Worksheet 1

Problem Solving for Computer Science

In this worksheet, the basic objectives are:

1. To get familiarity with the programming environment and get comfortable with different ways of testing your code
2. To complete a function that flips arrays
3. To get experience with `while` loops
4. To write a function that converts a decimal number into binary form stored in an array

Before you start the tasks in this worksheet you will need to have Node.js installed. If you haven't done so already, watch the relevant video on installing Node.js in Week 1. It is a good idea for you to go through the reading material "Introduction to Node.js".

Setting up

Now go to Week 2 in learn.gold and watch the short video called "Introduction to Worksheet 1".

Let's review the video. Go to Week 2 in learn.gold and download `lab1.zip`. From the zip file you will get a folder called `lab1` if you are a MacOS user. You can move this folder to somewhere else in your machine such as the `Documents` folder to make things easier. If you are a Windows user, when you unzip the file, you will see two folders, one of which is called `lab1`, which you can move to another folder as you wish. Inside the folder `lab1` you will see the files `lab1.js`, `package.json`, `reporter.js` and a folder called `node_modules`. We will only need to work with `lab1.js`.

1. Open `lab1.js` in your chosen IDE, e.g. Brackets
2. Open your command line interface (CLI) and change your directory to the `lab1` folder. You can do this by typing `cd` then a space and then drag and drop the folder `lab1` into the command line interface (this is also covered in the videos and reading material for installing Node)

From the video two important points are:

- Firstly, develop and test your code using `console.log()` and print the output of functions and variables to the console when you write `node lab1` in the command line
- Secondly, test the correctness of your code with `npm test` - this will tell you if your code passes or fails a series of tests

You can think of the first tests as you checking your own working out, but the second tests as having your work checked by something else if it is correct.

IMPORTANT: DO NOT CHANGE THE NAMES OF FUNCTIONS OR CREATE NEW FUNCTIONS

Task One: Flipping arrays

In this worksheet, we will be working with arrays and converting flowcharts into code. If you have forgotten the syntax for JavaScript arrays or what they are, go to Resources in learn.gold and read "JavaScript Arrays".

In lab1.js you will see a function called swap(array, i, j). This will swap the values at positions i and j in an array called array. Do not alter this function. Below this function write the following:

```
var arr = [1,2,3,4];
swap(arr, 1, 3);
console.log(arr);
```

Now when you run this code using node lab1, you should get the following printed:

```
[1, 4, 3, 2]
```

In the first task you will repeatedly apply the swap(array, i, j) function to flip the order of values in an array.

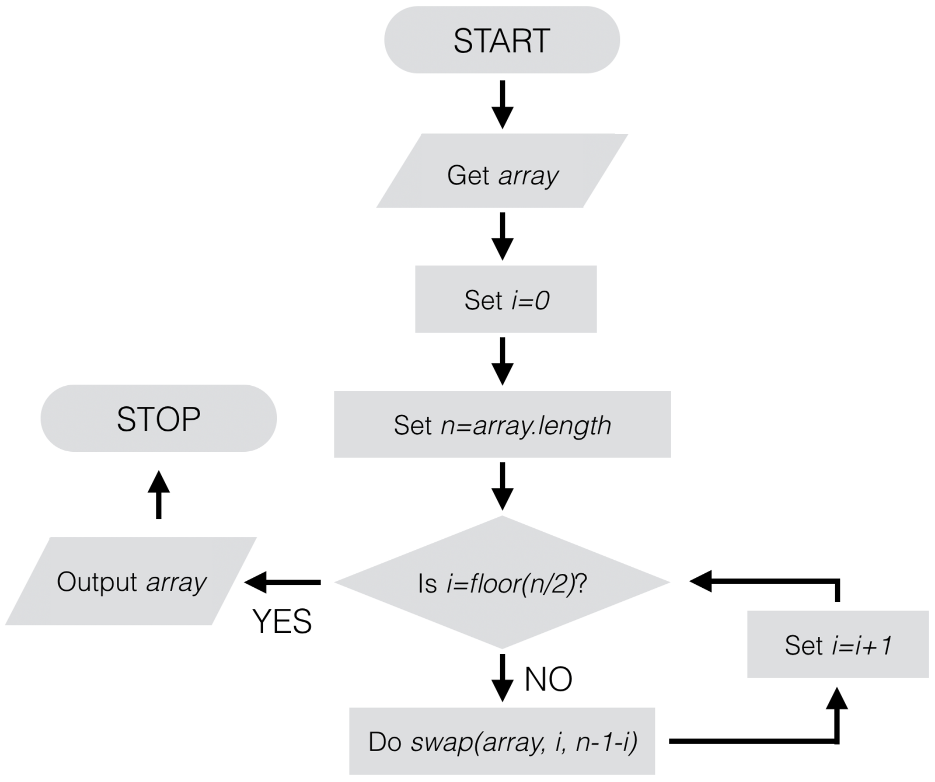
Go to the function swapArray(array) in the JavaScript file. Your first task is to complete this function so that it will take the argument array and swap the elements to flip it. For example, if we write this code into our file:

```
var arr = [1, 2, 3, 4];
swapArray(arr);
console.log(arr);
```

When swapArray(arr) is completed, we should get in the CLI:

```
[4, 3, 2, 1]
```

The flowchart to complete this function is the following:



Follow this flowchart to complete the function - some of the code is already there so work out what is missing. First test your function using console.log() as above, and then see if it passes the tests for npm test.

while loops

In the previous task, we had a `for` loop that iterated over values of `i` as in the flowchart. In addition to `for` loops, in JavaScript (and in other programming languages), there is another kind of loop: the `while` loop.

