Core JavaScript

Lab Exercise Suggestions

January 2016, Simon Roberts

Zeller's Congruence 1

Find "Zeller's Congruence" on Wikipedia. This formula returns the day of the week based on three numbers (day of month, month of year, year) provided to it.

Write code that takes values of day, month, year, performs the calculation, and then displays the number representing the day of the week for that day, month, and year values in the code.

Test a few times with different dates to verify that the code works for different dates. Be sure to verify dates in January or February (as these involve the most computation) and at least one year divisible by 100 (e.g. 1900) and another divisible by 400 (e.g. 2000).

Notes:

- There are several variations of formula presented on the Wikipedia page. These are based on different simplifications for software, and the difference between the no-longer-used “Julian” calendar and the modern “Gregorian” calendar. Be sure to select the right one!
- In the calculation, Zeller's Congruence modifies the month/year such that the months January and February are treated as month numbers 13 and 14 of the preceeding year. Use the ternary operator: `<boolean> ? <if-true-value> : <if-false-value>` to perform this modification.
- It might simplify the exercise, and any necessary debugging, if you calculate sub-expressions in stages, rather than the whole thing at once.

Temperature Converter

Take a number in a string, convert this from text to a number, then assuming that the value represents a Fahrenheit temperature apply the formula: **c = 5 x (f – 32) / 9** to convert to a Celsius temperature.

Display the result.

String Chewing

Assign a variable to some literal text.

Generate a random number x representing the position of a character in the input text.

Display x and the character at that position.

Display the text modified by removal of the character at position x.

That's Mister To You!

Take a string containing a first and a last name, entirely in lower case, and separated by at least one space.

Prepare text that starts with “Mr.” or “Ms.” as you choose, followed by the first and last names modified to have upper case first letters.

Display the resulting text.

Guessing Game Setup

Consider a game in which the computer simulates picking four colored pegs, ordered from left to right. The human player would then make guesses about what colors are in what positions.

In this exercise, devise a means of representing the set of colors that are possible for the pegs, then create an array to represent four such colors.

Use a loop and a random number generator to initialize the array of colors, simulating the computer setting up the game board.

Display the game board something like this:

```
RED GREEN BLUE BLACK
```

Zeller's Congruence 2

Modify, or copy and modify, your solution to the Zeller's Congruence 1 exercise. Make these changes:

- Use `if` / `else` rather than the ternary operator to handle the adjustments to month/year values for months January and February (this is for the exercise, it's not an improvement!)
- Devise two distinct means of converting the numeric day-of-week encoding to a textual form. Code each individually, and display the result as the textual name of the weekday.

Las Vegas 1

In a loop that executes ten times, simulate throwing two dice. At each throw, display the values that show on the dice faces.

Display special messages for each of: double six, scores totaling 7, and double one.

Modify the program so that the loop exits after a double six is thrown.

Text Alignment 1

Define three lines of text as strings, then print them out with sufficient leading spaces to make them right justified on a 60 column page (be sure your output is represented using a fixed pitch font!)

Text Alignment 2

Define three short lines of text as strings, then print them out with sufficient extra spaces embedded between words to make the lines justified (that is, flushed left with both left and right margins) on a 60 column page (be sure your output is represented using a fixed pitch font!)

Las Vegas 2

Simulate 1000 throws of a pair of dice, count the number of times each "face value" (the sum of the two dice values) shows, and print the frequency of each face value in a table.

Las Vegas 3a

Create an array of (at least) six strings that represent the images (use text descriptions) on the wheel of a “one-arm bandit” machine. (Examples are “Triple Bar”, “Bar”, “Cherry”, “Orange”, “Seven”, “Coin”)

Generate three random numbers that are suitable for use as indexes into the wheel array and use these numbers to print out the result of “pulling the handle”.

Example output: `Bar : Triple Bar : Orange`

Las Vegas 3b

Take your solution to Las Vegas 3a and extend it as follows.

Create a two dimensional array of (at least) four winning wheel combinations. Each minor array should have the three wheel positions (e.g. “Bar”, “Bar”, “Bar” that are the particular winning combination) and the textual description of the prize, e.g. “Ten Dollars!”

After the result of pulling the handle has been determined, determine if the result is a winning position. If a win is found, print out the prize below the regular output of the three image names.

Example output:

```
Triple Bar : Triple Bar : Triple Bar
Sixty-four Thousand Dollar Jackpot!
```

Code Breaker

A “Caesar Cipher” is a simple cipher that works by “adding” an offset to the letters of the plain text. For example, given a key of 5, a letter “a” would be encoded as a letter “f”. The letter “z” would wrap round and become a letter “e”. Capitalization is abandoned in encoding, and spaces, numbers, and punctuation are left unchanged.

