

JavaScript Programming

A detailed look at some Javascript Applications

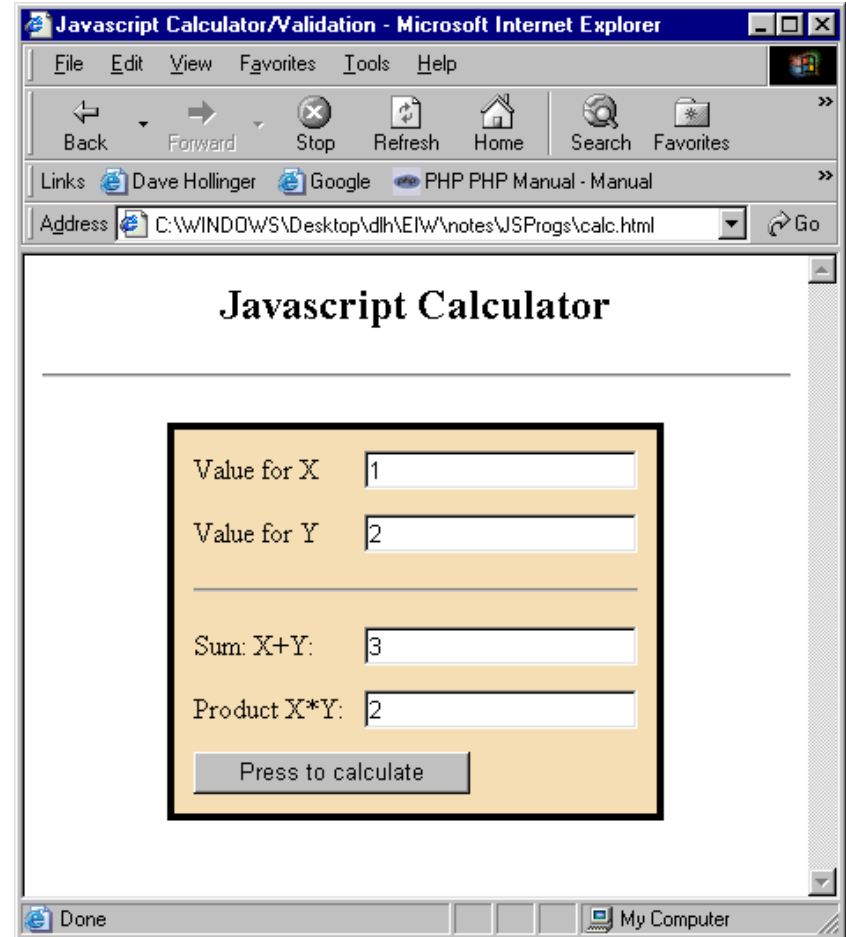
Calculator/Form Field Validation

Animation

Calculator/Form Field Validation

Issues

- Button press event.
- Accessing form field values.
- Dealing with type conversions.
- Formatting output.
- Input field validation.



Creating the HTML Form

id so we can access via `getElementById`

We could make the table pretty by putting it in a table.



```
<form>
Value for X <input type="text" id="x" /> <br />
Value for Y <input type="text" id="y" /> <br />
<hr>
Sum: X+Y:   <input type="text" id="sum" /> <br />
Product X*Y:<input type="text" id="product" /> <br />
<input type="button" value="Press to calculate"
      onClick="calc();" />
</form>
```

Getting the button to run some Javascript

```
<input type="button" value="Press to calculate"  
onClick="calc();" />
```



This is a Javascript Program!

The program calls the function `calc()`, which we need to write!

DHTML Documents

- Documents that include JavaScript often include a section inside the HEAD that defines some global variables and some Javascript functions.
- This code can go anywhere (can be in the body) as long as it's inside <SCRIPT> start and end tags.

Creating the `calc ()` function

- must be defined inside SCRIPT tags.
- will need to get at all the form fields:
 - `x`, `y`: we need to get values from
 - `sum`, `product`: we need to set the values
- we also need to do the math...

Using the DOM to get at form fields

```
// get objects that correspond to our fields
var x_field = document.getElementById("x");
var y_field = document.getElementById("y");
var sum_field = document.getElementById("sum");
var product_field=document.getElementById("product");
```

We now have Javascript variables that represent the form fields:

x_field, y_field, sum_field, product_field

Form fields have lots of attributes, including: **style, id, name, type, value**

Getting x and y values

```
// get the actual values from these fields  
var x = x_field.value;  
var y = y_field.value;
```

x is now whatever the user typed in the first form field.

