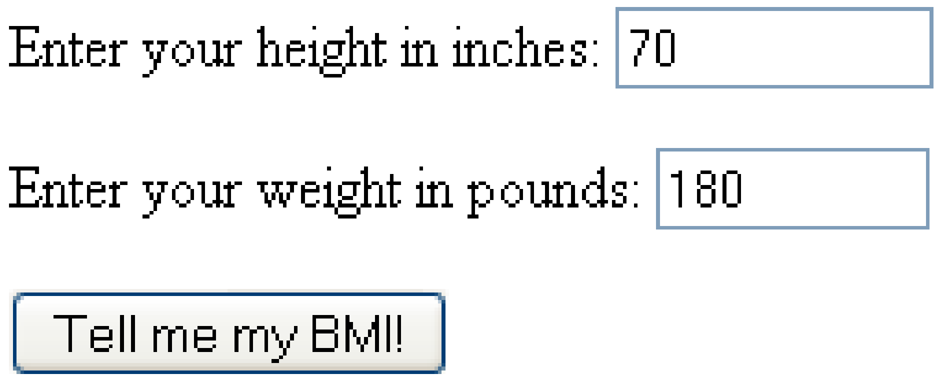
Let’s write a page that calculates the “Body Mass Index” used in fitness circles as a quick (though now mostly debunked) indicator of a person’s health.

The formula requires the user to enter their height and weight and then calculates their BMI.

- **The formula is**: BMI = weight in pounds* 703 divided by height in inches squared.

Try to create this page on your own before looking at my solution

The form:



Enter your height in inches: 70

Enter your weight in pounds: 180

The formula for BMI is:
Weight*703 / height^2

After the user submits the form:

For a weight of 180 pounds, and a height of 70 inches, your BMI is 25.8.

You can see my version in: `bmi_calculator.html`

JavaScript

Predefined Functions

Using the JavaScript Documentation

Learning Objectives

By the end of this lecture, you should be able to:

- Understand the difference between "user-defined" functions and "predefined" (a.k.a. 'built-in') functions
- Understand how to look up and investigate the documentation for predefined JavaScript functions
- Be able to comfortably work with built-in functions, especially:
  - The functions in the `Date` class
  - The functions in the `Math` class
    - `Math.random()`
    - `Math.sqrt()`
    - Be able to look up and apply other functions in this class
  - The `toFixed()` function

'Predefined' (aka 'Built-In‘) Functions

Think back to the user-defined functions we have been creating. For example:

```
function greetUser()
{
  alert("Hello");
}
```

In addition to the infinite variety of user-defined functions that we could write ourselves, JavaScript also comes with many, many 'built-in' or ‘predefined’ functions. The **`parseInt()`** and **`parseFloat()`** functions are examples of such built-in functions.

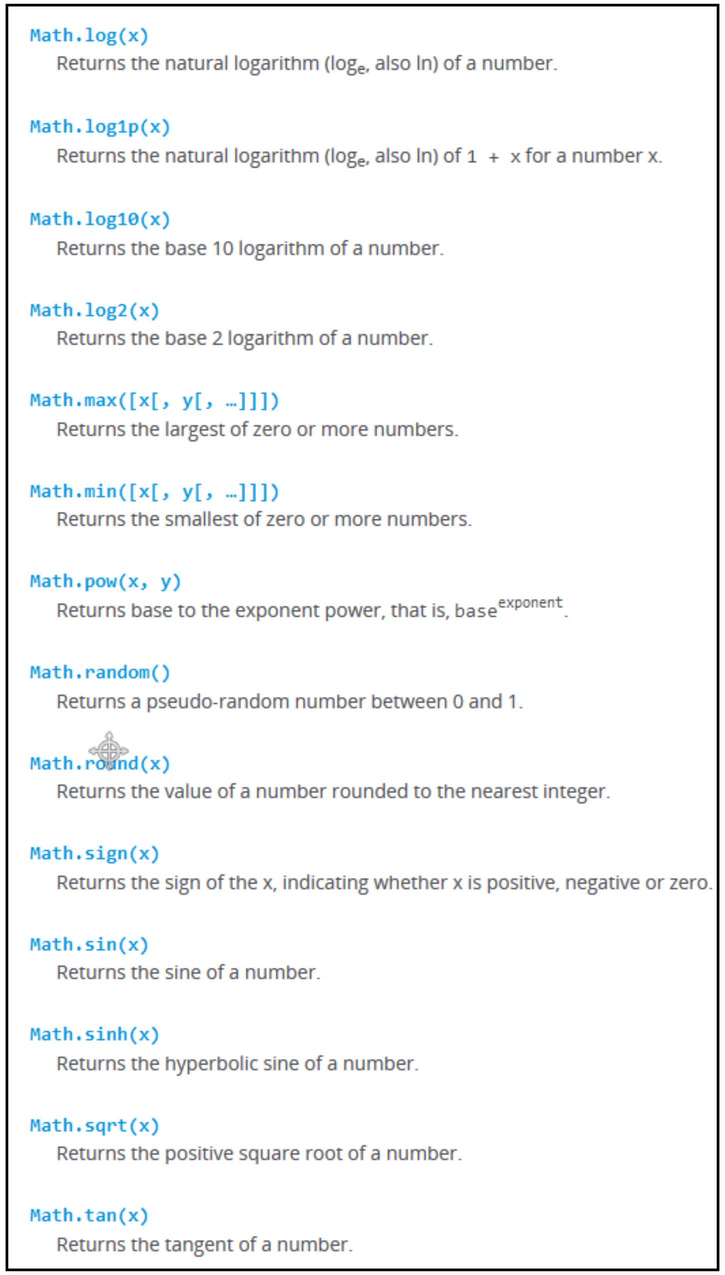
Predefined functions are written by the creators of a programming language in order to solve many common or anticipated coding situations that may arise. For example, the people who created JavaScript recognized that programmers would quite likely need to access the date or time in their code, or that they would need to do various mathematical operations. For this reason, the creators wrote a series of predefined functions to accomplish these tasks.

