Hyperiondev

TASK

Data Structures: 2-Dimensional Arrays

Visit our website

Introduction

WELCOME TO THE 2D ARRAY TASK

This task will introduce you to the concept of a 2-dimensional data structure, namely the 2-dimensional, or 2D, array. 2D arrays may also be referred to as a grid or nested list, as every element of the array is another array. In this task we will focus on the creation, manipulation, and usage of a 2D array.

Get in touch

Connect for support

Remember that with our courses, you're not alone! You can contact an expert code reviewer to get support on any aspect of your course.

The best way to get help is to login to Discord at **https://discord.com/invite/hyperdev** where our specialist team is ready to support you.

Our team is happy to offer you support that is tailored to your individual career or education needs. Do not hesitate to ask a question or for additional support!

USE CASES OF 2D ARRAYS

The JavaScript array that you have used in previous tasks is basically used to store multiple pieces of information in a linear order. You could think of this linear order as one single dimension, which you could visualise as a row.

In a 2-Dimensional array, the situation is very similar. The lists still contain pieces of information in a linear order, except a second dimension is added. You could visualise this second dimension as a column. There are many use cases where two dimensions of data are useful. Some common uses are images, where pixels are arranged in a row and column format, or board games, where the spaces on the board are arranged in rows and columns (e.g. chess or checkers).

A SHORT EXAMPLE

Consider the following node js output of the top 5 people in the **Forbes Real Time Billionaires List** along with their rank.:

```
[
    [ 'RANK', 'NAME                  '],
    [ '1.  ', 'Elon Musk             '],
    [ '2.  ', 'Bernard Arnault & Family'],
    [ '3.  ', 'Jeff Bezos            '],
    [ '4.  ', 'Bill Gates            '],
    [ '5.  ', 'Warren Buffet         ']
]
```

The above is meant to replicate the table and display the information in a neat and readable manner.

If we were to take a look at the code for the table displayed above it would look something like the following:

```
const forbesTable  = [
    [ 'RANK', 'NAME                  '],
    [ '1.  ', 'Elon Musk             '],
    [ '2.  ', 'Bernard Arnault & Family'],
    [ '3.  ', 'Jeff Bezos            '],
    [ '4.  ', 'Bill Gates            '],
```

```
        [ '5.  ', 'Warren Buffet           ']
      ];

      console.log(forbesTable);
```

The variable forbesTable defined in the code sample above is a 2D array where the first dimension is the ranking of the person in the Forbes list, and the second dimension is the name of the person.

WHAT TO NOTE IN THE EXAMPLE

When using **console.log()** the output is identical to the code, therefore in this case where we are using strings the spacing within every string value in the second dimension is important as this improves the readability of our code and provides a more accurate imitation of a table.

The first array (row) within the forbesTable variable contains the table headings. This is because every array within a 2D array can be considered a row and when imitating a grid/table using **console.log()**. Naturally, the first row should always contain any column headings.

In the 2D array above, every row contains a combination of both ranks and names. This improves the readability of our table and is also a good method to store records of important information. If we wanted to store multiple records of information a multi-dimensional array would be a great way to achieve this, where every dimension/column would contain multiple values of information on a single object.

An example of this is as follows:

```
// Define and Initialise 2-Dimensional Array
const forbesTable = [
    [ 'RANK', 'NAME                    ', 'NET WORTH', 'SOURCE' ],
    [ '1.  ', 'Elon Musk               ', '$215.6 B', ' Tesla, SpaceX' ],
    [ '2.  ', 'Bernard Arnault & Family', '$135.6 B', ' LVMH' ],
    [ '3.  ', 'Jeff Bezos              ', '$133.4 B', ' Amazon' ],
    [ '4.  ', 'Bill Gates              ', '$122.4 B', ' Microsoft' ],
    [ '5.  ', 'Warren Buffet           ', '$100.9 B', ' Berkshire H' ]
];
```

```
// Display 2-dimensional array
console.log(forbesTable);
```

DECLARING AND CREATING 2D ARRAYS

Creating a 2D array in JavaScript where you already have the values for the lists is relatively easy, and would work like the Forbes list example. The 2D array is declared in an almost identical manner to that of a normal array, the only difference being that every value in the array is another array. Every row in the 2D array contains every column or rather piece of information - attribute - of a single person entity. The 2D array was constructed this way to make it easier to read and to make it more closely represent the table or matrix format.