The goal of this lab is to create a program that uses a “brute force” approach to breaking messages that are in this code.

Decrypting a message may be handled by the same process as encyption, using a key that is 26 minus the original encryption key. This works since the processing of the message text is effectively “circular”; that is, clock arithmetic, or a modulo calculation. Since there are 25 possible keys (key 0 / 26 have no effect), your goal is to read a message from the console, and then output the effect of all 25 keys on the message. One of them, and usually only one, will be recognizably readable, the others will not.

Suggested approach:

Define the input text in a string, convert the text to lower case, then extract an array of all the chars in the message.

Duplicate the array 25 times, keeping track of the “number” of the copy.

Then for each copy work along each character of the array in turn.
If the character, let's call it c, is in the range 'a' <= c <= 'z' then add a key equal to the copy's number to each character in the array (so, for one array, add 1 to every character, for the next add 2, and so on). Be sure that if the resulting character value is greater than 'z' you adjust it so that it effectively wrapped round from 'z' to 'a'.

Do not modify characters that are outside the range of alphabetic characters.

After creating the 25 additional arrays, print out the key number and the converted text.

Test your program on the following text, determine the message, and the key used to decrypt it:

```
qcbufohizohwcbg, mci rwr wh!
```

Calendar

Code Zeller's Congruence as a method.

Write another method that takes a month and year and with the help of the Zeller's Congruence method, uses loops to print out a calendar for one month.

From the top level body of the script, define a month and a year and use them to call the calendar printing method.

Notes:

- Start the week on Saturday, print three letter weekday names as column headings.
- Leave empty days of the month as blank space, e.g.:

```
Sat Sun Mon Tue Wed Thu Fri
                          1
  2   3   4   5   6   7   8
  9  10  11  12  13  14  15
 16  17  18  19  20  21  22
 23  24  25  26  27  28  29
 30  31
```

Palindrome Checker

Write a program that checks a string to see if it is a palindrome. Palindromes are text that has the same sequence of letters backwards as forwards, such as “abcdefedcba”.

The program should define a text string and then determine if it is a palindrome or not, and print a message accordingly.

Some palindromes you can use for testing are:

- A Santa dog lived as a devil God at NASA
- Able was I ere I saw Elba
- Go deliver a dare, vile dog
- Racecar
- Some men interpret nine memos

The checker should ignore whitespace, punctuation, and capitalization. It's probably easiest to strip these out prior to calling the main, recursively defined, method.

Two versions:

Version 1: Implement the main behavior using a loop that counts from the start of the string to the middle, checking to see if the corresponding character at the other end of the string is the same.

Version 2: Implement the main behavior of this program using a recursive method call. As a hint, note that any string of length less than two characters is a palindrome. Further, if a string has the same first and last characters, and the rest of the String (other than the first and last) is also a palindrome, then that string is also a palindrome.

Birthdays 1—Simple Objects

The goal of this lab is take structured text strings representing birthdays, create an array of objects representing those same birthdays and then use methods in the objects to print the information out.

Create an array of strings called `birthdayLines`. This array should contain about a dozen birthday entries (you can either make these up, or find some real ones from `http://www.famousbirthdays.com/`). Use a comma separated values (csv) format for the strings, for example:

```
"Van Halen, Eddie, 1, 26, 1955"
```

Declare a variable called `birthdayProt`. Initialize this object with a literal containing fields and methods. There should be five fields called `firstName`, `lastName`, `day`, `month`, and `year`. There should be methods, with appropriate implementations, for `getName`, `getBirthdate`, and `toString`. These three methods should aggregate the first and last names, the three date fields, and all five fields, respectively.

Define a method `createBirthday` that takes a text argument in the format of the csv strings. The method should create a new object using the `birthdayProt` as the prototype. Then the argument string should be parsed into separate fields, which should be used to initialize the five fields of the newly created object. The new object should then be returned to the caller.

Create a variable that is initialized to an empty array called `birthdayArray`. In a loop, process each of the lines of text in `birthdayLines` to create an object representing the birthday. Once each object has been created add it to `birthdayArray`. When all the input lines have been processed, loop over the contents of `birthdayArray` and for each object that it contains, display the result of calling the `getName`, `getBirthday`, and `toString` methods.

Birthdays 2—Anonymous Functions

In this lab, you will sort an array of data based on differing criteria.

Copy the code you wrote for Birthdays 1, so that you do not damage the original.

Define a method `sortByName` that takes two arguments (assumed to be Birthday objects) and returns a negative number if the last name of the first is alphabetically "before" the last name of the second. It should return a positive number if the last name of the first comes after the last name of the second, and return zero if they're the same.

Sort the list using this function, then display it.

