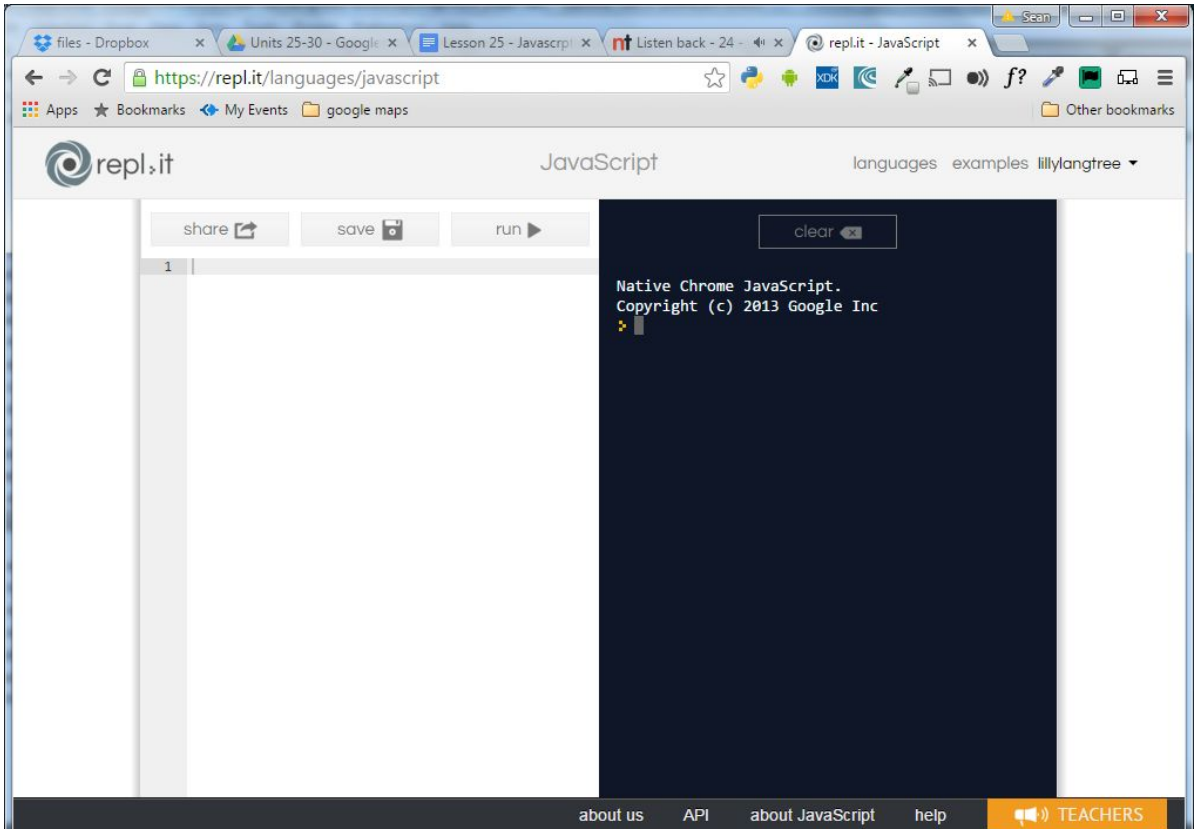
Web Development

Javascript

Javascript Basics



Javascript Basics

The syntax of Javascript is very similar to PHP with a few exceptions

Declaration of variables, no $, use **var** keyword

Javascript runs on the client side **in the browser**

Javascript code is **interpreted** by the browser

Javascript Basics

JavaScript is case-sensitive username and userName are different

In JavaScript, instructions are called statements and are separated by a semicolon (;).

Comments

Comments in Javascript code can be:

```
// a one line comment

/* this is a longer,
   multi-line comment
 */
```

Javascript Basics

Declaring variables

You can declare a variable in two ways:

- With the keyword var. For example,

  var x = 42;

  //note here the = is the assignment operator
  //a single = is not used for equality comparison

Javascript Basics

Data type conversion

JavaScript is a dynamically typed language.

You don't have to specify the datatype of a variable when you declare it, and data types are converted automatically as needed during script execution.

So, for example, you could define a variable as follows:

```
var answer = 42;
```

And later, you could assign the same variable a string value, for example:

```
answer = "Thanks for all the fish...";
```

Javascript Basics

For Javascript the most important HTML tag is **<script>.**

This tag allows us to include a piece of Javascript in a document.

```
<h1>Testing alert</h1>
<script>alert("hello!");</script>
```

Such a script will run as soon as its <script> tag is encountered as the browser reads the HTML.

The page shown above will pop up an alert dialog when encountered.

Javascript Basics

Including large programs directly in HTML documents is often impractical. The **<script>** tag can be given an src attribute in order to fetch a script file (a text file containing a JavaScript program) from a URL.

```
<h1>Testing alert</h1>
<script src="js/myfile.js"></script>
```

The *js/myfile.js* file included here contains any javascript code. When an HTML page references other URLs as part of itself, for example an image file or a script—web browsers will retrieve them immediately and include them in the page.

