CSI3140 (Spring-Summer 2017)
ASSIGNMENT 3

Due Date: Email your completed assignment directly to TA2 by 14:00 on Tuesday, June 27, 2017.

Instructions:

- This assignment has 4 pages and **must be done individually**. It consists of 6 questions with the mark for each question indicated below, resulting in 75 marks in total.
- **Late assignments will NOT be accepted: They will receive a grade of zero.**
- Save your answer to each question in a separate file (or a set of files) and name it (or them) using the following format: **AssignmentNo_QuestionNo_YourLastName_YourStudentID**
  For example, John Smith, whose student ID is 1234567, should save his answer to Question 1 under the file name: **A3_Q1_Smith_1234567.html**, and save his solution to Question 4 (consisting of 3 files) under the file names: **A3_Q4_Smith_1234567.html**, **A3_Q4_Smith_1234567.js**, and **A3_Q4_Smith_1234567.css**
- Create a cover page in Word format containing your full name, student ID, course number, and assignment number. Name your cover page as **A3_CoverPage_YourLastName_YourStudentID.docx**. The TA will provide your assignment mark on this cover page and send it back to you (with your answer files if he/she has comments/feedback on them).
- Zip all your answer files together with your cover page into a single .zip file and name it **A3_YourLastName_YourStudentID.zip**, and email that .zip file to the TA by the due time and date above.
- The subject line of your email should follow the format: **CSI3140_A3_YourLastName_YourStudentID**
- Note: Google has recently blocked JavaScript files (files with **.js** extension) from being attached to an email. They also prevent users from uploading a zip file containing **.js** files. Thus, you will have to change the **.js** extension into **.txt**, in order to email your files to the TA, who then will change the **.txt** extension back to **.js** before testing them.

The following questions require JavaScript programming. Your code should be well written (e.g., variables declared, meaningful variable names, easy to understand and no confusing shortcuts, meaningful functions, and useful comments, etc.).

1. Implement the following functions:
   a) Function *celsius* returns the Celsius equivalent of a Fahrenheit temperature, using the calculation: C = 5.0 / 9.0 * (F – 32)
   b) Function *fahrenheit* returns the Fahrenheit equivalent of a Celsius temperature, using the calculation: F = 9.0 / 5.0 * C + 32
   c) Use these functions to write a script (embedded in an HTML5 document) that enables the user to enter either a Fahrenheit or a Celsius temperature and displays the Celsius or Fahrenheit equivalent.

Your HTML5 document should contain two buttons – one to initiate the conversion from Fahrenheit to Celsius and one to initiate the conversion from Celsius to Fahrenheit. [10 marks]

2. Write a script (embedded in an HTML5 document) that plays a "guess the number" game as follows: Your program chooses the number to be guessed by selecting a random integer in the range from 1 to 1000. The script displays the prompt "Guess a number between 1 and 1000" next to a text field. The player types a first guess into the text field and clicks a button to submit the guess to the script. If the player's guess is incorrect, your program should display "Too high. Try again." or "Too low. Try again." to help the player zero in on the correct answer. When the user enters the correct answer, display "Congratulations. You guessed the number!". [10 marks]

3. Modify your solution to Question 2 above to count the number of guesses the player makes. If the number is fewer than 10, display "Either you know the secret or you got lucky!". If the player guesses the number in 10 tries, display "Ahah! You know the secret!". If the player makes more than 10 guesses, display "You should be able to do better!" In addition, handle incorrect input: If the player's input is not a number, display "Wrong input! You should enter a number between 1 and 1000." and do not count that guess. [10 marks]

4. A *prime* integer is an integer greater than 1 that is evenly divisible only by itself and 1. The Sieve of Eratosthenes is an algorithm for finding prime numbers. It operates as follows:

a) Create an array with all elements initialized to 1 (true). Array elements with prime indices will remain as 1. All other array elements will eventually be set to zero.

b) Set the first two elements to zero because 0 and 1 are not primes. Starting with array index 2, every time an array element is found whose value is 1, loop through the remainder of the array and set to zero every element whose index is a multiple of the index for the element with value 1. For example, for array index 2, all elements beyond 2 in the array that are multiples of 2 will be set to zero (i.e., indices 4, 6, 8, 10, etc.); for array index 3, all elements beyond 3 in the array that are multiples of 3 will be set to zero (i.e., indices 6, 9, 12, 15, etc.); and so on.

