GetSET 2009 JavaScript Workshop

Welcome to the GetSET Javascript workshop!

EXERCISE 1: Who Uses Javascript?

We'll start out by searching the net for examples of Javascript. Go to some of your favorite web pages, and **View->Page Source**. Do they use any Javascript? Look for **<script type="text/javascript">**.

Can you tell anything about how they're using Javascript, or what the Javascript is doing?

EXERCISE 2: An HTML Page

Find the GetSET exercises folder on your computer:

Go to **Start->My computer**, find **C:** there and double-click it.
Go into **getset**, and look for **javascript-getset**. Drag that folder to your desktop.

Then go to the folder you just dragged to the desktop, and open it.
Keep this window up – you'll be using it throughout the day.

Look for a file called **first-page.html**. That's the file you'll be using for Exercise 2.

To open a file in Wordpad:

Right-click on the icon and select **Open with...**
Then choose **Wordpad** from the menu of choices.

How to test an HTML web page

To test an HTML file, you need to open it in a browser.
We'll be using **Firefox** as our browser today. If it's not already running, start it now.

Then find your file in the exercises folder and drag that file into the Firefox window.

Once you've loaded a file in Firefox, you can test it again by clicking Firefox' **Reload** button.

Customize your page

Change the file to show your name and your favorite color. You can change other things in the page too, if you want. To test your changes, save the file in Wordpad, then go to Firefox and click **Reload**.

EXERCISE 3: Your First Javascript Page

Look for a file called **first-js.html**. That's the file you'll be using for Exercise 3.

Open the file in Wordpad.

Find the Javascript: it's the part between **<script type="text/javascript">** and **</script>**.

Find the **alert** line, and remove the // at the beginning of it (uncomment it).

Save the file to disk! **File -> Save**.

How to test a Javascript program

Testing your Javascript program is a lot like testing your web page in the last exercise.: you need to open it in a browser.Find your file in ***exercises*** and drag that file into the Firefox window.

Once you've loaded a file in Firefox, if you make more changes, test it again by clicking **Reload**.

EXERCISE 4: Change Your Alert

For this exercise, go back to the Wordpad window.
Change the text inside the alert to say something else. It can say anything you want!

Just be sure it starts and ends with " (double quotes).

Remember to S**ave** the file after you've changed it!

Since Firefox is already pointing at your page, to test it, just click **Reload**.

EXERCISE 5: The Error Console

Open the Firefox Error Console: **Tools -> Error Console**.

If it's full of errors, press **Clear.**

Try typing some commands in the text area at the top.

Try a math command, like **2 + 3**
Or try an alert, like **alert("Hi there!");**

Now try typing some lines that are definitely errors, such as:

```
alert("Hi!);
```

What happens?

Why is this an error? Do you see a problem with it?

EXERCISE 6: Prompt for Background Colors

Find **change-color.html** in the exercises folder, and open it in Wordpad.

Find the place where *newbackground* is set to yellow. Replace that line with a prompt to ask the user what color she likes.

NOTE: Web browsers don't understand that many colors. Stick to simple ones like red, blue, orange, not complicated ones like burnt sienna or chartreuse. But sometimes you can add "light", e.g. lightblue.

Try some and see – nothing bad happens if you choose a color the browser doesn't know about.

EXERCISE 7: Using Array Loops to Change Page Color

Find **colorloop.html** in the exercises folder, and open it in Wordpad.

This has an array called **colors**[] already defined, but it only has a few colors in it. Add some more colors there. (Keep them simple, like “red” or “yellow”, not “chartreuse” or “burnt sienna”.)

Then add a **for()** loop (see the example in the quick reference) to loop over the color list and write one

line for each color in turn. You can call the writeInColor() function, and pass in the string you want to write, and the color: e.g.

```
writeInColor("here is a string", color);
```

Tip: If you include "
 at the end of your string, you can have each color string on a line by itself. br stands for "line BReak".

Bonus: If you finish that and have time left over, try writing the color name as part of the string, e.g. instead of writing "Here is a string", write "Here is a green string".

EXERCISE 8: Change to Different Flowers

Open **growflowers.html** and try running it. Pretty boring, with only one flower, right?

Your task is to make it change to a different flower each time.

Suppose, instead of just f*lower1.gif*, you also have files named f*lower2.gif, flower3.gif* all the way up to *flower10.gif*. (They're in *exercises\flowerpix*.)

How would you change the code in the *addFlower()* function so that it changes to a different flower each time you click?

The trick is to keep track of which flower you're on, with *whichflower*. It starts at 1, then goes to 2, etc.

You do that like this:

```
whichflower = whichflower + 1
```

Once you know which flower you're on, *flowerimg* needs to be set to the right thing. So instead of

```
flowerimg = "flowerpix/flower1.gif";
```

you need to say

```
flowerimg = "something" + whichflower + "something else";
```

Can you figure out what the "something"s are?

EXERCISE 9: Go back to the beginning with "if"

You might notice that when it gets to 10, the *growflowers* program runs out of flowers. You only have 10 flower images, but it draws flowers 20 times on the page. How can you fix this, using an **`if`**?