We have seen and used several predefined functions already. Examples include:

- **`alert()`**
- **`getElementById()`**
- **`Date()`**
- **`Math.sqrt()`**
- **`parseInt()`**
- **`parseFloat()`**
- **`toFixed()`**

Example: Predefined functions in the `Math` class

Here is a partial screen capture from the MDN documentation showing some of the predefined functions that allow us to do various mathematical calculations.



`Math.log(x)`
Returns the natural logarithm ($\log_e$, also ln) of a number.

`Math.log1p(x)`
Returns the natural logarithm ($\log_e$, also ln) of `1 + x` for a number x.

`Math.log10(x)`
Returns the base 10 logarithm of a number.

`Math.log2(x)`
Returns the base 2 logarithm of a number.

`Math.max([x[, y[, …]]])`
Returns the largest of zero or more numbers.

`Math.min([x[, y[, …]]])`
Returns the smallest of zero or more numbers.

`Math.pow(x, y)`
Returns base to the exponent power, that is, $\mathtt{base}^{\mathtt{exponent}}$.

`Math.random()`
Returns a pseudo-random number between 0 and 1.

`Math.round(x)`
Returns the value of a number rounded to the nearest integer.

`Math.sign(x)`
Returns the sign of the x, indicating whether x is positive, negative or zero.

`Math.sin(x)`
Returns the sine of a number.

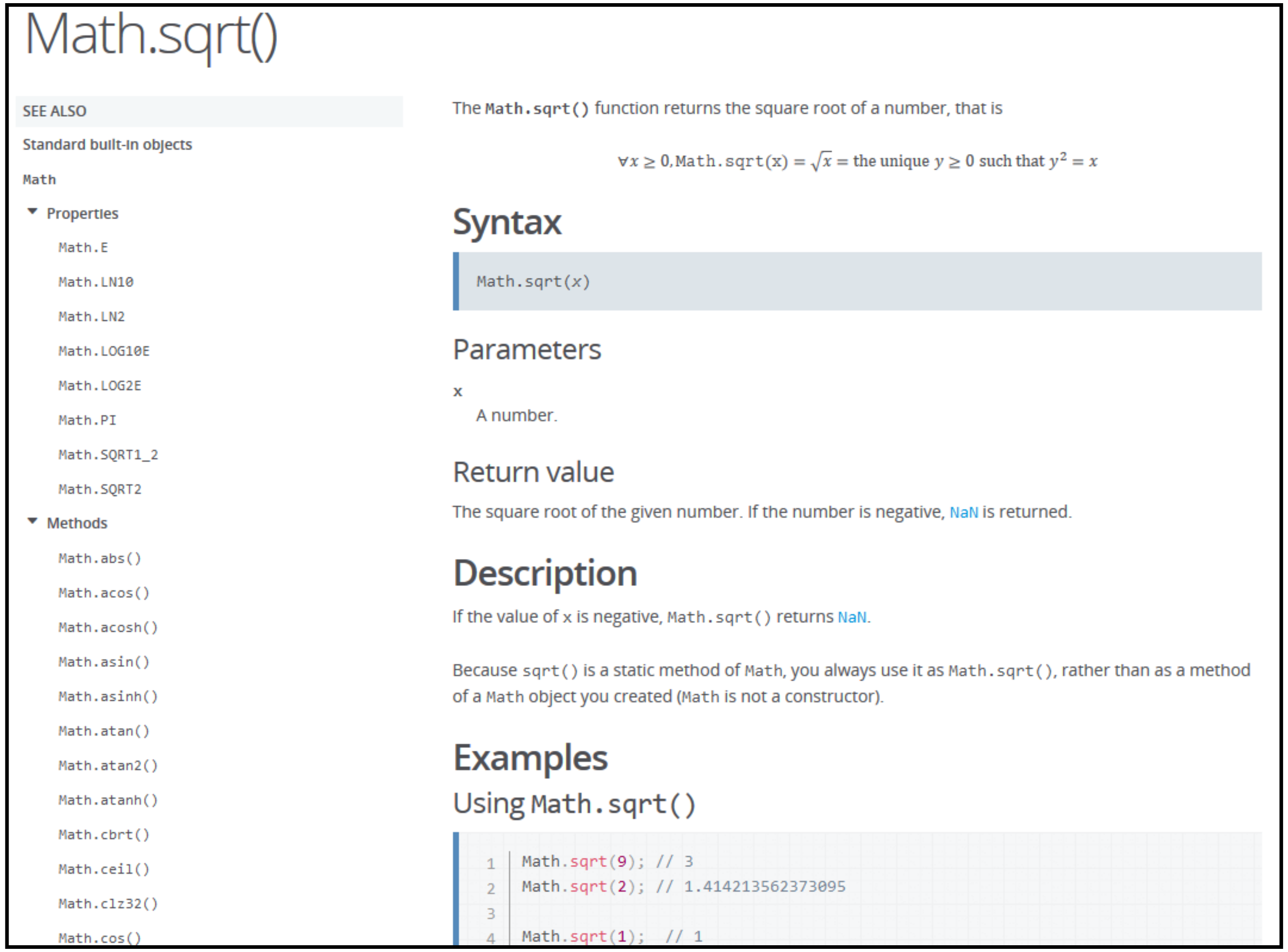
`Math.sinh(x)`
Returns the hyperbolic sine of a number.