When this process is complete, the array elements that are still set to 1 indicate that their indices are prime numbers. These indices can be printed. Write a script (in a .js file separate from the HTML5 document) that uses an array of 1000 elements to determine the number of the prime numbers between 1 and 999 and output these prime numbers. Ignore element 0 of the array. Your CSS rules (if any) should also be placed in a separate .css file from the HTML5 document. [10 marks]

5. You will use random number generation to develop a simulation for the classic race of the tortoise and the hare. The contenders begin the race at square 1 of 70 squares. Each square represents a possible position along the race course. The finish line is at square 70. The first contender to reach or pass square 70 is rewarded with a pail of

fresh carrots and lettuce. The course weaves its way up the side of a slippery mountain, so occasionally the contenders lose ground. Assume that there is a clock that ticks once per second. With each tick of the clock, your script should adjust the position of the animals according to the rules in Table 1 below:

Animal	Move type	Percentage of time	Actual move
Tortoise	Fast plod	50%	3 squares to the right
	Slip	20%	6 squares to the left
	Slow plod	30%	1 square to the right
Hare	Sleep	20%	No move at all
	Big hop	20%	9 squares to the right
	Big slip	10%	12 squares to the left
	Small hop	30%	1 square to the right
	Small slip	20%	2 squares to the left

Table 1: Rules for adjusting the position of the tortoise and the hare

Use variables to keep track of the positions of the animals (i.e., position numbers are 1 – 70). Start each animal at position 1. If an animal slips left before square 1, move the animal back to square 1. Generate the percentages in Table 1 by producing a random integer i in the range $1 \leq i \leq 10$. For the tortoise, perform a "fast plod" when $1 \leq i \leq 5$, a "slip" when $6 \leq i \leq 7$, and a "slow plod" when $8 \leq i \leq 10$. Use a similar technique to move the hare.

Provide a button labeled "Start Race", on which the user clicks to start the race. Begin the race by printing:

ON YOUR MARK, GET SET

BANG!!!

AND THEY'RE OFF!!!

Then for each tick of the clock (i.e., each repetition of a loop), print a 70-position line showing the letter T in the position of the tortoise and the letter H in the position of the hare. Occasionally, the contenders will land on the same square. In this case, the tortoise bites the hare, and your script should print OUCH!!! at that position. All print positions other than the T, the H, or the OUCH!!! should be blank.

After each line is printed, test whether either animal has reached or passed square 70. If so, print the winner and terminate the simulation. If the tortoise wins, print TORTOISE WINS!!! YAY!!! If the hare wins, print HARE WINS. YUCK! If both animals win on the same tick, print IT'S A TIE. Also, print the time elapsed (the number of ticks) of the race. If neither animal wins, perform the loop again to simulate the next tick of the clock. Separate your script (.js file) and your CSS rules (.css file – if any) from the HTML5 file.

Note: Later in the book, we introduce a number of Dynamic HTML capabilities, such as graphics, images, animation and sound. As you study those features, you may enjoy enhancing your tortoise-and-hare contest simulation. [20 marks]

6. Write a script that encodes English-language phrases in pig Latin. Pig Latin is a form of coded language often used for amusement. Many variations exist in the methods used to form pig Latin phrases. For simplicity, use the following algorithm:
To form a pig Latin phrase from an English-language phrase, tokenize the phrase into an array of words using *String* method *split*. To translate each English word into a pig Latin word, place the first letter of the English word at the end of the word and add "ay". Thus, the word "jump" becomes "umpjay", the word "the" becomes "hetay", and the word "computer" becomes "omputercay". Blanks between words remain as blanks. Assume the following: The English phrase consists of words separated by blanks, there are no punctuation marks, and all words have two or more letters. Your script should include function *printLatinWord*, which displays each pig Latin word. Each token (i.e., word in the sentence) is passed to function *printLatinWord* to print the corresponding pig Latin word. Enable the user to input the sentence through an HTML5 form. Keep a running display of all the converted sentences in an HTML5 *textarea*. Separate your script from the HTML5 document. [15 marks]