Javascript Basics

Evaluating variables

A variable declared using the **var** statement with no initial value specified has the value undefined.

An attempt to access an undeclared variable will result in a ReferenceError exception being thrown:

You can use undefined to determine whether a variable has a value. In the following code, the variable input is not assigned a value, and the if statement evaluates to true.

```
var input;
if(input === undefined){
  doThis();
} else {
  doThat();
}
```

Javascript Basics

Type coercion

Depending on the type of operation javascript will attempt to coerce a variable into the correct type. For example, consider the following statements:

```
x = "The answer is " + 42 // "The answer is 42"
y = 42 + " is the answer" // "42 is the answer"
```

In statements involving other operators, JavaScript does not convert numeric values to strings. For example:

```
"37" - 7 // 30
"37" + 7 // "377"
```

Javascript Basics

Javascript Strings

Strings cannot be divided, multiplied, or subtracted, but the + operator *can* be used on them. It does not add, but it *concatenates*—it glues two strings together. The following line will produce the string "concatenate":

```
"con" + "cat" + "e" + "nate"
```

Try this in **repl.it** , note: blank spaces will be honoured .

To output debug statements use the

console.log() function //similar to echo in php

Javascript Basics

Control Structures

```
var x = 10;
for (var i=0; i < x; i++) {
     console.log("loop value:" + i)
}
// loop 0
.
.
// loop 9
// i++ increments i by 1 each loop.
// i-- will decrement
```

Javascript Arrays

JavaScript provides a data type specifically for storing sequences of values. It is called an ***array*** and is written as a list of values between square brackets, separated by commas.

A javascript array can be defined as

```
var myArray = []; //empty array
```

A javascript array can initialized with values.

```
var listOfNumbers = [2, 3, 5, 7, 11];
console.log(listOfNumbers[1]);
// → 3
```

Javascript Arrays

We can determine the length (or number of elements) of an array using the **.length** property

```
var listOfNumbers = [2, 3, 5, 7, 11];
console.log(listOfNumbers.length)
// 5
```

Javascript Arrays

Array iteration

We can interrogate the elements of an array by iterating over it. One way is to use a for loop

```
var myArray = ['one','two','three','four','five','six'];

      for (var num in myArray) {
            console.log(num); //returns the index
            console.log(myArray[num]); //show element
      }
```

Javascript Arrays

indexOf

Access index of an element in array

```
var myArray = ['one','two','three','four','five','six'];
myArray.indexOf('two'); // 1
```

try in **repl.it**

for more on arrays see array reference

Javascript Functions

Functions in Javascript come in many forms. But predominantly we can have

named

or

anonymous functions.

A **named** function looks something like.

```
// define a function
function myFunction() {
   console.log("function called");
   //do something interesting
}
```

is executed

```
myFunction(); //execute a function
```

Javascript Functions

An **anonymous** function is a function without a name. It is generally assigned as a variable value. It can then be executed later, possibly being passed as a parameter to another function.

```
var func = function() {     // anonymous function
        console.log("anonymous function executed);
        }

func();  //execute function
```

Javascript Functions

All functions can accept parameters which are passed inside the () of the function.

```
function myFunc(param1, param2, param3) {
//
    var newValue = param1 + param2;
    console.log(newValue + " " + param3);
}
//
myFunc(12,13,30);   //execute function with parameters
```

Javascript Functions

All functions can use the return keyword to return a value from a function.

Execution will cease at this point.

```
function myFunc(param1, param2, param3) {
//
    var newValue = param1 + param2;
    return newValue + param3;
    console.log("this statement will not be executed");
}
//
var returnValue = myFunc(12,13,30);   //execute function with parameters
console.log(returnValue);             //and return value, store in variable
```

Javascript Control Structures

if...else statement

Use the **if** statement to execute a statement if a logical condition is true. Use the optional **else** clause to execute a statement if the condition is false. An if statement looks as follows:

```
if (condition) {
  statement_1;
} else  {
  statement_2;
}
```

DOM

When a web page is shown in the browser, each of the different elements e.g. Input Field, Text Field, Radio Buttons etc are all considered “Elements”.

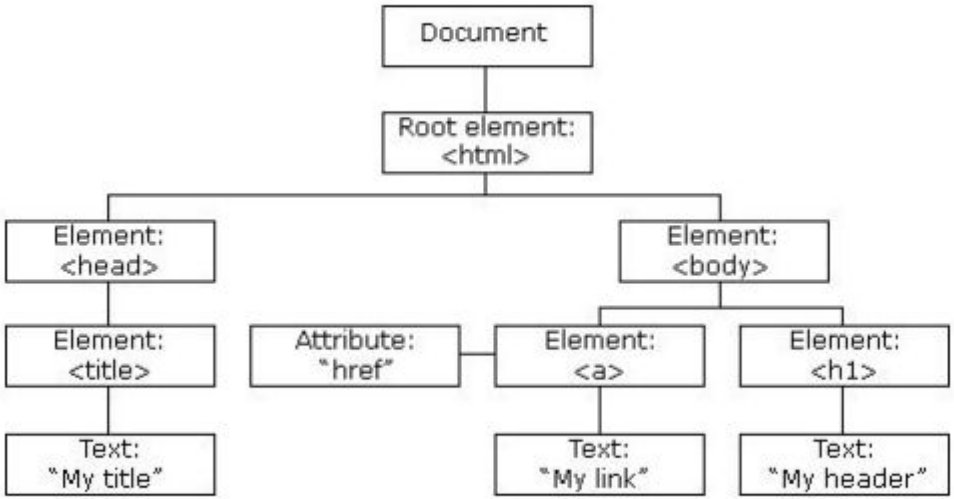
Each of these different elements are added to a Tree type structure called the **Document Object Model** (DOM).