MANIPULATING A 2D ARRAY

Let us once again consider:

```
const forbesTable = [
    [ 'RANK', 'NAME                    ', 'NET WORTH', 'SOURCE' ],
    [ '1.  ', 'Elon Musk               ', '$215.6 B', ' Tesla, SpaceX' ],
    [ '2.  ', 'Bernard Arnault & Family', '$135.6 B', ' LVMH' ],
    [ '3.  ', 'Jeff Bezos              ', '$133.4 B', ' Amazon' ],
    [ '4.  ', 'Bill Gates              ', '$122.4 B', ' Microsoft' ],
    [ '5.  ', 'Warren Buffet           ', '$100.9 B', ' Berkshire H' ]
];
```

DISPLAYING/IDENTIFYING A ROW

If the values and their corresponding index in the 2D array given above were known, to display or target the row that contains Elon musk's information we could use indexing. Elon musk's row is at an index value of 1 (remember that arrays are numbered starting from 0).

```
// Display Elon Musk's row by indexing


console.log(forbesTable[1]);
```

Output:

```
[ '1.  ', 'Elon Musk                ', '$215.6 B', ' Tesla, SpaceX' ]
```

DISPLAYING/IDENTIFYING A COLUMN VALUE

To display or target a value from the row that contains Elon Musk's information, such as his net worth, we would have to index the 2D array to find the value of the row and that of the column containing the information we want. The value we want is at an index value of row 1, column 2.

```
/* The row containing Musk's information has an index value of 1, so the
first index is 1

The index of the network in the row it is 2, so the second index is 2 */

console.log(forbesTable[1][2]);
```

Output:

```
$215.6 B
```

HOW TO IDENTIFY A ROW WHEN THE INDEX IS UNKNOWN

If the index of a row is unknown and we know at least one column value in that row, we could still find and print out the row as well as the values within it. We could achieve this by using a for loop or the forEach() method. Let's consider each of these.

USING A FOR LOOP

As the index of the row that we are looking for is unknown, the steps to find a row when at least some of the column values is known, are as follows:

1. DECLARE KNOWN VARIABLE

Declare a variable that will hold a known value that is in the second dimension (a column value) of a row. In this case let's say we want Bill Gates's row of information.

We will first create a variable that will contain the string “Bill Gates”.

```
/* The known column value is 'Bill Gates'. We will use this to filter the array. We store the value in a variable */

let name = "Bill Gates";
```

2. CREATE FOR LOOP OR USE forEach() METHOD

The for loop would have to loop through every row in the array. Therefore when using a normal for loop we would have to increment i, continuing (logical test) for as long as i remains less than the length of the array.

```
let name = "Bill Gates";

/* the variable i (which represents rows) will start with a value of 0 and then be incremented until it is equal to 5 (which is the length of the table). Therefore logging forbesTable[i] i to the console  will print out every row in the 2D array */

for (let i = 0; i < forbesTable.length; i++){

    console.log(forbesTable[i]);

}
```

We could alternatively use The forEach() method. The forEach() method takes in a function which can receive up to two parameters. The first parameter is the value which will be the placeholder variable that will hold the value of each row in the 2D array.

The second (optional) parameter is the placeholder variable that will hold the value of the index of its current iteration.

Example with only first parameter:

```
let name = "Bill Gates";
```

```
// The row placeholder value will be every row within the 2D array

forbesTable.forEach((row) => {

    console.log(row);

})
```

Here's an example with a second, optional, parameter:

```
let name = "Bill Gates";

// The row placeholder value will be every row within the 2D array

forbesTable.forEach((row, index) => {

// Will display the row value in the console

    console.log(row);

// Will display the index of the row value in the console

    console.log(index);

})
```

OUTPUT OF BOTH THE FOR LOOP & forEach() METHOD

Example 1 Output

```
[ 'RANK', 'NAME                ', 'NET WORTH', 'SOURCE' ]

[ '1.  ', 'Elon Musk            ', '$215.6 B', ' Tesla, SpaceX' ]
```

```
[ '2.  ', 'Bernard Arnault & Family', '$135.6 B', ' LVMH' ]
[ '3.  ', 'Jeff Bezos              ', '$133.4 B', ' Amazon' ]
[ '4.  ', 'Bill Gates              ', '$122.4 B', ' Microsoft' ]
[ '5.  ', 'Warren Buffet           ', '$100.9 B', 'Berkshire H' ]
```

Example 2 Output

```
[ 'RANK', 'NAME                 ', 'NET WORTH', 'SOURCE' ]
0
[ '1.  ', 'Elon Musk            ', '$215.6 B', ' Tesla, SpaceX' ]
1
[ '2.  ', 'Bernard Arnault & Family', '$135.6 B', ' LVMH' ]
2
[ '3.  ', 'Jeff Bezos           ', '$133.4 B', ' Amazon' ]
3
[ '4.  ', 'Bill Gates           ', '$122.4 B', ' Microsoft' ]
4
[ '5.  ', 'Warren Buffet        ', '$100.9 B', 'Berkshire H' ]
5
```

Note that the square brackets, commas, and quotation marks are included because we are displaying the rows as the list it is.

