

Chapter 9 Web Experiments

9.1 Getting started: HTML, Javascript, JQUERY

This tutorial makes use of HTML, JavaScript, CSS style sheets, and JQUERY (a useful JavaScript library). All of the code is written using any standard text editor. I use the free program Textwrangler, which automatically color codes your scripts and makes for easy editing. I also use Google Chrome to test and run code using its extensive developer tools for logging errors and command line for running and checking scripts.

HTML files are created by saving your text file with a .html ending. JavaScript files are saved with a .js ending. CSS style sheets are saved with a .css ending.

HTML, Javascript, and CSS work together to accomplish different functions. The HTML file will code the basic layout of the website. Things like text, buttons, and images that you want displayed on the website are coded in HTML. JavaScript is a programming language that works together with HTML. It can run on its own and run any sort of computations that you code, and more

- <http://www.google.com/chrome> - Download Google Chrome
- <http://www.jquery.com> - Download JQUERY
- <http://www.w3schools.com> - General tutorials for all things

importantly it can control the behavior of the website. JQUERY is a free library for JavaScript that makes controlling HTML behavior very easy. Commands from this library will be used to hide and show information during the experiment. CSS style sheets are a part of HTML. They are used to set display properties of text, buttons, and images; for example, font size, color, location etc. All of these properties can be set in the HTML code itself without CSS, but having these properties set outside the main HTML code may help to keep your scripts tidy and readable.

9.2 *Basic HTML*

The `htmlbasics.html` file shows a very simple webpage displaying a few html elements. The code reads:

web

- <http://www.codecademy.com> - Tutorials for learning Javascript and HTML
- <http://www.jsfiddle.net> - Develop and test HTML, Javascript, JQUERY and CSS online
- <http://www.stackoverflow.com> - Q and A forums for programming problems
- <http://www.stackexchange.com>

```

<html>
<head>

</head>

<body>
<!-- this is a comment, not shown
<p>A new paragraph for text</p>
<p id="p1">A new paragraph for tex
<button>A button</button>
<a href="http://www.google.com">A
<br /> <!-- this is a line break -
<p id="p2" align="center" style="c
</body>
</html>

```

Let's look more closely at the code block above. All html documents begin with <html> and end with </html>. This is a general rule for all html statements (although there are some exceptions), they all start with < >, and end with </ >. For example,

```

<p> </p> declares a new paragraph
<button> </button> makes a new but

```

These elements are placed on screen in the order that they are placed in the code. Paragraphs written with <p></p> go to a new line. Each element, whether a button or paragraph, or some other element, has many properties that can

List of common HTML elements

- <p> paragraph
- <button>
- <input> defines input controls
- <textarea> multi line text input
-
 line break
- <div> define sections
- <head> header for javascript
- <body> main page
- <script> embed scripts

change the look of the element and/or add new functionality. As an example, the last paragraph in the block above shows how to center the text, and how to make the font color red.

- `<iframe>`
inline
webpage
- `<canvas>` for
drawing

Important: HTML elements can each be assigned a unique id. This allows the element to act as a variable that can be controlled by Javascript. Notice the second paragraph states:

A new paragraph for text

This paragraph has the id, p1. We will be able to use Javascript to refer to this paragraph and control it's behavior (hide and show, change contents etc.).

9.3 *Javascript Basics*

Open Google Chrome, then choose View > Developer Javascript console, from the menu. Find the command prompt that should appear on the bottom of the web browser, drag up and resize as you see fit. You can now enter Javascript commands into the terminal. You can copy and paste the whole .js file into the terminal and run it. The file has examples of declaring variables, running loops, if/then statements, and writing functions. The following describes these in more detail.

9.3.1 Variables

In Javascript you need to declare variables before you use them. There are two ways of declaring variables. Type the following into the terminal and press enter:

```
a=1;
```

This will declare the variable a, and assign it a number value of 1. Because you have entered an integer, the number will be an integer, if you entered a decimal, it will become a floating point.

You can see the contents of variables by typing their name and pressing enter in the terminal

```
a
```

This should return 1. When you write code in Javascript you normally end each line with a semicolon (;). This tells javascript not to print the line to the terminal. If you declare a variable without the (;) then it will print to the console log.

```
b=1
```

will print 1 to the console. It is best practice to use semicolons at the end of each line.

Another way to declare variables is with var.

```
var c = 1;
```

This is essentially the same as above, except including `var` makes the variable a local variable. It is best practice to declare all variables with `var`, this is especially important when writing functions. It makes the variables local to the function that you are writing. Whenever you want a global variable, do not use `var`.

9.3.2 Variable Types

You can define strings, numbers, integers, decimal numbers, arrays, and objects in Javascript.

9.3.2.0.1 Strings

Strings are declared with straight single or double quotes.

```
var b = "hello";  
var c = "123";
```

Both `b` and `c` are now variables that contain strings. When a number is declared as a string it is treated as character, and not a number. You convert strings to numbers and viceversa. To do this you need to call a function that turns the string into a number. For example:


```
parseInt(c) // turns a number int  
parseFloat(c) // turns a number ir
```



I've used two forward slashes in the above code. These are comments, anything written after the forward slashes is not executed as code. You can use comments to make notes in your code.

9.3.2.0.2 Numbers

```
var d = 1; //creates an integer  
var e = 4.5; //creates a floating
```




9.3.2.0.3 Arrays

Javascript supports arrays. Arrays are variables that contain other variables inside them. For example:

```
var h = [7,4,3,6,5,8];
```

h is now an array. The array has 6 elements inside of it. Each of these elements can be addressed using square brackets.

```
h[0] // will return 7, the first  
h[3] //will return 6, the fourth e
```



The index for the array starts at 0, and this is important to remember. Arrays can hold different types of variables inside of them, for example, numbers and strings:

```
var g = ["hello",3,"test",7,5.56];
```

Sometimes you will want to create an empty array so that you can add to it later. Here's how to define a new empty

array

```
var h = new Array(''); //note thi  
var h=['']; //does the same as a
```

You can easily add new elements to this array again using bracket notation.

```
h[0] = 1;  
h[1] = 2;
```

9.3.2.0.4 Multidimensional arrays

Just as arrays can hold variables, they can also hold arrays. An array within an array is called a multidimensional array. One way to define them is:

```
var i = [[1,2,3],[8,4,5],[8,2,3]];
```

Now the array i has three subarrays. You can address the elements using bracket notation.

```
i[0] // will return the array [1,  
i[0][0] //will return 1, which is
```

You can nest multiple arrays within arrays and have many dimensional arrays. When building experiments we will use arrays to store trial information coding things like trial number, stimulus information, condition information etc.