`Math.sqrt(x)`
Returns the positive square root of a number.

`Math.tan(x)`
Returns the tangent of a number.

Example: Predefined functions in the `Math` class

Clicking on the `Math.sqrt(x)` function brings us to this page:



Math.sqrt()

SEE ALSO

Standard built-in objects

`Math`

▼ Properties

- `Math.E`
- `Math.LN10`
- `Math.LN2`
- `Math.LOG10E`
- `Math.LOG2E`
- `Math.PI`
- `Math.SQRT1_2`
- `Math.SQRT2`

▼ Methods

- `Math.abs()`
- `Math.acos()`
- `Math.acosh()`
- `Math.asin()`
- `Math.asinh()`
- `Math.atan()`
- `Math.atan2()`
- `Math.atanh()`
- `Math.cbrt()`
- `Math.ceil()`
- `Math.clz32()`
- `Math.cos()`

The **`Math.sqrt()`** function returns the square root of a number, that is

$$\forall x \geq 0, \mathtt{Math.sqrt(x)} = \sqrt{x} = \text{the unique } y \geq 0 \text{ such that } y^2 = x$$

Syntax

```
Math.sqrt(x)
```

Parameters

`x`

A number.

Return value

The square root of the given number. If the number is negative, `NaN` is returned.

Description

If the value of `x` is negative, `Math.sqrt()` returns `NaN`.

Because `sqrt()` is a static method of `Math`, you always use it as `Math.sqrt()`, rather than as a method of a `Math` object you created (`Math` is not a constructor).

Examples

Using `Math.sqrt()`

```
Math.sqrt(9); // 3
Math.sqrt(2); // 1.414213562373095

Math.sqrt(1);  // 1
```

File: `die_roll.html`

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="utf-8">
<title>Dice Roller</title>
</head>
<body>
<h1>Dice Roll Game</h1>
<button type="button" onclick="dieRoll()">Roll the Die!</button>
<div id="results"></div><!-- end of results div -->
<script>
  function dieRoll()
  {
      var die1, die2;
      var resultsString;

      die1 = (Math.random()*6)+1; //die1 holds a float between 1.0 and 6.9999
      die2 = (Math.random()*6)+1;

      die1 = parseInt(die1);
      die2 = parseInt(die2);

      resultsString = "Your first die was a " + die1
          + ", and your second die was a "
          + die2 + ".";

      document.getElementById("results").innerHTML = resultsString;
  }
</script>
</body>
</html>
```

NOTE: Every time you click the button, you are re-invoking the `dieRoll()` function. So each time you click, you will see the results of a new roll of the die. You do <u>not</u> need to refresh the page.

The `Date` object

We haven't discussed "objects" as that is beyond the scope of this course. However, try to follow along with this example and the ones on the upcoming slide(s):

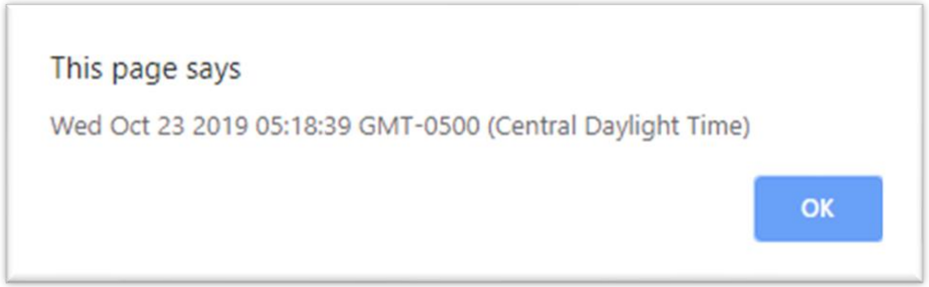
```
var today = new Date();
```

The variable '`today`' is an object that holds all kinds of information about the current date including the current hour, minute, second, month, day of the month, year, etc.

If we were to output the `today` variable like so:

```
alert(today);
```

we would get something like the following:



```
This page says
Wed Oct 23 2019 05:18:39 GMT-0500 (Central Daylight Time)
```

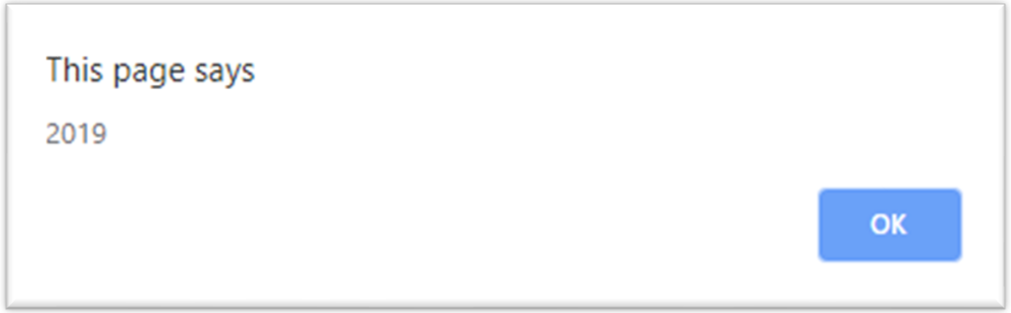
The Date object

Given the `today` object as shown below:

```
var today = new Date();
```

We can invoke various `Date()` functions using our **today** variable like so:

```
currentYear = today.getFullYear();
//currentYear holds the current year, e.g. 2021

currentHour = today.getHours();
//currentHour holds the hour as a number between 0 and 23

alert("The current year is: " + currentYear);
```


This page says

2019

We have just discussed a couple of the functions that you can invoke on a `Date()` object. There are several other useful functions such as `getMinutes()`, `getSeconds()`, etc. etc.

Pop-Quiz: How do you find out about other `Date()` functions that are available to you?

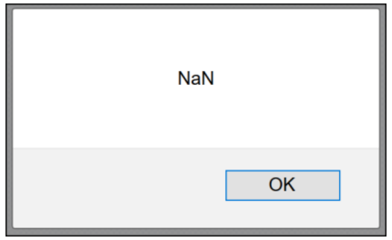
Answer: You look it up in the documentation! Practice by looking up the documentation and trying out some of the other `Date()` functions.

Checking for “NaN” (Not a Number)

Recall that if JavaScript expects a number and does not get one, it will return the error: **`NaN`**

For example:

```
alert( parseInt("hello") );
```



There is a useful predefined JavaScript function that allows you to test for this particular error: **`isNaN()`**

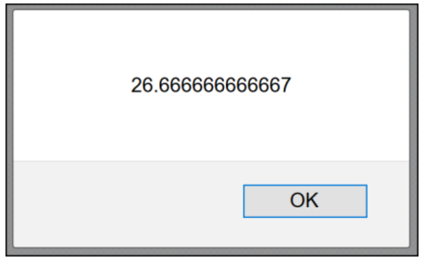
```
var age = document.getElementById('txtAge').value;
age = parseInt(age);

if ( isNaN(age) )
   alert("Please enter a valid age.");
else
   alert("You are " + age + " years old!");
```

File: **`nan_test.html`**

The `toFixed()` function

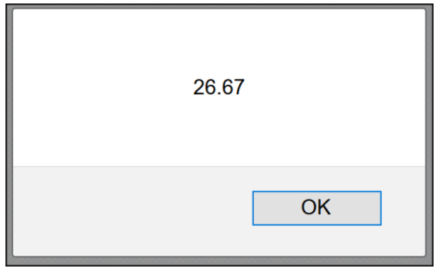
This useful function allows you to specify the number of decimal places displayed by a numeric value in a variable:

```
var number = 26.666666666667;
alert( number );
```



```
26.666666666667
```

```
number = number.toFixed(2);
alert(number);
```



```
26.67
```

JavaScript: Comparisons and Conditionals

Comparing for equality

Logical And/Or

Learning Objectives

By the end of this lecture, you should be able to:

- Describe the difference between the assignment operator = and the equality operator ==
- Describe the use and application of the logical ‘OR’ operator: ||
- Describe the use and application of the logical ‘AND” operator: **&&**

Relational operators

Relational operators allow us to test the relationship between two items.

Often this involves numbers (e.g. `age>=65`), but we also may want to compare, say, two Strings.

For example, we may want to check if a user’s password matches the password on file.

Here are some relational operators:

• Inequality	>, >=, <, <=
• Not equal	!=
• Equal	==

We haven’t discussed these last two yet…

Example If the day is a Sunday, tell them they don’t have to feed the meter.
If it is not Sunday, say: “Better get some quarters!”

```
var day = document.getElementById('txtDay').value;

if (day == "Sunday") {
   alert("No need to pay the meter!");
}
else {
   alert("Better get some quarters!");
}
```

Notes:

- Remember that it’s perfectly okay to declare a variable and assign it a value on the same line.
- It’s okay to have the opening brace of a block on the line that precedes the block.
- The closing brace should <u>always</u> be on its own line.

Multiple Conditionals Separated By Logical ORs

Example:

```
if  (day=="Sunday"   || day=="sunday" || day=="Domingo" ||
     day=="Vendredi"  || day=="Sondag")
```

Under what circumstances will this logical expression evaluate to `True`?

Answer:

It is perfectly acceptable – in fact, fairly common – to have *multiple* conditionals inside a logical expression.

In the above example, because all of the conditionals are separated by logical ORs, as long as ONE of the above conditionals is true, the entire logical expression will evaluate as true.

Parking meter example -- slightly improved version:

If the user says that today is Sunday – with upper-case or lower case ‘s’, tell them they don’t have to feed the meter.

If it is not Sunday, say: “Better get some quarters!”