If we wanted to remove the syntax, we could change the way we display the 2D-array by using a function that will deconstruct and reconstruct the 2D-array as a string so that we may display it in the manner that we want it .

Example of creating display method

```
// Define 2D-arrays

const rowWin = [

    ["O", "O", "O"],

    ["-", "-", "-"],

    ["-", "-", "-"]

];

// Define function that will deconstruct 2D-array and display it in an easy-to-read manner

function display(grid){

    // Create variable that will be displayed in the console once iterator method has run

    let output = "\n";

    // Use forEach() method to iterate over the 2D-array to deconstruct row value and add it to output variable

    // so that we may display the 2D-array in the manner we want
```

```
    grid.forEach((row) => {
        output += `${row[0]}| ${row[1]} | ${row[2]}\n`
    });
    // Display variable
    console.log(output);
}
 // Call display function with 'rowWin' 2D-array as argument
display(rowWin)
```

Display Function Output

```
0| 0 | 0
-| - | -
-| - | -
```

This is purely display and will not affect the manner in which we manipulate the 2D-Array.

In the real world when we are developing a game that uses a grid. We are required to implement the logic of that game/program. It is therefore best practice to first develop a game that is implemented in the terminal, as a first draft, before implementing the GUI (Graphic User Interphase).

3. DECLARE PLACEHOLDER VARIABLE:

Declare a placeholder variable that will hold a column value of the row in the current iteration. In this case we are attempting to filter through all rows in the 2D

array by searching for a row with a name value that is equal to “Bill Gates”.

Therefore the placeholder variable name will have the value of the current row at the index value of 1, because every name is at index 1 in every row.

Remember that every name has extra whitespaces that were added to improve the readability of our table. This means that we would have to remove the whitespace values. We can do this by using the **.trim()** method.

FOR LOOP EXAMPLE

```
let name = "Bill gates";

/* The row that we are currently on is at position i of the 2D array. The rowName will contain the name at index 1 of the row that we are currently on */

for (let i = 0; i < forbesTable.length; i++) {

    let row = forbesTable[i];

    let rowName = row[1].trim();

    console.log(rowName);

}
```

forEach() METHOD

```
let name = "Bill Gates";

/* The rowName value will contain the name at index 1 of every row in the 2D array */

forbesTable.forEach((row) => {
```

```
    let rowName = row[1].trim();

    console.log(rowName);

})
```

Output:

```
NAME

Elon Musk

Bernard Arnault & Family

Jeff Bezos

Bill Gates

Warren Buffet
```

4. WRITE CONDITION AND PRINT OUT ROW

Write a conditional statement to compare the known column value that we have against the current rowName value, and if they are equal console log the row.

FOR LOOP

```
let name = "Bill Gates";

for (let i = 0; i < forbesTable.length; i++) {

    let row = forbesTable[i];
```

```
    let rowName = row[1].trim();

/* If the known value, name, is equal to the name in the current row then
console log the row */

    if (name === rowName) {

        console.log(row);

    }

}
```

forEach METHOD

```
let name = "Bill Gates";



forbesTable.forEach((row) => {

    let rowName = row[1].trim();

/* If the known value, name, is equal to the name in the current row then
console log the row */

    if (rowName === name){

        console.log(row);

    }

})
```

```

```

Output:

```
[ '4.  ', 'Bill Gates              ', '$122.4 B', ' Microsoft' ]
```

HOW TO DISPLAY EVERY COLUMN VALUE ON A NEW LINE

When displaying or targeting every column value, steps 1 & 2 explained above remain the same. Afterwards we would have to add a nested for loop, to loop over every column value. Note that you may encounter a **ragged list** - in other words, a non-rectangular list where every row does not have the same number of column values.

To safeguard against this we should not hardcode the values in the condition that ends the for loop, and should instead use the length of the row to construct the for loop.