9.3.2.0.5 Warnings about Javascript Arrays

Array variables are referenced in Javascript in potentially unintuitive ways. For example:

```
A=[1,2,3,4,5]; //create array with  
B=A; //put array A into B  
B[0]=15; //set element 0 of B to 1
```

If you ran this code you might imagine that B becomes a new array that is the same as A. Then, we have replaced the first element in B with 15. However if you now look inside B and A you will see:

```
A=[15,2,3,4,5]  
B=[15,2,3,4,5]
```

The change made to B also changed A. This has to do with how Javascript references arrays. This is important to realize when you start using arrays. This kind of referencing can work to your advantage for certain problems, or to avoid it you can use `concat()`.

```
A=[1,2,3,4,5];  
B=A.concat();  
B[0]=15;
```

Now, if you look inside the variables you will see:

```
A=[1,2,3,4,5];
```

```
B=[15,2,3,4,5];
```

9.3.3 Common operations

Javascript has many built in functions that allow you to manipulate variables by changing them, adding to them, or to address variables and find out information about them.

9.3.3.0.1 Finding the length of variables

.

A common issue is finding the length of a variable or array. You might want to know how many characters are in a string, or how many elements in an array. For this we use .length

```
var aa = "hello world";  
aa.length //returns the number of  
var bb = [1,2,3,4];  
bb.length //returns the number of
```



9.3.3.0.2 Appending to a variable

Another common issue is appending to information to a variable. Javascript has a number of ways of doing this, one simple way is to use +

```
var aa = "hello world";  
gg=aa + " hello"; // gg will contain  
aa+="hello" //this will append hello
```

9.3.4 Loops

Javascript uses for loops to run iterations. The loop takes a number as a starting point, and a number as an endpoint, and finally a number for each increment. The syntax is:

```
var test = new Array('')  
for (i=0; i<=10; i++) {  
    test[i] = i+5;  
}
```

In addition to FOR loops, Javascript supports WHILE loops that continue until a logical condition is met. Look also for the BREAK and CONTINUE statements.

The above line of code defines a new array called test, and initializes it to be empty. The for loop starts with the variable i at 0, it says run until i is less than or equal to 10, and i++ means to add 1 to i for each loop. This statement is contained between () brackets. Next, the code in the loop is contained between { } curly brackets. Here we are simply filling elements in the array test, from 0 to 10, with numbers that equal i+5.

9.3.5 Logic

Javascript logical operators

The syntax for logic statements is similar to that of the for loop. The following example creates a for loop that runs from 0 to 100, and gives different instructions for what to do when the loop is less than 50 and greater than 50.

```
var test = new Array('')
for (i=0; i<=100; i++) {
    if (i<=50) {
        test[i] = i+5;
    }
    else if (i>50) {
        test[i] = i+10;
    }
}
```

When i is less than 50 we add i+5 to the array, but when i is greater than 50 we add i+10 to the array. The basic syntax is to start with if, put the logical statement between () brackets, then put the code to follow for these conditions in { } curly brackets. You can keep adding more else if statements, and you can also use a plain else statement that covers all remaining logical conditions.

- == equality
- != not equal to
- > greater than
- >= greater than or equal to
- < less than
- <= less than or equal to
- && and
- || or

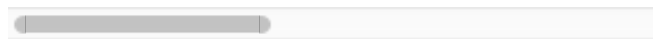
9.3.6 Writing a function

You can write your own functions very easily in Javascript. The following is an example of a function that sums all of the elements of array. Note, this function will only work if the array contains numbers

```
var j = [1,2,3,4,5]; //declares an
```

```
function sumarray(t) {  
    for (i=0; i<=t.length-1; i++){  
        tempsum+=t[i];  
    }  
    return tempsum  
}
```

```
sumarray(j) //returns the sum of a
```



We always start functions with the word function, next is the name of the function, in this case sumarray. You can choose any name for the function. Following the name we put () brackets, and inside these brackets we put the name of our input variable. We use any name for the input, and it generically stands for all possible inputs to our function. In this case I used the variable name t. We will refer to t later in the function. Just like for loops and if/then statements, the body of the function is contained between {} curly brackets. Inside the function is a simple for loop that starts at 0, and goes to the length of the input array (t.length-1). There is a -1 to account for the fact that the first value in the array starts at the 0 index, and the last value is the length-1. Inside the for loop is a new variable called tempsum, here we use += to have the contents of the current indexed element of the array to be added to tempsum. After the loop has finished

running we want to return the sum as output. Simply tell the function to return tempsum.

9.4 Programming a simple Stroop Experiment

This example covers a simple Stroop experiment, which is a good example of most trial-based experiments. Most trial-based experiments present a stimulus on screen during each trial, and require a response to the stimulus. The challenge for coding such an experiment is to display stimuli on screen, record timestamps for the onset of the stimulus, record responses to the stimulus, and record timestamps for the response.

A typical Stroop experiment presents colored word stimuli on each trial, for example the word BLUE printed in blue ink (a congruent trial), or the word RED printed in green ink (an incongruent trial). The task is usually to identify the color, not the word, as fast as possible. A multi-trial experiment would usually have 100s of trials, with the stimuli on each trial randomized. In this case we will be coding an experiment with 96 trials. We will have 50% of the trials be congruent and 50% be incongruent.