The `while` loop works by repeating some code for as long as the "loop condition" is satisfied, i.e. for as long as a condition is true. For example, consider the following piece of code:

```
var x = 0;
while (x < 5) {
  console.log(x);
  x++;
}
```

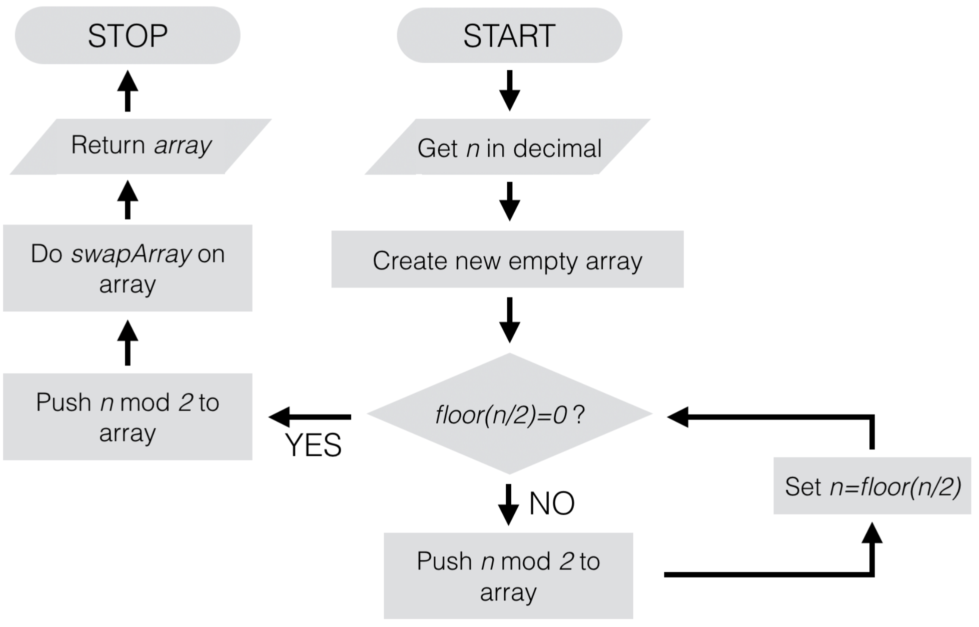
The loop condition here is `x < 5`: the code inside the curly brackets will be executed as long as `x < 5`. In this case, it will print the value of `x` to the console, and then increase it by one until `x` has the value `5`. At this point the condition is no longer satisfied and we break out of the loop. You can check this for yourself.

`while` loops are very useful when we have to repeat something an unknown number of times. In a `for` loop, we might need to work out the number of times to iterate something. For example, In the next task you should use a `while` loop.

Task Two: Converting from decimal to binary

In JavaScript, numbers are in decimal form, but a classic task in computing is representing a number in binary, i.e. in terms of zeroes and ones. For example, the number `20` in decimal form is `10100`. Your goal in this task is to complete the function called `changeToBinary(n)` that takes the number `n` as an argument and outputs an array which is `n` in binary form. For example, `changeToBinary(21)` should return `[1,0,1,0,0]`.

The following flowchart describes how this should be done:



In this flowchart "Push *x* to *array*" means using `array.push(x)`. In the case of this function the array is called `out`.

We can see a loop in the flowchart here starting and ending at the decision *floor(n/2)=0?*. For this loop you should use a `while` loop, where two actions should be repeated while *floor(n/2)* is *not* equal to 0. The loop will be exited then when *floor(n/2)=0*.

To develop your function try the following: `console.log(changeToBinary(20))`. This should print `[1,0,1,0,0]` to the console. Then when you are confident you have something that works, run `npm test`.

Why does this algorithm work? We will return to this later in the module, but on your own see if you can work out why it works.

Advanced Task: array of binary numbers of a particular length

Strings and arrays of binary digits (bits), or bit strings, can appear in all sorts of situations. An obvious example of a string of bits is a byte, which is a string of eight bits that is a fundamental unit of information.

The function `changeToBinary` will never generate an array such as `[0, 0, 0, 0]` for any argument `n`. In this harder task, the goal is to complete the function `binLength(n, m)` so that it returns an array of length `m` such that

1. The rightmost elements are the binary representation of the number `n` as returned by `changeToBinary(n)`
2. If `m` is larger than `changeToBinary(n).length`, then there will be `(m - changeToBinary(n).length)` elements with each storing the number `0`
3. If `m` is smaller than `changeToBinary(n).length`, return `"Error"`

For example, `binLength(5, 6)` will return `[0, 0, 0, 1, 0, 1]` since `changeToBinary(5)` returns `[1, 0, 1]`. Importantly, the function should call `changeToBinary`.

To test your function you can use `console.log(binLength(5, 6))` to see if the correct thing is being returned. Then, try other examples where `m` is less than, greater than, or equal to the `changeToBinary(n).length`. Then use `npm test` to see if passes the tests.