Resit Coursework Assignment

The 6 tasks in this work sheet are part of your coursework resit – your percentage mark from this will make up 60% of your total resit mark. Attempt as many of these tasks as you can. You will not need to get full marks in this part of the coursework resit to get a passing grade overall, but to increase the chances of getting a passing grade you should attempt as much as possible. Remember that **you get marks for attempting any solution to the tasks**.

1 Your lab folder

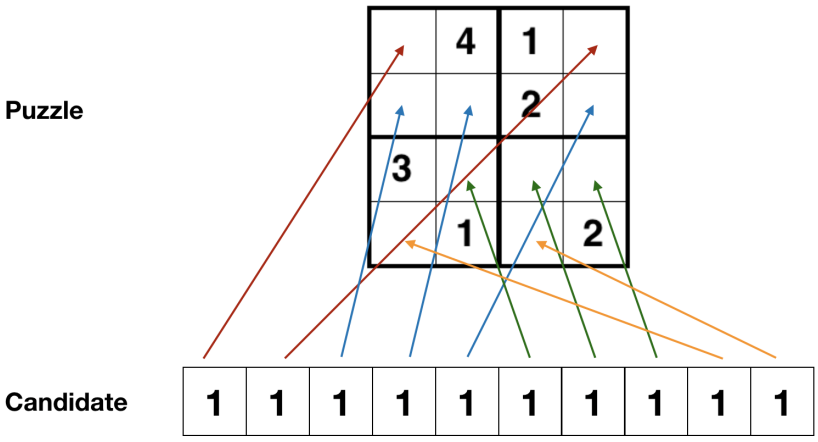
Created a folder called *resit* – you will be using this folder for this work sheet. Go to the learn.gold section from where you downloaded this work sheet, and download the file called *resit.js*. Make sure this file is in your newly created folder called *resit*.

Open the file called *resit.js* and have a look at its contents. You will see a variable called `puzzle`, which is a Pseudoku puzzle that you will use to test your (completed) functions. After this variable you will find six function templates, which you will complete in the six tasks.

2 Reminder: solving Pseudoku puzzles

In Lab sheets 3 and 4, the tasks involved generating and solving Pseudoku puzzles. In Lab sheet 4 we had a function called `solvePseudoku(array)`, which takes a Pseudoku puzzle in array form as an input parameter, called `array`. The input parameter `array` will contain elements that have values from 1 to 4, as well as a number of elements with the space string `" "`. The function `solvePseudoku` will return an array which will be the solved version of the input `array`.

In the solution method of `solvePseudoku` we generated candidates for the solution one by one and tested them. Let's look at an example of a Pseudoku puzzle and a candidate to remind ourselves of what is going on.



This was not a good candidate since it does not satisfy the Pseudoku conditions.

In this work sheet we are going to go through a few activities related to solving Pseudoku puzzles. Some of the activities are related, but the final task is to use the results of these activities to **randomly generate a candidate**.

3 Tasks

3.1 Counting the number of blank entries

In this first task, you will write a function that counts the number of times that the value `" "` appears in the input parameter, which will be a two-dimensional array such as `puzzle`.

Task 1: Complete function `numBlanks`, which takes as input an array called `array`, and returns a number.

Goal: To write a function that will return the number of all the entries of `array` that have the value `" "`

Method: Create a variable called `num` and set it to zero. Go through every element in the two-dimensional array called `array`, and increase `num` by 1 every time the element is equal to `" "`.

Test: Call the function on the variable `puzzle` using `numBlanks(puzzle)`. You should return the value `10`.

[4 marks]

3.2 Looking for possible missing values of the puzzle

In the next three tasks, we are going to create an array that stores the values of numbers that do not appear in each row of the puzzle. So if we look at the puzzle in the previous section we see that the numbers 4 and 1 appear in the top row, which means the other elements in the top row will take the values 2 and 3. Eventually after completing these tasks we will have an array of four elements, and each element will store the possible values that can be put into the elements in each row. We will use the Linear Search algorithm to do this.

Task 2: Complete function `linearSearch`, which takes as input an array called `array` and a number called `item`, and returns a Boolean.

Goal: To write a function that implements the Linear Search algorithm on the array called `array`.

Method: The algorithm proceeds by asking if every element of the array called `array` is equal to `item`; if any element is equal to `item` then return `true`, and if none of the elements is equal to `item` return `false`.