We start the logic of the experiment by thinking about the stimuli that we need to create, and the design that we need to implement. We need to choose the colors and words that we will use. Let's use:

red, green, yellow, and blue. With these four colors we have four possible congruent items (RR, BB, GG, YY), and 12 possible incongruent items (RG, RB, RY, BG, BY, BR, GR, GB, GY, YG, YR, YB). Thus, to ensure 50% congruent and incongruent items, we need to present all 12 incongruent at least once, and all 4 congruent stimuli 3 times each. This is a total of 24 trials. If we want more trials, to ensure a balanced design we run trials in multiples of 24 (hence, our 96 trials which is a multiple of 24).

9.4.1 Implementing the design

When you finally run the Stroop experiment you will want several important things to happen.

- have all the trials be presented in a randomized order, or in another order that you prescribe
- have the trial information coding what happened on each trial stored to your data file.

Your data file should contain as much useful information as possible so that you can reconstruct exactly what happened on each trial. For example, for each trial you want to code:

- trial number

- the stimulus that appeared, in this case the word and color that was presented
- condition information, you may want to code the kind of trial that it was (e.g., congruent vs. incongruent)
- timing information about when the stimulus was presented
- the response made to the stimulus, you can optionally encode whether the response was correct
- timing information about the response that was made.

Your data file will be a complete record of what happened during the experiment. When you code your experiment it is very important to think ahead and make sure that your experiment code will create the data file that you want for final analysis.

We can think of the data file as being composed of two parts. The first part codes the design of the experiment, that is what stimuli from which condition gets presented on each trial. The second part codes the results from the participant, the responses that were made to each stimulus on each trial. When coding up the experiment we are basically creating half of a data file. It will be a datafile with

all of the stimulus information for each trial, but will be missing the subject's results from each trial.

We need a logical way of representing the events on each trial, for example consider the table below:

Trial	Word	Color	Condition	OnsetTime	RT	response
1	RED	RED	Con	.	.	.
2	BLUE	RED	Inc	.	.	.
3	BLUE	BLUE	Con	.	.	.
4	GREEN	GREEN	Con	.	.	.
5	YELLOW	RED	Inc	.	.	.

The above table is half of a data-file, it is missing the onset times of each Stroop stimulus (when it appeared on screen), the response times for each trial, and the response key that was struck. When we finally program and run the experiment, we want to be able to save a complete datafile with all of the response information. Before that, however, we can use the first half of the data-file to control how each trial unfolds during the experiment. That is, we can use the information in the data file as part of the program of the experiment.


To run a Stroop experiment you need a computer with a blank screen, then on each trial you need to tell the computer to print a Stroop stimulus to the screen and wait for a response. You can use the first half of the data file to control this behavior. The description of each trial in the data file will instruct the computer to display certain words in certain colors on certain trials.

In plain language, the algorithm would accomplish something like this. You need a variable to act as a counter, this counter will start at 0, and increment by 1 on each trial. On the first trial it will be one, you can then use this information to index the datafile, that is go to line 1 of the datafile and retrieve column 1 for the word information, and column 2 for the color information. You need to figure out how to command the computer to display a “string” on screen, and to tell the computer to display the string that is in column 1 of line 1 of the data file. Then you need to tell the computer to print that string in the color that is contained in column 2 of line 1. On the next trial, the counter will be two, and using the same algorithm as above, the computer will be instructed to display the word and color contained in the columns of line 2 of the data file.

9.4.2 Coding the design in Javascript

The above table can easily be coded directly in javascript using arrays. We can code trials in different rows of the array, and code the parameters for each trial in the columns of the array. A code snippet is below:


```
var Strooptrials = [ ["red","red","  
["green","green","con"],  
["red","blue","inc"],  
["red","yellow","inc"],  
["blue","red","inc"],  
["red","red","con"]];
```



In javascript this is actually called a multi-dimensional array. This is because each line ["green","green","con"], is itself an array with 3 elements. So the above array has 6 lines, or elements, and the inner arrays each have 3 elements. Just like lines and columns. You can address or index the array using square [] brackets. One peculiarity about javascript arrays is that the index begins at 0, so line 1 is really 0, and the first element in an array is 0. The first trial has the word red in ink color red. You can index the array in the following way to access this information:

```
Strooptrials[0][0] // will equal r  
Strooptrials[0][1] // will equal r  
Strooptrials[0][2] // will equal c
```

```
Strooptrials[2][0] // will equal r  
Strooptrials[2][1] // will equal b  
Strooptrials[2][2] // will equal i
```

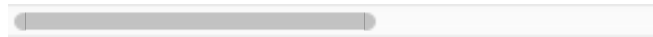


Eventually in the main program we will have a variable that counts the current trials. Say this variable is called i.

```
var i=0;
```

We can use this variable to control the index into the Strooptrials array.

```
Strooptrials[i][0] // will equal t  
Strooptrials[i][1] // will equal t  
Strooptrials[i][2] // will equal t
```



As i gets larger, we will cycle through all of the trials coded in the array.

To make a 96 trial Stroop experiment you would want to create a variable, like Strooptrials, that has 96 lines instead of 6. You could easily do this by typing each line yourself. You could also use Javascript to help you do this. An example is below. This example starts with a Stroop trials variables that has 24 lines, coding each of the unique possible Stroop items that exist by pairing all combinations of red, green, blue, and yellow. The code then uses a function to create a new variable called trials, that allows you to have multiples of 24 trials.

Here is the variable Stroopitems, containing 24 trials.

```

Stroopitems = [{"red","red","con"}
{"blue","blue","con"},
{"green","green","con"},
{"green","green","con"},
{"red","red","con"},
{"blue","blue","con"},
{"green","green","con"},
{"green","green","con"},
{"red","red","con"},
{"blue","blue","con"},
{"green","green","con"},
{"green","green","con"},
{"red","blue","inc"},
{"red","yellow","inc"},
{"red","green","inc"},
{"blue","red","inc"},
{"blue","yellow","inc"},
{"blue","green","inc"},
{"green","red","inc"},
{"green","yellow","inc"},
{"green","blue","inc"},
{"yellow","red","inc"},
{"yellow","green","inc"},
{"yellow","blue","inc"}];

```

```

var trials=randomizearray(createtr

```



The last line creates a new variable trials,
and it uses two functions,
randomizearray, and createtrials.