FOR LOOP

```
for (let i = 0; i < forbesTable.length; i++){
    let row = forbesTable[i];

    // For loop will make j the index of every element in the row
    // Therefore the col is at the index of j of the row
    for (let j = 0; j < row.length; j++ ){
        let col = row[j].trim();

        console.log(col);
    }
}
```

forEach() METHOD

```
forbesTable.forEach((row) => {

```

```
    // forEach() method will make col every next value in the row
    row.forEach((col) => {

        // .trim() method will remove all whitespaces
        col = col.trim();
        console.log(col);
    })
})
```

Output:

```
RANK
NAME
NET WORTH
SOURCE
1.
Elon Musk
$215.6 B
Tesla, SpaceX
2.
Bernard Arnault & Family
$135.6 B
LVMH
3.
Jeff Bezos
$133.4 B
Amazon
4.
Bill Gates
$122.4 B
Microsoft
5.
Warren Buffet
$100.9 B
Berkshire H
```

THE FOLLOWING METHODS CAN BE APPLIED TO BOTH THE ROWS AND COLUMNS IN A 2D ARRAY

ADDING A VALUE

.push() - Can be used to add an additional row at the end of a 2D array or to add an additional column at the end of a row.

```
// Create array
const  courses = ["IT", "Chemistry", "Law", "Psychology", "Mathematics"];


// .push method will add a value to the array
courses.push("Art");


// Display array in console
console.log(courses);
```

OUTPUT

```
[ 'IT', 'Chemistry', 'Law', 'Psychology', 'Mathematics', 'Art' ]
```

.splice() - Overrides the original array. It takes in the following parameters in corresponding order:

- The position in the array from which we start.
- The amount of values that we want to remove from the starting index. If we are adding this parameter may be given a value of 0.
- Every parameter that follows will be the next item that is added to the array.

```
// Create array
const  courses = ["IT", "Chemistry", "Law", "Psychology", "Mathematics"];
```

```
// Add 'Latin' And 'Physics' starting at index 2 in the list

// .splice(starting index, amount to remove, items added)

courses.splice(2, 0, "Latin", "Physics");


// Display array in console

console.log(courses);
```

Output:

```
[

    'IT',

    'Chemistry',

    'Latin',

    'Physics',

    'Law',

    'Psychology',

    'Mathematics'

]
```

REMOVING A VALUE

.pop() - Can be used to remove and return the last row in a 2D array or the last column in a row.

```
const  courses = ["IT", "Chemistry", "Law", "Psychology", "Mathematics"];


// Removes and returns the last element in the array

console.log(courses.pop());
```

```
console.log(courses);
```

Output:

```
Mathematics
[ 'IT', 'Chemistry', 'Law', 'Psychology' ]
```

.shift() - Can remove a row from the beginning of a 2D array or a column value from the beginning of a row.

```
const  courses = ["IT", "Chemistry", "Law", "Psychology", "Mathematics"];


// Removes and returns the first element in the array
console.log(courses.shift());


console.log(courses);
```

Output:

```
IT
[ 'Chemistry', 'Law', 'Psychology', 'Mathematics' ]
```

.splice() - Takes in at least two parameters, an index and the amount of values you want to remove from it. Given the following.

```
// Create array
const  courses = ["IT", "Chemistry", "Law", "Psychology", "Mathematics"];
// The following line will remove values starting from the value at the index
// of the first parameter.
```

```
// and will replace the amount of values specified by the second parameter

console.log(courses.splice(2, 2));

console.log(courses);
```

The values that were removed are also returned therefore they could also be stored in a variable.

Output:

```
[ 'Law', 'Psychology' ]

[ 'IT', 'Chemistry', 'Mathematics' ]
```

OVERRIDING A VALUE IN A 2D ARRAY

We can use indexing to overwrite both row values and column values. We identify the row/col we want to edit via indexing and then equate it to the new value we want to set it to.

CHANGE ROW VALUE

In the 2D array below:

```
const forbesTable = [
    [ 'RANK', 'NAME                    ', 'NET WORTH', 'SOURCE' ],
    [ '1.  ', 'Elon Musk               ', '$215.6 B', ' Tesla, SpaceX' ],
    [ '2.  ', 'Bernard Arnault & Family', '$135.6 B', ' LVMH' ],
    [ '3.  ', 'Jeff Bezos              ', '$133.4 B', ' Amazon' ],
    [ '4.  ', 'Bill Gates              ', '$122.4 B', ' Microsoft' ],
    [ '5.  ', 'Warren Buffet           ', '$100.9 B', 'Berkshire H' ]
];
```

We can change the row containing Elon Musk's information by doing the following:

```
// Create new array that will replace elon musk's row
```

```
const newArray = ["0", "0", "0", "0"];


// elon musk's row is the row at the index of therefore we target that row via
// indexing and equate it to new value
forbesTable[1] = newArray;


// display 2D array
console.log(forbesTable);
```