EXERCISE 10: Make the flower jump

Open **flowerjump.html**.

Your task is to write the code that figures out whether the user clicked on the flower. If they click anywhere else, you don't want to do anything.

The important code is inside `handleClick()`. (It says Ex 9, not 10 – don't worry about that.)

The click happens at a specific (x, y) coordinate on the page: you can get that coordinate with `event.clientX` and `event.clientY`.

The flower is also at a specific (x, y) coordinate on the page. You can get the coordinates of the top and

left of the flower image with:

```
var flowerLeft = parseInt(img.style.left);
var flowerTop = parseInt(img.style.top);
```

(this part has already been done for you).

Your job is to compare the two sets of coordinates to find out whether the mouse click was in the right place – was the click somewhere where it would be more or less inside the flower? You can assume that a flower is roughly 200 pixels high and 200 pixels wide – it doesn't matter if that isn't exact as long as you get fairly close.

BONUS EXERCISE 11: (If there's time)

Make a copy of your *flowerjump.html* file – maybe call it **jump.html**. Your job is to change it so that it uses an array of image names.

Define an array with flower names like "flowerpix/flower1.gif". Use an array like the color array you had in Exercise 7, *colorloop.html*, but call it **pictures** instead of colors, and instead of having colors like "orange", you'll have the names of the pictures, like **"flowerpix/flower4.gif"**. You don't need to use all the flower pictures – you can start with an array that only has two or three. It should look something like this:

```
var pictures = new Array("flowerpix/flower1.gif", …);
```

Then you'll need to change whichflower by removing the **+ 1** at the end, like this:

```
var whichflower = parseInt(Math.random() * pictures.length);
```

By saying `pictures.length`, you're telling javascript to figure out how many pictures are in your array, so you don't have to use exactly 10.

Then you can pick a picture from the array by doing this:

```
var flowerimg = pictures[whichflower];
```

What's the advantage of doing it this way? One big advantage: you can put links to any picture on the web, not just the flower pictures here.

Try using some pictures of your own! Instead of "flowerpix/flower4.gif", find an image you like on the web, right-click on it and choose "Copy image location". Then paste that into your array (don't forget to put quotes around each web address). You can make your picture array have as many or as few pictures in it as you want.

More Bonus Exercises
(if there's extra time, or if you want to try them at home)

Number Guessing Game

Open the file called **numberguess.html**.

This is mostly the same as the number guessing program you saw in class, but you have to write two parts of it.

FIRST: *A*sk the user for a number.

SECOND: There is already some Javascript there to check whether the user's number is too low. Your job is to add similar checks for whether it's too high or exactly right.

Bonus: If you finish early and have time left over, think of some ways you could improve your number guessing game. How about making it choose a number between 1 and 100, or asking the user how big a number it should choose?

Make a broom fly across the page

Open broom.html – it displays a broom – and look for moveBroom(). There's an image of a broom on the page (it's just a starter broom, not as snazzy as a Nimbus 2000). Can you make the broom fly from right to left across the page? Use the same timeout technique that growflowers.html used, but this time you have to write the whole function.

HANGMAN EXERCISES

You have almost all of a Hangman game, but it needs some work for you before you can actually play it. Here's what's left to do:

Add Words to Hangman

With what you know now about programming, make a list of what a hangman game program would need to do. As you work through the exercises, see if you can find the code that does the various things on your list.

Open **hangman.html.**

Find the Word List, and add at least 6 new words to the list. You choose the words!

You can remove the words that are there, if you want.

It's okay to hit return in between words and make several short lines instead of one long line (that makes the list easier to read).

You won't be able to test your work in this exercise, so move on to Exercise 10.

Display Underscores

A Hangman game has to display the right number of underscores, one for each letter in *theword*, the word you're trying to guess. If the word is “fish”, then it has to show _ _ _ _, four underscores.

Your job is to write Javascript in *showWord()* that will do that.

Testing hint: You'll be able to see whether your word has any underscores, but since you don't know which word it picked, you won't know if the number is right. If you want to test that for sure, stick in an alert(myword) just after your Javascript that creates the underscores.

See if the user chose the right letter

In *matchLetter()*, we have to search *theword* to see if the letter the user guessed is in it anywhere. (It might even be there more than once.) Remember, userguess is a variable with a bunch of underscores anywhere the user hasn't guessed the letter yet, so it might look something like “__a_a__t_”.

The Javascript *split* function takes a word and splits it into an array of letters (that part is already done for you). Your job is to examine that array, find places the new letter occurs, and replace the underscore there with the letter.

While you're doing that, if you find a match, set *matched* to *true*. That will tell us whether the user chose a letter that was in the word – otherwise she gets another piece drawn on her hangman!

But wait! There's one more part to this exercise. Down near the end of *matchLetter()*, the program checks to see if the user won. But what if she lost (all 6 parts on the hangman are drawn)? Write some code for that case. Use a **confirm** dialog, just like in the case where she wins, but with a different message, and then call *newGame()* and return the same way as in the case where she wins.

Now you have a complete working Hangman game!