Next, write a single sort statement that embeds an anonymous function that sorts

by the year of birth, such that most recent years are at the start of the sorted result. Again, display the sorted list.

Birthdays 3—Closures

In this lab you will create a function that uses a closure to create a parameterized method.

Start by copying the code of the previous lab (again, so you do not risk damaging it).

Write a function that takes a single argument of Birthday type and returns true if the birthday takes place on a date with a year that is more recent than 1990. (Ensure your data set includes some dates that are either side of this point).

Use the `Array.filter` method and this function to create a new array that contains only dates that match the recency criterion. Display the resulting array.

Imagine that you want to filter the array based on another year threshold. Clearly, at present, you would have to duplicate the entire filter method, which would be wasteful.

Next, create a factory function that takes a single parameter (which will be the threshold year) and returns a ***function*** that compares a Birthday object's year with the year given.

Use the `Array.filter` method a few more times, passing a call to the factory function with a different threshold year each time. Display the resulting arrays.

Note, expect your call to the array filter method to look something like this:

```
var recentBirthdays =
  birthdayArray.filter(getBirthdayFilter(1980));
```

Birthdays 4—Error Handling

In this lab you will extend and improve the code you created in the previous lab. Copy your project first (so you do not damage the original). Then make the following enhancement:

Add a “damaged” record to the **`birthdayLines`** array by putting a negative day of month in one record.

Next, improve the `createBirthday` method so that it performs tests to ensure that the day, month, and year elements in the input string are greater than 1, and that the day and month fields are less than 31 and 12 respectively. If the record fails the validation throw an `Error` with an appropriate message to indicate this failure.

In the calling code, handle the exception so as to print a message reporting the problem, and then skip the processing of the offending data record. Verify the

behavior by temporarily making changes to the data records with no errors, and one or more errors in one or more records.

Birthdays 5—Function Constructors

In this lab you will further extend the Birthdays example. Copy your project again, and then make the following changes:

Convert/rename the `createBirthday` method to a constructor function. Modify the initialization operations to set fields in the object `this`.

Assign the prototype of the constructor function appropriately.

Verify that the code works as before both with good and bad date data in the file.

For a single record in the text data set, call the constructor as a simple function, that is, omitting the new keyword. What is returned? Look in the global object the variables that should have been defined in the object.

Add a check to the beginning of the constructor function to test if the current `this` object is an instance of `Birthday`. If it is not, throw an `Error` with a message explaining the misuse of the constructor function. Verify that your correct uses of the constructor function are accepted, but that the incorrect one causes a runtime error.

Correct the code so the runtime error does not break the program, then add code to set the day field of one `Birthday` to -1. Display the object via its `toString` method. Delete the year field using the `delete` operator and display the object again. What just happened?

Test one of the `Birthday` objects using the `instanceof` operator. Are these objects instances of `Birthday`? Of `Object`? Of `String`?

Number Guessing Game

Create a web page that has three text fields. Two are used to specify a range for a random number. The third is used to input a guess. The page should provide a button labeled “submit guess”. An empty unnumbered list should be placed after all these components.

When the button is clicked the first time, or the first time after a successful guess, the page reads the minimum and maximum numbers and generates a random number in that range.

If the button has been clicked for the first time after a successful guess, the list should be emptied.

On all clicks (including the first) the page reads the number in the guess. The page then appends a list item to the list that has text in the general form: “Guess 23 is too low”. If the guess is correct, then the message should say so and note the number of guesses taken to get the correct answer.

Concordance

The goal of this exercise is to create a table of frequency of occurrence of words in a text document, so that the table has the word, followed by the number of times that the word occurs in the document.

The web page should present a text field. When enter is pressed on the text field, the page should use the text as a URL and load the document at that address.

Assuming the page loads successfully, split the input text into an separate words (don't try too hard with this, English word structure is tricky; simply split the text on any non-word characters, including spaces and any punctuation).

For each word, convert the word to lower case.

Build a map (remember that JavaScript objects are actually map structures). Determine if the word is already in the map. If it is there then increment the number stored against it. If the word is not in the map, add it as a key with the value 1.

Finally, for every word in the resulting map, create a table row to reflect the word and the count of that word. Use these to populate a table structure, and add that to the end of the page.

Note: Due to the XSS protections, you will probably be unable to read your text document from any server except your own. For testing, create a simple text document on your own server, and provide a relative URL to that. For more demanding (and more interesting) data, you can download plain text versions of out-of-copyright books from http://www.gutenberg.org/, for example, try: http://www.gutenberg.org/cache/epub/1342/pg1342.txt

Extra credit:

- Arrange that the table is sorted in descending order of word frequency.
- Arrange to show only the 100 most frequent words in the table.