The createtrials function simply takes an array as input, as well as a multiplier value. Then it returns a new array with multiple copies of the first array specified by the multiplier

```
function createtrials(input, ntimes)
    var temparray = new Array(new
    var iii=-1;
    for (i=0; i<=ntimes-1; i++) {
        for (ii=0; ii<=input.length
            iii++;
            temparray[iii]=input[i
        }
    }
    return temparray;
}
```

9.4.2.0.1 Randomizing an Array

The randomizearray function rearranges the order the elements in an array. We use this to shuffle the order of trials in the trial array, this way all subjects will receive different trial orders.

```

function randomizearray(t){
    var tt= t;
    var n = 0;
    var a = "";
    var b = "";
    var i = 0;
    for (i=0; i <= t.length-1; i++)
        n = Math.floor(Math.random() * t.length);
        a = tt[i];
        b = tt[n];
        tt[i] = b;
        tt[n] = a;
    }
    return tt;
}

```

Running this code results in a new variable called `trials`, it will contain 96 trials, and all of the trials will have a randomized order. Now we really have half of a datafile, all that is missing is the responses from each. To get those, we need to program the main experiment. This involves learning how to control the display of information on screen.

9.4.3 Displaying trials on screen

We are going to program a web-browser to present each trial on screen. This is mildly complicated because in order to do this we need to create an HTML based web-page, and then control this

web-page with Javascript. We will also use a javascript library called JQUERY that allows easy control of the HTML elements.

You can create an html page in any text editor by assigning the file name a .html extension instead of a .txt extension.

Here's a really simple webpage script:

```
<html>
<head>
</head>

<body>

<p>Hello world</p>

</body>
</html>
```

We are going to slowly expand this script to program our experiment. Most everything in HTML has opening and closing statements, and more opening and closing statements nested within. The overarching opening statement is <html> and you will see the last closing statement is </html>. Within this we have <head> </head>, we will insert code in here later to tell the webpage where to look for javascript code. The main webpage is coded between the <body> </body> statements. Inside here we see <p>Hello world</p>. If you ran this in a web browser, you would see the line

Hello world printed to the screen. The `<p></p>` statement declares a paragraph. Anything between `<p>anythinghere</p>` is what gets printed to the screen.


IMPORTANTLY, we can treat paragraphs as variables and give them a name.

Hello world

Now this paragraph has an individual id or variable name called Probe. This is important, because we can use Javascript and JQUERY to control the behavior of this paragraph. We can show and hide the paragraph and change the contents of the paragraph. Our first goal will be to print a Stroop stimulus into this paragraph. Before we do that though, we need to set up Javascript and JQUERY to work with this HTML document.

The following code assumes that your HTML document is located in a folder with the two following javascript files, one for jquery, and one for the creation of Stroop items described above.

```
<html>
<head>
<script type="text/javascript" src
<script type="text/javascript" src
</head>
<body>
<p id="Probe">Hello world</p>
</body>
</html>
```



We use the header to refer to the location of the relevant .js files so that they can be used in our HTML document. These load preexisting javascript files. In our program, we will also write some javascript directly into the HTML document. To do that we include a few more lines in the header as follows:

```
<html>
<head>
<script type="text/javascript" src
<script type="text/javascript" src
<script type="text/javascript">
$(document).ready(function(){
    //our code will go here
});
</script>
</head>
<body>
<p id="Probe">Hello world</p>
</body>
</html>
```

Notice that we have again called `<script type="text/javascript">`, but this time we haven't specified a source file `src=""`. Instead we will write the script directly between the opening statement `<script type="text/javascript">` code here `</script>`, and the closing statement.

Finally, we will write our code inside the document ready function, which ensures that everything is loaded before the page is displayed. The document ready function looks like this from above:

```
$(document).ready(function(){
    //our code will go here
});
```

If we run the above HTML code, we will automatically create the trials variable because we will have called the .js file that executes this code. This means that by the time the webpage loads, the variable trials already exists and can be used. As well, we have already specified a paragraph that can display text and given it a name called Probe. We eventually want the contents of this paragraph to be controlled by the contents of the trials variable. To do this we will use JQUERY. As a first test of the concept, let's write a code that will cause the first word from the first trial in variable trials to be written and displayed in the paragraph Probe.

```
<html>
<head>
<script type="text/javascript" src
<script type="text/javascript" src
<script type="text/javascript">
$(document).ready(function(){
    $("#Probe").html(trials[0]
});
</script>
</head>

<body>
<p id="Probe">Hello world</p>
</body>
</html>
```



All we have done here is added one line of code inside the document ready function. The code is:

```
$("#Probe").html(trials[0][0]);
```

This uses JQUERY, and allows information from javascript variables (e.g, trials[0][0]) to control the behavior of HTML elements. The HTML element is called using the syntax \$("#Probe"), and the \$("#Probe").html() statement allows us to set the html contents of paragraph Probe to the contents inside the () brackets. This should display the first word in the trials array in the paragraph. Notice, if you keep reloading the page, the word that is displayed should keep randomly changing, this is because of the randomizearray function that we use everytime we construct the trials array. This occurs with every page refresh.

To illustrate a few more aspects of JQUERY, like hiding and showing an HTML element, and using buttons in HTML consider the next code.

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

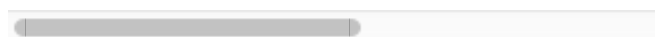
<script type="text/javascript">
$(document).ready(function(){
    $("#showit").click(function(){
        $("#Probe").show();
    });
    $("#hideit").click(function(){
        $("#Probe").hide();
    });
});

</script>
</head>

<body>

<p id="Probe">Hello world</p>
<button id="showit">show</button>
<button id="hideit">hide</button>

</body>
</html>
```



In the body of the HTML we have inserted two buttons, and given each a unique id or variable name. One is called show, and the other hide. Clicking these buttons will demonstrate how to show

and hide a paragraph element in HTML. We do this in the javascript section using JQUERY.