Output:

```
[
    [ 'RANK', 'NAME                  ', 'NET WORTH', 'SOURCE' ],
    [ '0', '0', '0', '0' ],
    [ '2.  ', 'Bernard Arnault & Family', '$135.6 B', ' LVMH' ],
    [ '3.  ', 'Jeff Bezos            ', '$133.4 B', ' Amazon' ],
    [ '4.  ', 'Bill Gates            ', '$122.4 B', ' Microsoft' ],
    [ '5.  ', 'Warren Buffet         ', '$100.9 B', 'Berkshire H' ]
]
```

CHANGE A COLUMN VALUE:

In the 2D array below:

```
const forbesTable = [
    [ 'RANK', 'NAME                  ', 'NET WORTH', 'SOURCE' ],
    [ '1.  ', 'Elon Musk             ', '$215.6 B', ' Tesla, SpaceX' ],
    [ '2.  ', 'Bernard Arnault & Family', '$135.6 B', ' LVMH' ],
    [ '3.  ', 'Jeff Bezos            ', '$133.4 B', ' Amazon' ],
    [ '4.  ', 'Bill Gates            ', '$122.4 B', ' Microsoft' ],
    [ '5.  ', 'Warren Buffet         ', '$100.9 B', 'Berkshire H' ]
]
```

We can change Elon Musk's name to 'Jeff' by doing the following:

```
// Create new array that will replace elon musk's row

const newName = "Jeff";


// elon musk's row is the row at the index of 1 therefore we target that row
via indexing

// Elon musk's name is at the index of 1 of that row so we target it via
indexing and equate it to the new value

forbesTable[1][1] = newName;


// display 2D array

console.log(forbesTable);
```

Output:

```
[
    [ 'RANK', 'NAME                  ', 'NET WORTH', 'SOURCE' ],
    [ '1.  ', 'Jeff', '$215.6 B', ' Tesla, SpaceX' ],
    [ '2.  ', 'Bernard Arnault & Family', '$135.6 B', ' LVMH' ],
    [ '3.  ', 'Jeff Bezos            ', '$133.4 B', ' Amazon' ],
    [ '4.  ', 'Bill Gates            ', '$122.4 B', ' Microsoft' ],
    [ '5.  ', 'Warren Buffet         ', '$100.9 B', 'Berkshire H' ]
]
```

Compulsory Task

Follow these instructions:

Within your Dropbox folder, in the same directory as this pdf, there should be a folder named **ticTacToe**. Download the ticTacToe folder along with the **ticTacToe.js** and **index.html** files within it and save them in the directory that you will be working in.

Within the ticTacToe.js file are the following variables and a function by the name of evaluatePlay.:

```
// Define 2D-arrays

const rowWin = [

    ["O", "O", "O"],

    ["-", "-", "-"],

    ["-", "-", "-"]

];


const colWin = [

    ["-", "X", "-"],

    ["-", "X", "-"],

    ["-", "X", "-"]

];


const diagonalWin = [

    ["-", "-", "O"],

    ["-", "O", "-"],

    ["O", "-", "-"]
```

```
]



const diagonalWinInverse = [

   ["X", "-", "-"],

   ["-", "X", "-"],

   ["-", "-", "X"]

];
```

Within the function, write the code that will determine whether a winning play has been made or not.

You may add additional parameters

The function should log the results to the console - that X has won and O has lost, or vice versa. The output of the following:

```
console.log(diagonalWin);
```

Should be as follows:

```
X Won

O Lost
```

Given that the game of tic tac toe is on a fixed grid of 3 X 3, you may hard code the positions when validating your grid.

Your function should be able to determine the winner of all the 2D arrays that are defined in the **ticTacToe.js** file

Compile, save, and run your program.

Here is a tip: When checking positions adjacent to a specific position in the grid, the following table might assist you in determining adjacent indexes:

NW position = currentRow - 1 currentCol - 1	N position = currentRow - 1 currentCol	NE position = currentRow - 1 currentCol + 1
W position = currentRow currentCol - 1	Current position = currentRow currentCol	E position = currentRow currentCol + 1
SW position = currentRow currentCol + 1	S position = currentRow +1 currentCol	SE position = currentRow + 1 currentCol + 1

Also ensure that when checking adjacent positions in the grid that you take into account that on the edges of the grid, you may go out of bounds.

Rate us

Share your thoughts

Hyperion strives to provide internationally-excellent course content that helps you achieve your learning outcomes.

Think that the content of this task, or this course as a whole, can be improved, or think we've done a good job?

Click here to share your thoughts anonymously.