```
var day = document.getElementById('txtDay').value;

if (day=="Sunday" || day=="sunday" )
{
    alert("No need to pay the meter!");
}
else
{
    alert("Better get some quarters!");
}
```

Multiple Conditionals Separated By Logical ANDs (&&)

Also very common are situations in which two or more conditionals must ALL be True:

For example, suppose that in order to be allowed to vote, an individual must 1) be registered, and 2) be at least 18 years old.

In other words, *ALL* requirements must be met. The syntax follows:

```
if  (  age >= 18    &&    registered=="yes")
{
   alert("You may vote")
}
else
{
   alert("You may not vote");
}
```

- As with OR statements, you can have as many conditionals as you need inside the logical expression.

Summary: When separated by logical AND (i.e. **&&**), the rule is that ALL conditionals must be true in order for the logical expression to evaluate to true.

Multiple Conditionals are Perfectly Acceptable

You can have as many conditionals as you like inside the logical expression.

```
if  (age >= 18 && registered=="yes" && citizen=="yes" &&  notFelon=="yes")
{
  alert("You may vote")
}
else
{
  alert("You may not vote");
}
```

Also, recall that when multiple conditionals are separated by the logical and operator, &&, all of the conditionals must be True for the whole logical expression to evaluate to True.

Using braces for 'if' statements

If your 'if' statement has only one line of code, the braces are <u>optional</u>.

```
– if (logical expression)
  {
          statement 1
          statement 2
  }
```

more than one statement – braces are <u>required</u>

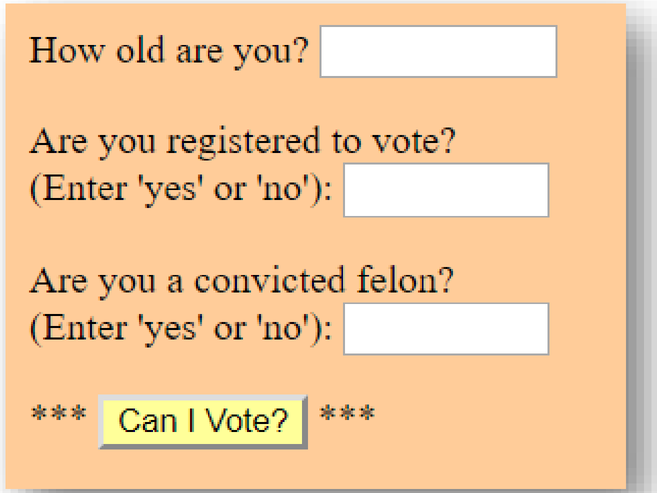
```
– if (logical expression)
    statement
```

one statement only – braces are <u>optional</u>

In general, I recommend that you ALWAYS use braces – even in those cases where your block is only one statement long. Only once you get very comfortable, should you use this shortcut.

vote_check.html
(partial)

```
<body>
  <h2>Vote Checker</h2>
  <form id="voterInfo"  class="form-style-classic">
   <p>How old are you? <input type="text" id="txtAge" size="10"></p>

   <p>Are you registered to vote?
    (Enter 'yes' or 'no'): <input type="text" id="txtRegistered" size="8"></p>

   <p>Are you a convicted felon?<
    (Enter 'yes' or 'no'): <input type="text" id="txtFelon" size="8"></p>

   <p>***
     <button type="button" onclick="checkVoterEligibility()" style="background-color:#ffff99;">
       Can I Vote?</button>
      ***</p>
  </form>
<script>
  function checkVoterEligibility()
  {
    var age, isFelon, isRegistered;

    age = document.getElementById('txtAge').value;
    age = parseInt(age);

    isRegistered = document.getElementById('txtRegistered').value;

    isFelon = document.getElementById('txtFelon').value;

    if ( age>=18 && isRegistered=="yes" && isFelon=="no")
    {
      alert("Congratulations, you may vote!");
    }
    else
    {
      alert("I'm sorry, you are not qualified to vote.");
    }
  }
</script>
</body>
```


How old are you?

Are you registered to vote?
(Enter 'yes' or 'no'):

Are you a convicted felon?
(Enter 'yes' or 'no'):

*** Can I Vote? ***

JavaScript: ‘`else if`’

and coding efficiently…

Learning Objectives

By the end of this lecture, you should be able to:

- Describe how the JavaScript flow operates with **`if`** and **`else if`** statements
- Understand with examples the concept of being efficient in your coding

Cascading if statements: if, and <u>else if</u>

- The **`if / else`** examples we have been using so far have involved only two possible scenarios, that is, a situation that has only one of two possible results. For example:
  - Pay the meter or not?
  - Pay overtime or not?
  - Qualified to vote or not?

- In the real world, we frequently have <u>multiple</u> possible scenarios we want to evaluate for.

- **Example – Assigning a grade:**
  - if the percent grade is 90 or above, assign ‘A’
  - if grade is 80 or above, and less than 90, assign ‘B’
  - if grade is 70 or above, and less than 80, assign ‘C’
  - if grade is 60 or above, and less than 70, assign ‘D’
  - if < 60 assign ‘F’

```
var percent, letterGrade;
percent = document.getElementById('txtPctg').value;
percent = parseFloat(percent);

if (percent>= 90)
{
   letterGrade = "A";
}
else if (percent>=80 && percent<90)
{
   letterGrade = "B";
}
else if (percent>=70 && percent<80)
{
   letterGrade = "C";
}
else if (percent>=60 && percent<70)
{
   letterGrade = "D";
}
else if (percent>=0 && percent<60)
{
   letterGrade = "F";
}
```

Remainder of the program continues….

How it works:

- As soon as any one of the conditionals is evaluated as being TRUE, the JavaScript interpreter will execute the block that follows.
- Once **any of block has been executed,** the program will skip ALL of the remaining `else if` statements.
  - If there is an `else` statement at the end, that block will also be skipped.
- Note that there is a space between the word '`else`' and the word '`if`': `else if`

Is an `else` block necessary?

- Sometimes we need or want an `else` block, and sometimes we do not. It depends on the situation.

- **The else block gets executed ONLY if all of the prior logical expressions are false.**

- However, the moment any block gets executed, flow will jump to the end of the entire block of `if / else if / else` statements (as discussed previously).
    - Again, this includes the '`else`' block.