Test: Call the function on the array `puzzle` with `linearSearch(puzzle, [])`; this should return `false`. Call the function on the array `puzzle` with `linearSearch(puzzle, [" ", 4, 1, " "])`; this should return `true`

[4 marks]

Now we want to use the Linear Search algorithm to look for each number from 1 to 4 in each row of an input array. We will break this down into two functions: the first `notAppear` will take an array called `row` to see if all numbers 1 to 4 appear in the row, and if a number does not, push it to an array; the second `possibilities` will produce an array of four elements where each element stores the output of `notAppear` for that row. Let's go through the tasks to complete these functions:

Task 3: Complete function `notAppear`, which takes as input an array called `row` and returns an array called `list`.

Goal: To write a function that returns an array (called `list`) that stores all numbers between 1 and 4 that do not appear in `row`.

Method: The function should call `linearSearch` with the inputs `row` and `i` where `i` goes from 1 to 4. If the function call returns `false` then the number `i` should be pushed to the array `list`. After checking whether all values of `i` appear, then `list` should be returned.

Test: Call the function on the first row of `puzzle` with `notAppear(puzzle[0]);` this should return `[2, 3]`.

[5 marks]

In the next task we will give a two-dimensional array of four elements where each element stores all the numbers that do not appear in a row of the input array.

Task 4: Complete function `possibilities`, which takes as input an array called `array` and returns an array called `poss`.

Goal: To write a function that returns an array (called `poss`) of four elements where each element stores what is returned by `notAppear` for that row.

Method: The function should call `notAppear` with the input `array[i]` for all values of `i` from 0 to 3. What is returned by `notAppear(array[i])` should be pushed to the array `poss`. Finally the array `poss` should be returned.

Test: Call the function on `puzzle` with `possibilities(puzzle);` this should return `[[2, 3], [1, 3, 4], [1, 2, 4], [3, 4]]`.

[5 marks]

3.3 Detecting entries of the puzzle array with the value `" "`

In order to generate candidate solutions for our Pseudoku puzzles, we need to find the entries of our Pseudoku puzzle array that need to be replaced with numbers from 1 to 4. In particular, we need to find the values of the rows and columns for entries in the puzzle array that store the value `" "`. In the next task, you will generate an array that will store these values of the rows and columns as arrays themselves. For example, for our Pseudoku puzzle above with 10 blank entries (stored as `" "` in the array), we have the following array of row and column indices where we can find these blank entries:

```
var blank = [[0,0],[0,3],[1,0],[1,1],[1,3],[2,1],[2,2],[2,3],[3,0],[3,2]];
```

Going from left to right, every element is a `[row,column]` index of an entry of our array `arr` that has the value `" "` stored there. In other words, for the element `blank[j]`, we have that `blank[j][0]` and `blank[j][1]` are respectively the row and column indices of a blank entry in our Pseudoku puzzle. If you select the element `blank[4]`, for example, then `arr[blank[4][0]][blank[4][1]]` will be `arr[1][3]`, which will have the value `" "`.

In the next task, you will write a function that will return such an array like `blank` as an output, if you give a Pseudoku puzzle like `puzzle` as an input parameter.

Task 5: Complete function `blankEntries`, which takes as input an array called `array`, and returns another array

Goal: To write a function that will return an array like `blank` above that lists all the row and column indices of the entries of your input array, called `array`, that have the value `" "`

Method: The function should create an array, called `blank`, look for entries that have the value `" "`, and then when it finds those entries, it should put the row and column indices into `blank`. When everything has been searched, the array `blank` should be returned

Test: Call the function using `blankEntries(puzzle);` the array
`[[0,0],[0,3],[1,0],[1,1],[1,3],[2,1],[2,2],[2,3],[3,0],[3,2]]` should be returned.

[5 marks]

3.4 Randomly generating a candidate for a solution

In the final task, you will randomly generate a single candidate for a Pseudoku puzzle. Each element of the candidate should use the possible values returned by the function `possibilities` for the input array. So to generate a candidate you need to randomly pick from elements of the array returned by `possibilities`. There are multiple ways of doing this, but you should use what it is returned by `blankEntries` and `possibilities` to randomly generate a candidate.

Task 6: Complete function `pickCandidate`, which takes the array called `array` as input and returns an array called `candidate`.

Goal: To write a function that will return an array called `candidate` that gives a candidate solution for the input puzzle called `array`.

Method: The function should randomly pick elements of `possibilities(array)` to make a candidate. It should also use `blankEntries` to know which elements of `possibilities(array)` to pick. The function should use `Math.random()` to randomly pick the elements.

Test: Call the function using `pickCandidate(puzzle);` the function should return something that could be a candidate.

[7 marks]