Think about the web page being the root of the tree and each of the elements added to the tree are different branches coming off the tree. Each of the elements has different attributes such as the name, id, value.

DOM

<html>
<head>
<title>My Title</title>
</head>
<body>
<h1>My Header</h1>
my link
</body>
</html>

DOM


Document
Root element:
<html>
Element:
<head>
Element:
<body>
Element:
<title>
Attribute:
"href"
Element:
<a>
Element:
<h1>
Text:
"My title"
Text:
"My link"
Text:
"My header"

DOM

There is a parent/child relationship between elements

In the previous example <h1> is said to be a **child** of the <body> element

<body> is a **parent** of the <h1> element

DOM

In javascript the root element is defined by the javascript object

document.

There are a number of functions we can use to ‘traverse’ or interrogate the document object i.e. elements of the DOM

DOM

If we want, after the web page has loaded, we can pick one element from the DOM tree and check to see what the values current are.

we first start off by using the **getElementById** function. This will go off and find the element in the tree for us .

e.g. ***document.getElementById('id')*** // we pass the id of the element

DOM

The DOM object is a tree based structure.

If we want, we can use JavaScript to add new elements to the DOM tree.

In the example below we are calling **createTextNode** to make a new node. In the second line, we are adding the child node to a div called myDiv.

*var txt = document.**createTextNode**(" This text was added to the DIV.");*
*document.**getElementById**('myDiv').appendChild(txt);*

Javascript Events

When we want a function to be called when something happens on a page, we use event handlers.

These are just like the ActionListener and ActionPerformed in Java.

When a button is pressed we can call a function:

<button onclick='callFunction()'>Submit</button>

DOM

When we call the function document.getElementById it will return is a reference to the element in the DOM tree where that element is.

Once we get this reference, we can then make changes to the element. The **innerHTML** function allows us to set the HTML of an element dynamically.

In this example we are just adding text but we could add any html we like e;g; <p>, , <a> etc,

```
<div id="demo">This is a sample div!</div>
<button onclick="updateDiv()">Update Div</button>

<script>
  function updateDiv(){
    document.getElementById("demo").innerHTML = 'I have updated!';
  }
</script>
```

DOM

```
1  <html>
2  <head>
3  </head>
4  <body>
5  <div id='demo'>
6  This is a sample div
7  </div>
8  <button onclick='updateDiv()'>Submit</button>
9  </body>
10 <script>
11 function updateDiv(){
12   document.getElementById('demo').innerHTML='i have updated';
13 }
14 </script>
15 </html>
```

DOM

Updating Style

```
<div id="demo">This is a sample div!</div>
<button onclick="updateDiv()">Update Div</button>

<script>
  function updateDiv(){
    document.getElementById("demo").style.color = "blue";
  }
</script>
```

DOM - Value

We can extract the value of an element e.g. input element in a form using the

value method

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

DOM - Validation

Execute function on submit

```
<form id='form1' action='login.php' method='post'  onsubmit="return validate()">
                username:<input type='text' name='username' required><br/>
                email:<input type='text' name='email' id='email'><br/>
                <input   type='submit' value='Submit'>
</form>
```

```
function validate(){
        var test = document.getElementById('username');
        if (test.value == "") {
                alert("Please enter a value");
                return false;
        }
        else
                return true;
        }
```

Javascript Validation

Return the number of characters in a string:

```
var str = "Hello World!";
var n = str.length;
```

The result of *n* will be:

```
12
```

Exercise:

Amend the validate function to check that username is more than 3 characters

Javascript Validation

```
<html>
<head>
</head>
<body>
<form id='form1' action='jslogin.php' method='post' onsubmit='return validate()'>
username: <input type='text' id='username' name='username' ><br/>
email: <input type='email' name='email'  ><br/>
<input type='submit' value='submit'>
</form>
<script>
function validate(){
  var test= document.getElementById('username');
  if (test.value == "" || test.value.length < 4 ){
     alert("Please Enter Username");
     return false;
  }
  else
     return true;
}
</script>
</html>
```

Javascript Debugging

When using javascript in a browser it's vital that we are comfortable using the browser's developer tools to debug javascript code.

The first port of call should be the console window in the developer tool. This will flag any errors when we refresh our web page. Hopefully there will be helpful messages to allow us to fix our code. Syntax errors etc should show up here.