```
if (percent>=90)
{
   letterGrade = "A";
}
else if (percent>=80 && percent<90)
{
   letterGrade = "B";
}
else if (percent>=70 && percent<80)
{
   letterGrade = "C";
}
else if (percent>=60 && percent<70)
{
   letterGrade = "D";
}
else if (percent>=0 && percent<60)
{
   letterGrade = "F";
}
else
{
   alert("Percent grade must be
          greater than 0.");
}
```

JavaScript:

Limiting Free Text Entry

Learning Objectives

By the end of this lecture, you should be able to:

- Understand and describe why it’s a good idea to limit free-text entry by users
- Apply various techniques for limiting free-text entry

Limiting / Avoiding Free-text Entry by Users

When visitors to your web page are allowed to type in data on their own, there is tremendous opportunity for bugs – and even hacking. For example, hackers have successfully entered JavaScript code into text fields and used that code to gain entry to "secure" systems and do tremendous damage.

On a more benign level, imagine you want the user to tell you the day of the week (recall our parking meter example). Will they enter Sunday, sunday, SUNDAY? What about typos? Ssunday, Sunda, etc, etc.

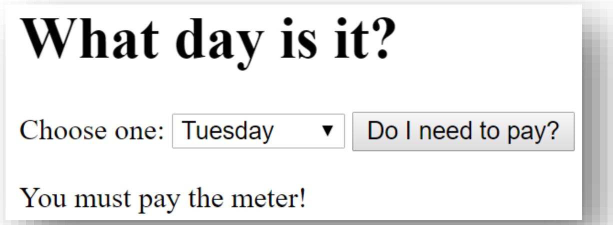
This is why web designers will, <u>whenever possible</u> take the option for entering text *out* of the user's hands. There are many techniques for doing so including the use of

- ✓ Select boxes (e.g. for day of the week)
- ✓ Check boxes (I agree/Disagree)
- ✓ Radio Buttons (rate from 1 to 10)
- ✓ Calendar widgets (for dates)
- ✓ Etc.

Generally speaking, you should only allow the user to enter text where it is absolutely required. Some examples:

- Name
- Email Address
- Address
- Phone number

Eliminating free-text entry



You must pay the meter!

Recall our parking meter example in which we asked the user what day or the week it was. We then had them type the day of the week into a text box.

Now that we know that this is something to avoid, let's redo the example but instead of using a text field, we will use a select box.

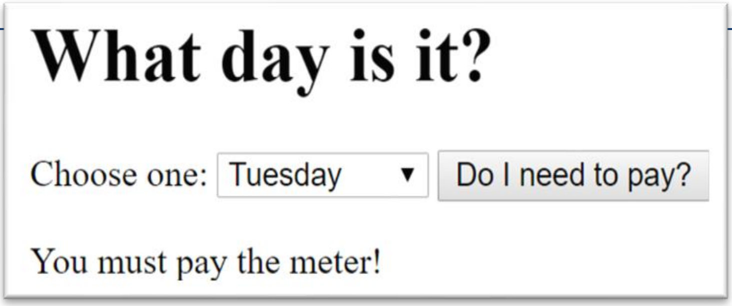
Here is the code for the web page itself. Before looking at my solution, try to convert this into a working program in which if the day is Sunday, you tell the user to pay the meter. If it is not Sunday, you tell them that they do not have to.

My version of the solution (which may well differ from yours) is on the next slide. But DO work it out on your own before looking at my answer.

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="utf-8">
  <title>Parking Meter</title>
</head>
<body>
<header>
  <h1>What day is it?</h1>
</header>
<main>
  Choose one:
  <select id="selDayOfWeek">
      <option value="su">Sunday</option>
      <option value="mo">Monday</option>
      <option value="tu">Tuesday</option>
      <option value="we">Wednesday</option>
      <option value="th">Thursday</option>
      <option value="fr">Friday</option>
      <option value="sa">Saturday</option>
  </select>
  <button type="button" onclick="determinePayment()">
    Do I need to pay?</button>
<p>
<div id="output_area">
</div> <!-- end of output_area div -->
</main>
<script>
           WRITE YOUR FUNCTION HERE
</script>
</body>
</html>
```

