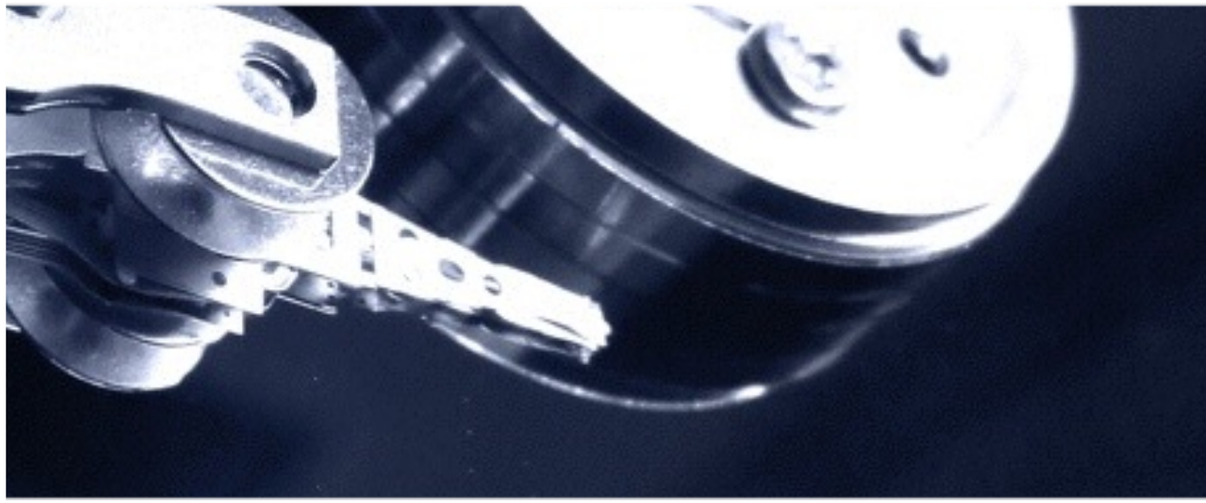
FULL SAIL
UNIVERSITY

# web design and development



## programming for web applications 1

courseMaterial.3

courseMaterial.3

# debugging.**Errors**

# debugging.**Errors**

- debugging.**Tools**
  - always keep a browser debug window open
    - **firefox:** use both ***firebug*** and "***Inspect Element with Firebug***"
    - **chrome:** use both ***firebug*** and ***inspect***
  - **http://wddbs.com/jshero/**
  - **http://www.jshint.com/**
  - **http://www.jslint.com/**

# debugging.**Errors**

- types of errors
  - there are 3 categories of errors:
    - **syntax**
    - **runtime**
    - **logic**

# debugging.**Errors**

- syntax.Errors
  - these errors (also known as parsing errors) occur when the programmer types something incorrectly, such as:
    - forgetting to close a *string* with quotes, or escaping quotes with \
    - forgetting to separate array values with a comma
    - missing other necessary syntax characters such as ( ) , { }

# debugging.**Errors**

- syntax.Errors

```
var fsStudent = {
    age: 22,
    career: “Web Dev”
 ;
```

**Error:** missing } after property list

Source File: file:///Users/msmotherman/Desktop/SWA%20Courses/SWA-1/Lecture%20S

```
;
```

# debugging.**Errors**

- syntax.Errors

```
var fsStudent = {
    age: 22
    career: “Web Dev”
};
```

**Error:** missing } after property list

Source File: file:///Users/msmotherman/Desktop/SWA%20Courses/SWA-1/Lecture%20

```
career: "Web"
```

# debugging.**Errors**

- syntax.Errors
  - syntax errors will usually be displayed immediately ***before*** a script's execution
  - any single script is an individual set of code using <script> tags - the code below indicates two different sets of sequential scripts
  - *note that both scripts are part of the same document, and share variables*

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

# debugging.**Errors**

- syntax.Errors
  - syntax errors will also completely prevent the execution of that **script**'s code *(we'll only see the first syntax error, there could be more to follow)*, however, other scripts will still attempt to run as normal

```
<script type="text/javascript">
    alert("Error here! );
    // this entire script will be skipped because of 1 typo
</script>

<script type="text/javascript">
    alert("I'm error free!");
</script>
```

# debugging.**Errors**

- runtime.Errors
  - runtime errors are exceptions that occur ***during*** the code's execution
  - because these are not syntax issues, they will not cause an error until the problematic line of code is executed
  - once the error occurs however, script execution will stop
  - the most common cause of runtime errors is when a variable or function does not exist (or the reference is misspelled), or when an object being accessed is invalid
    - *most often, if you think your logic is correct, then it is a spelling typo*

**Error:** this.doThis is not a function

Source File: file:///Users/msmotherman/Desktop/SWA%20Courses/SWA-1/Labs/lab2/c

# debugging.**Errors**

- runtime.Errors

```
<script type="text/javascript">
    var myVar = "yay";
    alert(mvar);
    // this line of code will not be run
</script>
```

# debugging.**Errors**

- logic.Errors
  - logic “errors” are the apparent lack of success *(the desired effect doesn’t happen)*
  - this is the most difficult type of error to find - this type of problem does not return an actual error because no syntax or runtime exception has occurred
  - the problem is simply in the programmer’s logic - can you find the mistake?...

```
var myArr = [“1”, 2, true];
for(var i = 0; i < myArr.length; i++){
  if(typeof myArr[i] = “number”){
    alert(“Yay, a number!”);
  };
};
```

# debugging.**Errors**

- debugging techniques
  - use a good text editor or IDE
  - use multiple browsers (they display different error messages)
  - keep the browsers console open at all times
  - use console.log EXTENSIVELY!!!!!!!
  - write JavaScript in external files
  - take a break!

# debugging.**Errors**

- suggestion - using console.log
  - because of the nature of JavaScript, I recommend constant rigorous testing of your code ***as*** you write it
  - test after finishing a *code block that has an impact on the application* - use console.log EXTENSIVELY

```
console.log(value);
```

  - console.log a string to determine if your script is following the program's logical flow

```
console.log("string");
```

# debugging.**Errors**

- example - using console.log

```
var myTest = function(){
  //console.log a string to determine if the script gets into this
function
  console.log ('in myTest Function');
  return "returning.This";
};
var output = myTest();
//console.log the output variable:
//1. to see if the function ran & exited correct
//2. to see if the output is correct
console.log(output);
```

# debugging.**Errors**

- error.Handlers
  - in javascript, we have an additional statement for error handling
    - the **try, catch, throw** statement
    - **try** - use to test a block of code for errors
    - **catch** - use to let you handle the error
    - **throw** - use to let you create custom errors
  - this technique will catch **runtime errors** *only.*
  - the primary purpose is in production-use - when in development mode, it is usually easier to refer to an error console (such as firefox), or use alerts to find problems

# debugging.**Errors**

- try.. catch.. throw..
  - a **try** statement allows you to try out a block of code. If anything causes an exception during that block, the code is aborted, and the **catch** statement block is executed instead

```
try {
  // code to try  throw "error message";
} catch (error) {
  // code to run on error throw "error message";
}
```

  - this can be useful in preventing your end users from seeing errors or disruptions

# debugging.**Errors**

- try.. catch.. throw..

```
try
  {
   //getElementByID will be reviewed later in the course
  var x=document.getElementById("demo").value;
   if(x==="")     throw "empty";
   if(isNaN(x)) throw "not a number";
   if(x>10)      throw "too high";
   if(x<5)       throw "too low";
  }
catch(err)
  {
   var y=document.getElementById("mess");
   y.innerHTML="Error: " + err + ".";
  }
}
```