ITWS Assignment - 2

Instructions

Install Node (version 8.x.x || 9.x.x)

1. Download node release from nodejs.com;
2. Extract node-*
3. cd node-*/bin
4. export PATH=$PATH:$(pwd)

Assignment setup

1. Extract assignment-2.zip
2. cd assignment-2
3. npm install

Running test cases

1. Change ‘xit()’ to ‘it()’
2. npm run test

Additional details

1. Strictly follow the naming convention. Boilerplate code in provided in ‘index.js’.
2. Refer to the documentation on MDN. No clarification will be provided for syntax.
3. Refer to the test cases for usage and other finer details with regards to the questions.
4. The test cases provided are very minimal, additional tests will be run during evaluation.
5. You can modify the test cases or add new tests.

Submission format

1. Submit a zipped folder - ‘2017XXXX.zip’ with ‘index.js’ file.

Grading

1. The grading for the assignment will be automatic.
2. Strict action will be taken against any copy cases found.

Questions

Q1) Write a function named ‘myArrayFilter’ which accepts 2 arguments - Array, callback. The function ‘myArrayFilter’ returns a new Array. The callback function accepts 3 arguments - value, index (optional), Array (optional). The callback function is called on every element of the Array. If the callback function returns true on any element, then that element is added to the new Array. Do not use the any default methods in Array.prototype.

Q2) Write a function named ‘myArrayReduce’ which accepts 3 arguments - Array, callback, accumulator (optional). The function ‘myArrayReduce’ returns the value of the accumulator. The callback function accepts 4 arguments - accumulator, value, index (optional), Array (optional). The callback function is called on every element of the Array and the return value is assigned to the accumulator. Do not use the any default methods in Array.prototype.

Q3) Write a function named ‘myTreeReduce’ which accepts 2 arguments - in-function, end-function. The return value is a new function which accepts a Tree as an argument. The in-function is called on the internal nodes of the Tree and the end-function is called on the leaf nodes of the Tree.

```
Internal node: {
        type: ‘in’,
        value: 1,
        left: {},
        right: {}
}

Leaf node: {
        type: ‘end’,
        value: 1
}
```

Q4) Write a function named ‘myTreeSize’ which accepts a Tree as an argument and returns the number of nodes in the Tree. Use the function ‘myTreeReduce’. Hint: Think about what the in-function and end-function will be for this case.

Q5) Write a function named ‘myTreeTraversal’ which accepts a string whose values are - ‘pre’, ‘in’, ‘post’. The return value is a function which accepts a Tree as an argument and returns the corresponding pre-ordering, in-ordering and post-ordering of the Tree. Lookup pre-order, in-order and post-order traversal on Trees.

Q6) Write a function named ‘hangman’ that simulates the Hangman game. The maximum mistakes allowed is 3. Assuming that passing the same letter more than once is not a mistake. Refer to the test cases for more details.

Q7) Write a constructor function named ‘Person’ that adds the following properties - name, age. Add a method named ‘about’ to ‘Person.prototype’ which returns a string with the name and age. Write another constructor function named ‘Student’ that uses the ‘Person’ constructor and adds an additional property - roll. Add a method named ‘id’ to ‘Student.prototype’ which returns a string with the roll property. Add the ‘about’ method of ‘Person.prototype’ to ‘Student.prototype’. Do not change the constructor property for ‘Student.prototype’.

Q8) Create an Object named ‘numberList’ which has the following properties:

numbers:	Array of Numbers
add:	Adds a new element to ‘numbers’
sum:	Sum of elements in ‘numbers’
average:	Average of elements in ‘numbers’

Hint: Refer to the documentation of getters & setters on MDN.

Q9) Write a function named ‘carRace’ that accepts 2 arguments - Array of Promises, callback. Each Promise object resolves with a car object. Find the car that has the fastest time and pass a string with its name and time to the callback function. Refer to the test case for output string format.

```
Car: {
        name: ‘Ferrari’,
        time: 10
}
```