File: parking_meter_select.html

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="utf-8">
  <title>Parking Meter</title>
</head>
<body>
<header>
  <h1>What day is it?</h1>
</header>
<main>
  Choose one:
  <select id="selDayOfWeek">
      <option value="su">Sunday</option>
      <option value="mo">Monday</option>
      <option value="tu">Tuesday</option>
      <option value="we">Wednesday</option>
      <option value="th">Thursday</option>
      <option value="fr">Friday</option>
      <option value="sa">Saturday</option>
  </select>
  <button type="button" onclick="determinePayment()">
    Do I need to pay?</button>
<p>
<div id="output_area">
</div> <!-- end of output_area div -->
</main>
<script>
function determinePayment() {
  var outputString;
  var dayOfWeek = document.getElementById("selDayOfWeek").value;

  if ( dayOfWeek != "su" )
    outputString = "You must pay the meter!";
  else
    outputString = "You do not have to pay the meter!";

  document.getElementById("output_area").innerHTML = outputString;
}
</script>
</body>
</html>
```


What day is it?

Choose one: Tuesday ▼ Do I need to pay?

You must pay the meter!

Where are the braces?
Recall that if the block of an `if` or `else` or `else if` statement is only one statement long, then the braces are optional.

Eliminating free-text entry

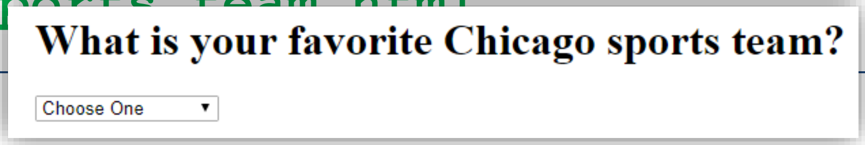
Suppose you wanted to ask the user their favorite Chicago sports team. Will they enter: Cubs, cubs, Chicago cubs, Sox, White Sox, White-Sox, CWS, Hawks, Blackhawks, BLACKHAWKS!, etc, etc, etc.   In other words, we have absolutely no idea what the user may type!

Depending on the team they choose, we output "You are a fan of the Chicago Blackhawks" (or whichever team they selected).

Again, our solution to this problem is to take away the user's ability to enter any free text at all. Instead, we use form elements such as radio buttons, or, in this case, a select box.

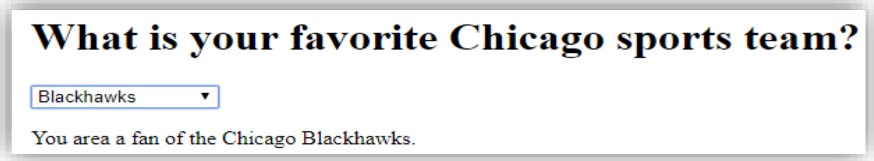
File: favorite_sports_team.html

```
<h1>What is your favorite Chicago sports team?</h1>

    <select id="selTeam"  onclick="checkTeam()">
        <option value="choose">Choose One</option>
        <option value="cubs">Cubs</option>
        <option value="sox">Sox</option>
        <option value="bears">Bears</option>
        <option value="hawks">Blackhawks</option>
        <option value="other">None of the Above</option>
    </select>

    <div id="output_area">
    </div> <!-- end of output_area div -->

<script>
  function checkTeam()
  {
    var outputString = "You are a fan of the Chicago ";
    //We will concatenate onto this string the team that they choose.

    var team = document.getElementById("selTeam").value;

    if ( team=="cubs" )
    {
       outputString = outputString + "Cubs.";  //Concatenate "Cubs." onto the variable 'outputString'
    }
    else if (team=="sox")
    {
       outputString = outputString + "White Sox.";
    }
    else if (team=="bears")
    {
       outputString = outputString + "Bears.";
    }
    else if (team=="hawks")
    {
      outputString = outputString + "Blackhawks.";
    }
    else if (team=="other")
    {
      outputString = "Not a sports fan?";
    }

    if (team!="choose")  //We output ONLY if they chose one of the options from the select box
    {
      document.getElementById("output_area").innerHTML = outputString;
    }
  }
 </script>
```



What is your favorite Chicago sports team?

Choose One

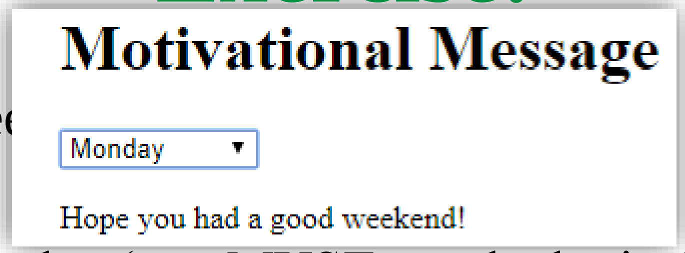


What is your favorite Chicago sports team?

Blackhawks

You area a fan of the Chicago Blackhawks.

Exercise:


Motivational Message

Monday ▼

Hope you had a good weekend!

Prompt the user for the day of the wee

- If they choose Saturday or Sunday (you MUST use the logical or "||" to do this) output: "`Have a great weekend!`"
- If they choose Monday output: "`Hope you had a good weekend`"
- If they choose Tuesday OR Wednesday OR Thursday output: "`Mid-week Blahs`". (Note: Use your OR operators for these three).
- Friday, alert: "`TGIF!`"

- For this program, place the `onclick` attribute inside the `<select>` tag.

You can see my version in: `day_of_the_week.html`

JavaScript

Working with ‘checked’ items

Checkboxes and Radio buttons

Learning Objectives

By the end of this lecture, you should be able to:

- Determine in your JavaScript code whether or not a checkbox was checked.
- Determine in your JS code which button in a group of radio buttons was checked.
- Retrieve a value from a checkbox or radio button.

Retreiving info from form elements that contain ‘text’

Recall the

