Project Goals

Context: You're part of a research team that has found a new mysterious organism at the bottom of the ocean near hydrothermal vents. Your team names the organism, Pila aequor (P. aequor), and finds that it is only comprised of 15 DNA bases. The small DNA samples and frequency at which it mutates due to the hydrothermal vents make P. aequor an interesting specimen to study. However, P. aequor cannot survive above sea level and locating P. aequor in the deep sea is difficult and expensive. Your job is to create objects that simulate the DNA of P. aequor for your research team to study.

As you progress through the steps, use the terminal and console.log() statements to check the output of your loops and functions.

Prerequisites

1.

In order to complete this project, you should have completed the first few sections of Introduction to JavaScript (through Learn JavaScript: Objects).

Project Requirements

2.

Look over the starter code. There are two helper functions: returnRandBase() and mockUpStrand().

DNA is comprised of four bases (**A**denine, **T**hymine, **C**ytosine, and **G**uanine).
When returnRandBase() is called, it will randomly select a base and return the base ('A','T','C', or 'G').

mockUpStrand() is used to generate an array containing 15 bases to represent a single DNA strand with 15 bases.

You'll use these helper functions later to create your objects that represent *P. aequor*.

3.

Since you need to create multiple objects, create a factory function pAequorFactory() that has two parameters:

- The first parameter is number (no two organisms should have the same number).
- The second parameter is an array of 15 DNA bases.

pAequorFactor() should return an object that contains the properties specimenNum and dna that correspond to the parameters provided.

You'll also add more methods to this returned object in the later steps.

4.

Your team wants you to simulate *P. aequor*'s high rate of mutation (change in its DNA).

To simulate a mutation, in pAequorFactory()'s returned object, add the method .mutate().

.mutate() is responsible for randomly selecting a base in the object's dna property and changing the current base to a different base. Then .mutate() will return the object's dna.

For example, if the randomly selected base is the 1st base and it is 'A', the base must be changed to 'T', 'C', or 'G'. But it cannot be 'A' again.

5.

Your research team wants to be able to compare the DNA sequences of different *P. aequor*. You'll have to add a new method (.compareDNA()) to the returned object of the factory function.

.compareDNA() has one parameter, another pAequor object.

The behavior of .compareDNA() is to compare the current pAequor's .dna with the passed in pAequor's .dna and compute how many bases are identical and in the same locations. .compareDNA() does not return anything, but prints a message that states the percentage of DNA the two objects have in common — use the .specimenNum to identify which pAequor objects are being compared.

For example:

ex1 = ['A', 'C', 'T', 'G']
ex2 = ['C', 'A', 'T', 'T']

ex1 and ex2 only have the 3rd element in common ('T') and therefore, have 25% (1/4) of their DNA in common. The resulting message would read something along the lines of: specimen #1 and specimen #2 have 25% DNA in common.

6.

P. aequor have a likelier chance of survival if their DNA is made up of at least 60% 'C' or 'G' bases.

In the returned object of pAequorFactory(), add another method .willLikelySurvive().

.willLikelySurvive() returns true if the object's .dna array contains at least 60% 'C' or 'G' bases. Otherwise, .willLikelySurvive() returns false.

7.

With the factory function set up, your team requests that you create 30 instances of pAequor that can survive in their natural environment. Store these instances in an array for your team to study later.

Project Extensions & Solution

8.

Great work! Visit our forums to compare your project to our sample solution code. You can also learn how to host your own solution on GitHub so you can share it with other learners! Your solution might look different from ours, and that's okay! There are multiple ways to solve these projects, and you'll learn more by seeing others' code.

9.

If you'd like to challenge yourself further, you could consider the following:

- Create a .complementStrand() method to the factory function's object that returns the complementary DNA strand. The rules are that 'A's match with 'T's and vice versa. Also, 'C's match with 'G's and vice versa. (Check the hint for more details)
- Use the .compareDNA() to find the two most related instances of pAequor.