```
$("#showit").click(function(){  
    $("#Probe").show();  
});
```

This is the syntax for allowing a specific button click, from button showit, to execute some code. In this case the code is: `$("#Probe").show()`; which shows the paragraph Probe. The code is the same for the Hide button, with the exception that it causes the paragraph field to be hidden.

The next steps will start to build something closer to a trial based experiment. We will have one paragraph element that used to display the Stroop stimulus, and one button that used to increment the trial counter. Clicking the button will show a new Stroop stimulus each time, simulating going from one trial to the next

```

<html>
<head>
<script type="text/javascript" src
<script type="text/javascript" src
<script type="text/javascript">
$(document).ready(function(){

var counter = -1;

    $("#next").click(function(
    counter++;
    $("#Probe").html(trials[cc
    $("#Probe").css("color",tr
    });
});

</script>
</head>

<body>

<p id="Probe">Hello world</p>
<button id="next">Next Trials</but

</body>
</html>

```



We've added a variable called counter, and initialized it to -1. We added a button called next, and written some code that controls what happens when the button is clicked. When the button is clicked, the trial counter is incremented by 1. The Probe paragraph html is set to

trials[counter][0], which displays the word for the current trials, and the color of the font is set by:

```
$("#Probe").css("color",trials[counter][1]);
```

This line also reads from the trials array to set the color.

Now, clicking button presents a new trial. Next, the data should be recorded. We need to save the onset time of the Stroop stimulus, a button-press for the response, and the response time for the button press. Finally, we need to put these all together, along with the trial information (word, color, condition, trial number) in a data file. We might also implement some control in the script so that when a participant makes a response the next trial is automatically triggered.

Let's begin by recording a response, and the response time, and having that printed to a paragraph onscreen so that we can be sure we are measuring everything correctly.

```
<html>
<head>
<script type="text/javascript" src
<script type="text/javascript" src
<script type="text/javascript">
$(document).ready(function(){

var counter = -1;

    $("#next").click(function(
        trialcounter++
        $("#Probe").html(trials[cc
        $("#Probe").css("color",tr
    });

$("body").keypress(function(event)
d = new Date();
$("#responsetime").html(d.getTime(
});

});

</script>
</head>

<body>

<p id="Probe">Hello world</p>
<button id="next">Next Trials</but
<p id="Response"></p>