```
document.getElementById('elementName').value
```

code that we’ve used for textboxes, textareas, and, select boxes.

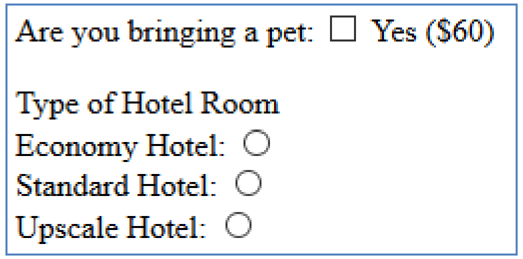
Example: Suppose you have a select box called `selNationality`. To retreive the value you could type:

```
var country = document.getElementById('selNationality').value;
```

The value that gets retreived will be the one you encoded inside the `<option>` tag. Similarly, for text boxes and textareas, the value retrieved is the information that was typed in by the user.

Determining if a checkbox or radio buton is checked

But what if there is no text? For example, what if you are simply working with a checkbox or a radio button and simply wish to know if the button was checked or not?

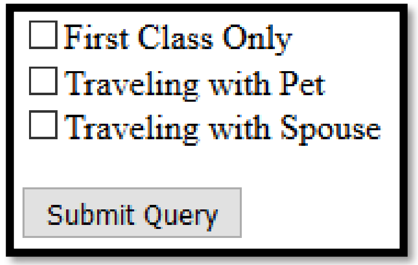

Are you bringing a pet: ☐ Yes ($60)

Type of Hotel Room
Economy Hotel: ○
Standard Hotel: ○
Upscale Hotel: ○

Answer:

We use

```
document.getElementById('element_name').checked
```



```
    <input type="checkbox" id="chkFirst">First Class Only
    <input type="checkbox" id="chkPet">Traveling with Pet
    <input type="checkbox" id="chkSpouse">Traveling with Spouse

    <button type="button" onclick="doStuff()">Submit Query</button>

<script>
function doStuff()
{
    if (document.getElementById('chkFirst').checked)
         alert("Going in style!");

    if (document. getElementById('chkPet').checked)
         alert("Taking your pet!");

    if (document. getElementById('chkSpouse').checked)
         alert("Taking your spouse");
}
</script>
```

Retrieving the value of a radio button

With things like text fields and select boxes, we must ultimately retrieve a 'value'. After all, that is the whole point of those particular types of form elements.

However, with checkboxes and radio buttons, this is not necessarily the case. That is, with things like checkboxes / radio buttons, we often only need to know *if* the box/button was checked.

However, while your radio buttons and checkboxes do not necessarily *have* to have a value, it is often very useful to do so. If your radio button (or checkbox) does contain a value attribute, you can retrieve it using the familiar '**`.value`**' syntax. Again, though, whether or not you choose to include a value is a matter of context in terms of how you set up your page.

For example, in the following radio buttons, we have include a 'value' attribute:

```
What is your favorite color?
RED   <input type="radio" name="favColor" id="radFavRed" value="red">
BLUE  <input type="radio" name="favColor" id="radFavBlue" value="blue">
```

You could retrieve the value of the '`value`' attribute of any of the buttons as follows:

```
var favoriteColor = document.getElementById("radFavRed").value;
//favoriteColor will hold the value "red"
```

Retrieving the value of a radio button

```
What is your favorite color?

RED   <input type="radio" name="favColor" id="radFavRed"  value="red">
BLUE  <input type="radio" name="favColor" id="radFavBlue" value="blue">
```

You could check to see which button was selected,
and then retrieve the value of its '`value`' attribute as follows:

```
if (document.getElementById("radFavRed").checked)
{
  favoriteColor = document.getElementById("radFavRed").value;
}
else if (document.getElementById("radFavBlue").checked)
{
  favoriteColor = document.getElementById("radFavRed").value;
}

alert("Your favorite color is " + favColor);
```

Example: `radio_buttons_with_values.html`

```
 Where are you traveling to? <br>
  <form id="travelInfo">
   New York: <input type="radio" name="city" id="radNy"                      value="New York"> <br>
   Los Angeles: <input type="radio" name="city" id="radLa"                   value="Los Angeles"><br>
   Washington:       <input type="radio" name="city" id="radWash"            value="Washington"><br>

   <!-- Recall that we give all the radio buttons the same name so as to group them together.
    This way, the user can only select one button from this group. -->

  <button type="button" onclick="checkCity()">Go!</button>

  <div id="output">
  </div>
  </form>

<script>
  function checkCity()
  {
    var outputMessage = "You are going to ";
                     var cityChoice;

    if (document.getElementById('radNy').checked)
    {
      cityChoice = document.getElementById('radNy').value;
    }
    else if (document.getElementById('radLa').checked )
    {
      cityChoice = document.getElementById('radLa').value;
    }
    else if (document.getElementById('radWash').checked )
    {
      cityChoice = document.getElementById('radWash').value;
    }

    outputMessage = outputMessage + cityChoice + ".";
                     document.getElementById("output").innerHTML = outputMessage;
  } //end of checkCity function
```

Exercise #2:

- Prompt the user for their income over the past year. Then have radio buttons for the 4 options shown below. Then have a button that says "Calculate my taxes". Calculate their tax based on the following criteria:
  - If the user has an income less than $30K (i.e. $30,000), their tax rate should be 0%.
  - If their income is between $30K and below $50K their rate should be 25%.
  - If their income is between $50K and below $100K, use 35%.
  - $100,000 and more use 40%.
  - Round the result to 0 decimal places.

- Output a result that says something like:

For an income of $64000, you must pay $22400 in taxes.