...GTCAGATCGTTCGAACGGGGCTCCAAACA...

Javascript: DNA Sequences

NOTES:

You should attempt all three tasks here in a single program.

The testing framework will be looking for all the features in all the tests.

REMEMBER - LOOP STRUCTURES ARE BANNED!

To stand ensure you can handle all tests, think thoroughly of various test cases.

Background:

A common task for DNA sequencing is the identification of patterns of nucleotides - the individual building blocks of both RNA and DNA. Unfortunately, it can be quite difficult to identify the specific nucleotide when doing DNA sequencing, so often instead of a specific organic molecule, the sequencing machine is only able to guess at a number of options at the same position.

The machine indicates this by not using one of the definite chemical characters (A, C, G and T), but with one that indicates some combination of these four, according to the table below:

Recorded Character	Possible Actual Nucleotides			
R	G	A		
Y	T	C		
K	G	T		
M	A	C		
S	G	C		
W	A	T		
B	G	T	C	
D	G	A	T	
H	A	C	T	
V	G	C	A	
N	A	G	C	T

(See also https://en.wikipedia.org/wiki/Nucleic_acid_sequence for the intricate details)

You have been given the recorded DNA sequences from a sequencing machine and a number of sequences to search for within this body of data.

BUT

There is a special rule for these tasks - **Loop structures are not allowed** - this means you cannot use any `for`, `while`, or `do/while`, or any other form of these structures. Instead, you should make extensive use of the Array functions (https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array) and Object functions (https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Object) to search, match and refine your data.

The Mozilla JavaScript documentation can help you with this, take a look at the links above, along with the documentation link here: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference . You will have to do some reading around the documentation to be able to complete these tasks!

You will begin this coursework by familiarising yourself with the description and examining the provided files to grasp the tasks. Research will be essential to understand some of the APIs that you'll need to utilise.

Task 1: Frequency Counts

You've been asked to produce the counts of particular sequences handed to you from a test framework. The framework itself has a function for you to call with the results.

In essence, if you have been given the following sequence of nucleotide letters:

CCAAATT

We should be able to generate the following counts of each pattern in the following structure:

```
{ AA: 2, AT: 1, CC: 1, CA: 1, TT: 1 }
```

Note that 'AA' is detected as two patterns, as the first pair of 'A's then a second pair of *overlapping* 'A's:

CC **AA** ATT *and* CCA **AA** TT

Your code will be handed the input character-by-character, so it will be up to you how you decide to buffer and detect these sequences.

In the supplied .zip file, you should have 6 data files (in pairs, for each task), and two JavaScript files. One is the library you are to use (testlib.js) and the other is a template to base your implementation on (minimal_demo.js). **You should not modify testlib.js**. All need to be in the same folder to operate correctly.

To run the demo file, execute 'node minimal_demo.js' from the command line or terminal in the program's folder.

minimal_demo.js

```
"use strict";
const testlib = require( './testlib.js' );
testlib.on( 'ready', function( patterns ) {
    console.log( "Patterns:", patterns );
    testlib.runTests();
} );
testlib.on( 'data', function( data ) {
    console.log( "<<<", data );
} );
testlib.setup( 1 ); // Runs test 1 (task1.data and task1.seq)
```

The minimal_demo includes how to connect to the testlib library, along with how to start the test. The source code of the library is documented on the functions available for your use.

For this particular task, you are required to use the `testlib.frequencyTable()` function to report your frequency counts. This function expects that you supply it with an object containing keys as specified by the `patterns` array in the `'ready'` handler, each with a count of the number of times the program has seen that pattern.

Remember that you can access JavaScript object and array keys by square brackets.

An (extremely simple) example of how to do this might be:

```
let data = { "AA": 1, "TT": 12 };

testlib.frequencyTable( data );
```

Note: To know when one sequence stops and another begins, use the `'reset'` hook via the `testlib.on()` function!

To test your code for this task, remember to set `testlib.setup( 1 )`

Task 2: Short Sequence Matching

Now that you have counting and matching working, you're now tasked with reporting the precise location of each match for each pattern.

Testlib provides a `testlib.foundMatch( pattern, offset )` function which should be called when a match is found, passing it the pattern found along with the offset in characters from the start of the current sequence.

Again, in the library you have been given, this function will simply print the details it has been provided with.

To test your code for this task, remember to set `testlib.setup( 2 )`

Task 3: Complex Sequence Matching

Remember the uncertainty inherent in sequencing DNA? Where sometimes the sequencing machine can only guess? This task deals with that situation.

So far you have only been dealing with A, C, G and T symbols, now you have been given a much longer set of sequences containing many partially matched nucleotides - this data now can contain A, C, G, T, R, Y, K, M, S, W, B, D, H, V, and N! (See the table in the Background section for the details).

Your code should be able to detect the sequences handed to it for this task (a reminder that these are provided to the `'ready'` handler). These sequences *and* the data you will get have the new set of symbols.

A match for a sequence is now *any combination of possible symbols,* so a pattern MTAC should match with ATAC *or* CTAC, as M can resolve to A *or* C.

A successful solution to this should be able to report frequency tables *and* identify matches as per tasks 1 and 2.

To test your code for this task, remember to set `testlib.setup( 3 )`

Sample Test cases

It is essential to conduct comprehensive testing on your code, encompassing a variety of test cases. Here are a few examples:

Patterns	**Input (data)**	**Frequency**	**Offset**
CC AC	ACCA	CC: 1 AC: 1	CC: 1 AC: 0
NN	ATCGATCG	7	0,1,2,3,4,5,6
TCG	ATCGATCG	2	1,5
WNGRC	RTMYYCDYWNGRCYCANTKMH	1	8
RTMY	RTMYGTCY	2	0,4

Marking

Marks will be awarded according to the following table:

Task	Mark/60
Task 1: Frequency Counts	15/60
Task 2: Short Sequence Matching	10/60
Task 3: Complex Sequence Matching	35/60

You should have Tasks 1 and 2 working fully before attempting Task 3.

REMEMBER - LOOP STRUCTURES ARE BANNED!