</body>
</html>
```

Press the next trials to start a trial, press a key on the keyboard to make a response. When you press a key, the computer will display the key you pressed, and the computer time in milliseconds when you pressed it. This will be printed to paragraph Response below. A note on the timestamps. The time that is recorded in variable d is recorded when new Date() is called. The latter code d.getTime() is a method for outputting the contents of d in millisecond format. Calling for a new Date() is time sensitive, and should be placed immediately at the front of keystroke collection and immediately after stimulus presentation for precise timing.

At this point the major obstacles to programming an experiment have been solved. The design of 96 Stroop trials has been captured and stored in an array. That array is used to determine the words and colors that get printed to the screen on each trial. The example shows how to program a simple paragraph element in HTML to display text, and shows how to use JQUERY to update and the change the contents of the paragraph. Finally, the Javascript code for recording time stamps is covered.

All that is left is organizational and making things look pretty. One issue is to create a final data structure that can be exported after the experiment is completed. This tutorial only briefly covers this final operation as different

researchers will want to code their data structures in different ways for different purposes. The important thing is that you code a data structure that preserves all of the important information, saves it in a file, and can be meaningfully interpreted during analysis at a later date.

9.4.4 Next steps: WebStroopTutorial

This section on programming a simple Stroop experiment comes with several example tutorial files. All of these files are contained in the [WebStroopTutorial](#) repo. These files incrementally build a Stroop experiment. A brief list of the files and notes is below:

9.4.4.1 StroopDemo2.html

File Dependencies: jquery-1.7.1.js, StrooptrialGenerator.js, Stroop.css

New Changes:

- Makes the font bigger
- Centers Stroop stimulus horizontally
- Stroop stimulus disappears after response
- Response automatically triggers next trial with a delay of 500 ms

9.4.4.2 StroopDemo3.html

File Dependencies: jquery-1.7.1.js, StrooptrialGenerator.js, Stroop.css

New Changes:

- Uses CSS style sheets
- Place Stimulus in the center of the window (both horizontally and vertically)
- Make the background Black
- Set many paragraph properties with CSS

9.4.4.3 StroopDemo4.html

File Dependencies: jquery-1.7.1.js,
StrooptrialGenerator.js, Stroop.css

New Changes:

- Include a fixation warning before stimulus onset
- Give feedback after each trial

9.4.4.4 StroopDemo5.html

File Dependencies: jquery-1.7.1.js,
StrooptrialGenerator.js, Stroop.css

New Changes:

- Disable use of Backspace and space keys (general example of disabling default web Browser behavior triggered by pressing keys.)
- Web-browsers have some default behaviors that may interrupt your experiment. For example, pressing the backspace key can cause the browser to return to the previous page. Or press space can cause the browser to page-down. It may be important to override these defaults.

9.4.4.5 StroopDemo6.html

File Dependencies: jquery-1.7.1.js,
StrooptrialGenerator.js, Stroop.css,
StroopDemo6b.html, TURKconsent.htm

New Changes:

- This demo begins covering options you may want for running on AmazonTurk. `StroopDemo6.html` is now an opening page that displays a consent form, and an accept button.
- The opening window also has a form that can later be used to submit data to Amazon. One of the form inputs (called RTs) would normally be hidden, but is shown in the demo at the bottom of the screen. This form will be populated with the datafile, a string of data values after the experiment is complete.
- The pop-up window that runs the main experiment goes full-screen.
- A challenge with using a pop-window is to have the data collected from this window be sent back to the parent window. There is a new submit data button in the pop-up window that gives example code for sending data from the child pop-up back to the parent pop-up.
- This also shows how to present an html document as a frame inside a document. This is used to present the consent form on the opening page.

9.4.4.6 StroopDemo7.html

File Dependencies: jquery-1.7.1.js,
StrooptrialGenerator.js, Stroop.css,
StroopDemo6b.html, TURKconsent.htm

New Changes:

- Added a demographics questionnaire to beginning of experiment window on popup.
- Shows examples of extracting data from HTML elements and forms using JQUERY
- Demographics is saved as part of the data, which is returned to the main window form element input RTs.
- Buttons like submit data, and the RTs form in the parent window are shown to give examples of functionality, these can easily be hidden.

9.5 Amazon Turk

There are several steps involved in running your experiment on Amazon Turk. This is a short guide to get up and running. *Fair Warning*, this tutorial was created in 2014, so some aspects may have changed. For example, Dropbox no longer let's you serve webpages from a public folder (use [github pages](#), or [firebase](#)).

Todd Gureckis and I delivered workshops on web-experiments and Amazon Turk at APS in 2015 and 2016. [We created a website with our slide deck](#), and [links to helpful resources](#)..

Check out:

- [PsiTurk](#)
- [JSPsych](#)

9.5.1 Create an Account

Go to the Amazon Turk website and create an account. To do this you will need an Amazon account. You can use your Amazon account to log-in to all of the Amazon Turk services. There is a verification process, as being a work requester requires that you link an American credit card or debit card to the account so that you can put money in to pay the workers.

9.5.2 Worker site, Requester site, & the Sandbox

Amazon Turk has different websites depending on how you want to log in. As an experimenter you will need to become a work requester. You can also become a worker. On the worker site you will see the hits (jobs) that are currently posted. On the requester site you will have the ability to post and manage your jobs. It is good to have access both as a worker and requester so that as a requester you can check to see that the workers are seeing what you want them to see. Most of the important de-bugging and testing can be accomplished in the Amazon Turk Sandbox. This is basically a mirror-copy of the main worker and requester sites, except they operate in test-run (sandbox) mode. Once you have your experiment programmed and ready to roll, you can first load it up as a requester in sandbox

mode, then you can log in as worker in sandbox mode and test out your experiment. In sandbox mode you can 1) verify that your experiment runs at it should, and 2) MOST IMPORTANTLY, that your script interfaces properly with Amazon so that two things happen, A) the worker can accept, complete, and submit the hit to get paid, and B) your data gets sent to Amazon so that you can later retrieve it. Both of these functions can be tested in sandbox mode prior to loading the hits on the main page.

9.5.3 Basic overview of interfacing with Amazon Turk

The rest of this tutorial assumes that you have a Mac. The basic process will be the same on a Windows machine with slight modifications that are not covered here. First, you need to create a web-based experiment, then you need to be able to host this experiment on a server so that people on the web can access the experiment. If you host on your own server then many more options become available to you. For example, you can have your website send data back to your server as participants complete the experiment.

This tutorial covers a different option. If you have a dropbox account, then you can host your experiment directly from your public folder. Any file in your public folder can generate an external link

allowing anyone with the link to access the file. This allows your experiment to be served to the web. However, this method does not allow client-side information to be passed back to the Dropbox server. All of the data handling will be dealt with client-side, on the workers computer. To accomplish this, your experiment needs to put the data that it generates into HTML form element (which will be a hidden text box). At the end of the experiment the contents of this form element (your data), will be submitted to the Amazon system along with other worker information (such as worker ID, time started, time completed, and other information that automatically gets passed). You will retrieve the data from Amazon's servers.

The process of loading a hit onto Amazon Turk can be accomplished directly from the Turk requester website. This website gives the tools to construct and load simple HITs, things like questionnaires. If you are running a multi-trial experiment using your own website you will not be able to use Amazon's built in tools. Luckily, Amazon provides a rich set of command line tools that allow you to use shell scripts in UNIX to load and manage your HITs, this includes paying workers and downloading your data. The shell scripts can be run directly from the terminal on a mac. If you are on windows there are similar options that are not covered here.

9.5.4 Download the Command Line Tools

The first step is downloading the command line tools provided by Amazon. You access these by clicking on the developer tab on your requester page. You will see a link to download the CLT or command line tools. Click this link, then find the option for UNIX downloads. Download the file. You will get an archive that looks like `aws-clt-1-1.3.0.tar`. Extract the archive. Now you have access to a folder with all of the command line tools. In the main folder there is a `UserGuide.html` and other pages that walk through all of the examples.

9.5.5 External Hit sample application

In the user guide you will want to read up on the External hit sample application, these are tools used to interface your web-experiment with Amazon.

You can find all of the scripts associated with this sample application by looking in the samples folder in the command line tools that you downloaded. You should see a sample called `external_hit`. This folder contains several files. With the exception of `externalpage.htm`, all of these files are unix shell scripts. They are all currently configured to run an example of loading a hit onto Amazon.

A good first step is figuring out how to load this hit into your requester sandbox. If you can get this hit working, then you are one step closer to getting your own hit working. Eventually, we will copy some of the code from the `externalpage.htm` example into your own experiment code, using the same methods as in the example to interact with amazon.

9.5.6 Running the example

1. Amazon will provide you with two access codes. An `access_key`, and a `secret_key`. Find these two codes, you will need them to run the example.
2. In the CLT folder, you will see another folder called `bin`. Inside this folder is a `.txt` file called `mturk.properties`. You will need to edit this file. You can do use in any text editor. A good free one is TextWrangler. You will see `access_key=[insert your access key here]` and the same for `secret_key`. You need to insert both codes into the file. Notice the example in comments above in the file. The codes here do not `[]` square brackets around them. Follow this example. Do not enter the code in between the square brackets. Enter the code without square brackets.
3. In the same file you will see an Advanced properties section. This is where you set whether or not you load your HIT into the sandbox for testing, or the main website. You want to load the example into the sandbox. The

comment symbol, #, controls the behavior. You should see the following:

```
# -----  
# ADVANCED PROPERTIES  
# -----  
#  
# If you want to test your solution in the Amazon Mechanical Turk sandbox,  
# use the service_url defined below:  
#service_url=http://mechanicalturk.sandbox.amazonaws.com/?  
  
# If you want to have your solution work against the Amazon Mechanical Turk  
# use the service_url defined below:  
service_url=http://mechanicalturk.amazonaws.com/?  
  