A form field holds text (not a number). We need to force this to be a number. We could use either of the Javascript functions: **parseInt()** or **parseFloat()**



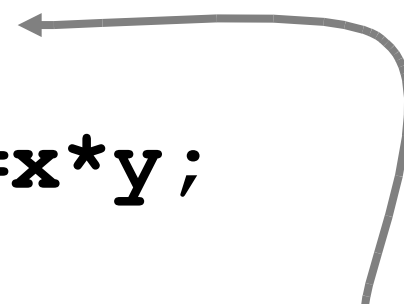
You don't need to write these – they are part of JavaScript

Getting x and y as numbers

```
// get the actual values from these fields
//     assume that each is a floating point number
var x = parseFloat(x_field.value);
var y = parseFloat(y_field.value);
```

Calculating the sum and product

```
// compute the sum and product  
var sum=x+y;  
var product=x*y;
```



What would happen if we didn't make sure that x and y were numbers?

Updating the form with new values for sum and product

```
// update the form so that it shows the results  
  
sum_field.value = sum;  
product_field.value = product;
```

Controlling output format

JavaScript doesn't provide any way to format numbers, so we need to do this ourselves.

Let's assume we want to see no more than 2 digits to the right of the decimal point.

```
// update the form so that it shows the results
// 2 places to the right of the dec. point
```

```
sum_field.value = Math.round(sum*100)/100;
product_field.value = Math.round(product*100)/100;
```

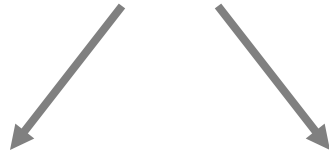
Math object will round to nearest integer

Users can be silly

- What happens if the user says X has the value “Hello”? Example: [calc1.html](#)
- We can avoid this by doing some validation on the values we get from the user.
- Don't do calculations (or submission) until the user fixes things.

Checking x and y

is Not a Number



```
if ( isNaN(x) || isNaN(y) ) {  
    alert("Try giving me some numbers!");  
    return;  
}
```

return means calc() is done!

Complete Example: [calc.html](#)

Submission & Validation

- Forms are typically submitted as a request to a web server.
- It's a good idea to make sure the form is complete before letting the user submit.
 - error messages appear right away (they come from the browser, not the server).
 - use network for valid requests only.

Validation function

- The general approach is:
 - create a function that makes sure all the form fields are filled out correctly.
 - the function can pop up alerts if there are problems.
 - the function returns true if everything is OK.
 - the function returns false if there were any problems (in which case we don't want the form to be submitted).

Intercepting the Submission

- Capture the onSubmit event of the FORM tag:

```
<form onSubmit="return (verify ( ) ) ;" . . .
```



The program here should return true if we want the form submitted, false means that the browser does not submit.

Animation

- There are lots of animation possibilities, the general idea is to use CSS positioning properties to change the location of *something*.
 - an image
 - a paragraph

Moving Text Example

- Changes the position of a paragraph at regular intervals of time.
- Program needs to deal with things like the size of the window and the paragraph, the frequency of movement and the magnitude of each movement.

Complete Example: [animation.html](#)

animation.html issues

- The initial paragraph tag (CSS style)
- Global variables used.
 - make sure things are numbers.
 - browser compatibility issues.
- horizontal movement (function **across**)
- circular movement (function **circle**)
- Scheduling the movements.

Paragraph that will move

```
<p id=txt style="font-family:arial;  
font-size:24pt;  
color:blue;  
position:absolute;  
left:100;  
top:100;  
width:200">
```

```
Moving text</p>
```

Global Variables

mytxt: object that represents our paragraph.

xpos: x position of left side of paragraph.

IE reports units in “px”, example: “100px”

txtwidth: xwidth of the paragraph.

increment: how much to move each step.

speed: how often to make a step (ms/step).

Global Variables (cont.)

`total_width, total_height`: dimensions of the browser window.

IE/Netscape have different DOMs!

`xdirection`: for `across()`, which direction should we move.

Horizontal movement

- at each step the paragraph is moved:
`xpos = xpos + increment*xdirection;`
- The rest of the code just checks to see if we need to change `xdirection` (to change which direction the text moves).
 - did we bump in to the edge of the window?

Circular movement

- Global variable angle is updated each step.
- x,y position computed from angle.

Scheduling the movements

- `window.setInterval (prog , time) ;`
- Need to be able to shut down:
`clearInterval () ;`