# The settings below should only be modified if you are having trouble.  
# You should not need to adjust these values.  
retriable_errors=Server.ServiceUnavailable,500  
retry_attempts=6  
retry_delay_millis=500
```

To use the sandbox, remove the # from the line:

```
#service_url=http://mechanicalturk.sandbox.amazonaws.com/?  
Service=AWSMechanicalTurkRequester
```

And add a comment to the following line:

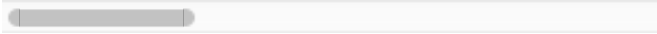
```
service_url=http://mechanicalturk.
```

Your script should read:

```
# ADVANCED PROPERTIES
# -----
#
# If you want to test your sol
# use the service_url defined
service_url=https://mechanical

# If you want to have your sol
# use the service_url defined
#service_url=https://mechanica

# The settings below should or
# You should not need to adjust
retriable_errors=Server.Service
retry_attempts=6
retry_delay_millis=500
```



IMPORTANT NOTE: I have run into an error using this file exactly as written above. The error has to do with the [http://](#) statement. To run without errors, the [http://](#) needs to be changed to [https://](#). Notice, in the above code the [http://](#) has been changed to [https://](#)

9.5.7 Running the example in the terminal

The scripts provided by Amazon in the `external_hit` folder are run in the terminal on a mac. There are several scripts in the folder and each accomplishes different

jobs. The scripts are currently configured to run the `external_hit` example. To do this you will use the `run.sh` script.

1. Open the terminal
2. navigate to the directory that contains the `external_hit` folder, go into the `external_hit` folder
3. run the `run.sh` script, type:

```
./run.sh
```

Press enter. If everything has worked, your terminal will contact Amazon and load up the `external_hit` example onto your sandbox. The terminal ought to return a direct link to this HIT in the sandbox. You can copy and paste it into your browser to view the hit online.

Using the sandbox as a worker you should find and complete the hit. You will see the amazon interface that the workers see, and this will show you how the worker finds, sees, and interacts with your HIT.

When you click on the HIT to view it, you will see an accept hit button. You can complete the hit, and then submit it to amazon.

9.5.8 Getting results

After you have completed the hit, you can use the terminal again to download the data. Using the same steps as above you should run the `getResults.sh` script in the terminal. This will download the results into a file called `external_hit.results`, which is contained in the `external_hit` example folder.

9.5.9 Approving and deleting

If this was a real hit, then you would need to approve the job to pay the worker, and delete the hit from amazon. You can do this in the terminal by running `approveAndDeleteResults.sh`. This command also works in the sandbox, and will pay you fake money for your hit as well as delete the hit from Amazon. If you do not make a habit of managing your HITs on Amazon then it will become very cluttered and you may accidentally fail to pay workers.

9.5.10 General points about the example

If you ran the example, you will see that you can complete the hit from within the amazon interface. The amazon interface loads the external page as an inline frame. If you directed Amazon to your own webpage instead the example, then your webpage would be loaded in the

frame. As a quick side note, the external_hit.question file is where you tell amazon which webpage to load. If you look at this file you will see that amazon is hosting the webpage (a copy of which is in your external_hit folder). For example, the file reads:

```
<ExternalQuestion xmlns="http://me
    <ExternalURL>http://s3.amazona
    <FrameHeight>450</FrameHeight>
</ExternalQuestion>
```

You would replace the external URL reference with your own website, as in:

```
<ExternalQuestion xmlns="http://me
    <ExternalURL>http://www.yourwe
    <FrameHeight>450</FrameHeight>
</ExternalQuestion>
```

You can look at a copy of the externalpage.htm file that amazon is hosting on it's own server, it is in your external_hit sample folder. This file contains javascript code, and html forms that show how to interact with Amazon. This is very important code.

1. The code determines whether the worker has accepted the HIT
2. Once the HIT has been accepted, the code submits data back to amazon, and tells amazon that worker

completed the hit.

The important features of the code are:

1. the `gup` function

This appears at the top of the `externalpage.htm` file. The function is written in javascript. In your own code you will have a section in the header where you have your own javascript, and this is where the `gup` function should be placed. This function allows the website to information to determine whether the HIT has been assigned to a worker.

2. the `mturk_form`

In the body of the HTML you will see an HTML form, the id is `mturk_form`. This form allows users to input information, and then submit that information back to amazon. In the example you will see a question, what type of webpage is shown below? Then some radio buttons, then a submit button.

Also notice, if you are viewing the page in the sandbox and haven't yet clicked accept hit, then the submit button that is part of the form will display the message "You must accept hit...". If you accept the hit, then this message goes away, and you can submit the hit. The javascript controlling this behavior is at the bottom of the html file.

9.5.11 Data handling

The data from the hit is sent back to Amazon through the `mturk_form`. Any input that is in the form will get sent back. If you add form elements, or take away form elements, then you will add or take away from the data that you will see in your results file.

When you run the `getResults.sh` script you will return the `external_hit.results` file. The data need to be post-processed. The first line in the data field contains headers for each of the data values. These are like column names in a spreadsheet. The main exception is that each name is enclosed in “ “ quotes. So, you will see something like this:

```
“hitid” “hittypeid” “title” “descr
```



Each line that follows is the data from a single hit. Each of the data values are also enclosed in quotes and separated by tabs. You could copy and paste the whole thing into excel if you wanted to, and it would fit neatly into the rows and columns.

Each line is very long containing information from multiple variables. The data that you want returned from the hit will be near the very end, or the last quoted item in the line. Any element in the `mturk_form` that is submitted will be included in the data line as its own quoted item.

There are probably many options for sending and receiving data through Amazon. The direction that I chose was to have my web experiment compile all of the data from the experiment into a single line, and then put this single line of data into the `mturk_form` as a form element. This way the built-in form sends the data back using the code that Amazon provides. This long line of data will be in string format, that is just a long list of characters. Data in this format will need parsing later for final analysis.

The strategy that I adopt is to have my data be separated by different item-delimiters that can be uniquely identified later to parse the line. For example, I store all of the data during the experiment in a javascript array. Each element in the array can be outputted as a comma separated list using the `.join` method (e.g., `yourarray.join()` returns a comma separated list in string format). Trial 1 might look like:

```
red,green,congruent,13241986,12341
```



The next step is to aggregate across lines of data into a single line. For this I choose another unique item-delimiter, like `:`

So, two lines of data can be concatenated onto one line like this:

red, green, congruent, 13241986, 12341



The `:` becomes an end of line character that is used in post-processing to individuate lines.

You will need to make your own decisions about how to format and process your data. This tutorial does not cover data-analysis or processing of data, the main purpose is to show how data can be passed and retrieved from amazon.

9.5.12 Loading the Stroop example experiment onto Amazon Turk

`StroopDemo8.html` is a more complete example providing code to run a Stroop experiment like the ones in the previous demos. This script extends the previous examples in several ways.

1. The demo can be loaded on Amazon Turk as HIT. It would be best to do this in sandbox mode.
2. The demo allows workers to preview the hit before accepting. In preview mode a message is displayed saying that in order to complete the hit, the worker must first press accept hit before continuing. Having a preview mode is helpful for workers to decide if they want to participate in your experiment.
3. The demo includes a browser detection script. If the browser is internet explorer, then a message is

returned saying that the HIT cannot be completed.

Links are provided to download safari, google chrome, or firefox.

4. The demo has functionality that interacts with amazon in important ways. When the worker accepts the hit, and completes the task, the pop-window automatically closes, and the information in the main window changes. After completion the main window shows a submit button, and a web form for workers to leave comments if they choose to. When the submit button is pressed the data from the experiment and any text in the form is passed back to amazon.

9.5.13 Steps to run the demo on Amazon Turk

1. Create a free dropbox account so that you can serve the experiment to the web. With the dropbox folder, anything placed inside your dropbox public folder can generate a link. Create a new folder inside your dropbox public folder. This folder will house the files needed to run the experiment. Copy the following files from the tutorial folder into the folder that you create:
 - StroopDemo8.html This is the HTML for the opening page
 - StroopDemo8b.html This is the main experiment, opens as a pop-up

- `StrooptrialGenerator.js` Javascript functions for the experiment
- `Jquery-1.7.1.js` JQUERY library
- `Stroop.css` Style sheet for experiment
- `TURKconsent.htm` Example consent form
- `StroopInstruct.htm` Stroop Instructions

You can test the experiment at this point. If you open up `StroopDemo8.html` in a web-browser everything should run smoothly. However, at the end of the experiment you will not be able to submit or retrieve the data, as this part only works when the experiment is loaded and run on Amazon Turk.

2. The tutorial folder contains a folder called `DemoStroop`. This is basically a copy of the `external_hit` folder provided by Amazon and reviewed in the previous section. The contents of the `DemoStroop` folder are:

- `approveAndDeleteResults.sh` Run after all hits complete, approves and deletes hits
- `external_hit.input` Carry-over from `external_hit`, not used but left in place
- `external_hit.properties` Edit to change properties of your experiment
- `external_hit.question` Edit to change web-address for experiment
- `external_hit.results` Will contain results after

they have been downloaded

- `external_hit.success` A log file
- `externalpage.htm` Carry-over from `external_hit`, not used
- `getResults.sh` Script to retrieve data from Amazon
- `reviewResults.sh` Allows you to review results
- `run.sh` Load a HIT onto Amazon

Make a copy of this folder and/or place it into your `aws-mturk-clt-`

`1.3.0/samples/` directory. I made this folder by simply copying the `external_hit` folder, giving the folder a new name, and then editing the contents of the text files that are contained within. In the `DemoStroop` folder most of the files have already been edited, but a few more edits will be required.

9.5.13.1 Edits to file path

If you were to make a copy of the `external_hit` folder, then the main edits that you will need to make are to the file paths in each of the scripts. In each of the scripts, whenever you see the filepath `/samples/external_hit`, you will need to change `external_hit` to the new filepath, in this case `DemoStroop`. All of the scripts in the `DemoStroop` folder have already included this change.

9.5.13.2 Edits to web-address

The file `external_hit.question` needs to be edited so that you can refer Amazon to the web address for the experiment. The web address that is currently there links to a file on my dropbox folder. You will want to change this to link to the `StroopDemo8.html` file in your dropbox folder (note: control-clicking or right-clicking on `StroopDemo8.html` in the finder will give a dropbox option to copy public link, this is the link to place in `external_hit.question`).

9.5.13.3 Edits to properties

The `external_hit.properties` file allows you to change the title, description, keywords, reward (payment), and other properties of your HIT.

9.5.13.4 NUMBER OF ASSIGNMENTS vs. NUMBER OF HITS (BIG DIFFERENCE)

The `external_hit.properties` file allows you to change the number of assignments per hit. Most likely this is option that you want. Using this option Amazon generates 1 HIT, and ensures that any individual worker completes the hit just once. If you want many unique workers to complete the HIT, then set the number of assignments to a larger value.

You can change the number of HITS in the `run.sh` script. You will see `-maxhits 1` at the end of the third last line. If you

increase this number then Amazon will load the corresponding number of HITS, however this will give the same worker the option to complete the hit as many times as it is listed.

If you want unique subjects, then change the number of assignments and keep maxhits to 1.

9.5.14 Ready to Load!

All the files should be in place. To load the experiment onto amazon use the same steps as in the external_hit example

1. open up a terminal
2. change the directory to DemoStroop
3. Make sure your `mturk.properties` file has the sandbox line uncommented, and the main website commented out.
4. Type `./run.sh` into the terminal and press enter
5. If the hit loads you will get a link to it, copy and past into a browser and accept and complete the hit
6. Note: the experiment is set to last 10 trials.
7. Submit the hit back to Amazon
8. To download the data type `./getResults.sh` into the terminal and press enter
9. View the data that is now in `external_hit.results` in your DemoStroop folder

10. To approve and delete the hit type:

`./approveAndDeleteResults.sh` and enter into the terminal