**JavaScript Algorithm and Data structure Notes**

RPG Game

console.log(“Hello World”);  
  
- In javaScript there should be no space between line of codes and the semi-colon that end them. So let name=”alex” ; will cause an error.

- the script tag should always be added at the end of the html document ( just before the body closing tag, </body>). This is because if the script tag is placed at the top, the javascript file will run before the html body is loaded. All queries done by the js file on the html will then be null since it hasn’t been loaded yet.

- JavaScript interacts with html using the **Document Object Model , DOM.** The DOM is a tree of objects that represent the html. You can access the html using the document object which represent your entire html document

Query selector()

It is used to find specific html elements in your document. It takes as parameter or argument a CSS selector and **returns the first element that matches that selector.**  
*Ex: let monsterHealth = document.querySelector(“#monsterHealth”);*

The above will select what ever element in html document who has *monsterHealth* as id. In this case *#monsterHealth* By selecting it, we have the possibility of changing it’s content , color, transform etc.. In short we will have access to all of it’s attributes.

Note that when it comes to styles you have to pass through the style class of the the element selected. Eg **paragraph.style.display = ‘block’.**

For buttons who have even more interesting properties like **onclick**, we will be able to define or assign actions that occure when they are clicked.

*Button2.onclick = buttonaction;*

The above defines what happens when button2 is clicked. It assigns the function buttonaction to the onclick event.

**.innerText** class is used to access the Text content of an html element. The *.innerText* is designed to return text as it is been displayed on the webpage. Therefore it doesn’t return **~~whitespaces~~** . And also doesn’t handle text overflow

**.textContent** on the other hand keeps whitespaces and respects kind of the html structure stuff like text wrapping when assigned via js.s

**Objects** are declared in javascript like this : **{}** 🤔 This is weird🤔.

*newObject = {name = “anderson”, age=21, “favorite color” =”blue” }*

That his how to declare and set the attributes of an object in javascript. Notice there is no semicolon the end of variable declarations. It’s almost like python dictionaries. We can also assign functions to object variables.. insane.

Just look at this extract

**const locations = [**

**{**

**name: "town square",**

**"button text": ["Go to store", "Go to cave", "Fight dragon"],**

**"button functions": [goStore, goCave, fightDragon],**

**text: "You are in the town square. You see a sign that says \"Store\"."**

**}**

**];**

#Saturday rest day

Variable +=1 or Variable ++ in js.

If a variable is declared inside a block of code, it is only accessible to the code inside that block. It is called block scope.

Variables in javascript follow the block scope way. If you declare a variable inside an if, it remains in that if bla bla..

**Shift()** method removes the first element in an array and returns it.

Discover functions like .math() function which has mathematical functions, .floor() which rounds a given number down to the nearest integer. Etc.

The **innerHTML** property allows you tu access or modify the content inside an HMTL element using JavaScript.

The difference between .**innerText** and **.innerHTML**  is that innerText will only redefine the text of the element say a paragraph. On the other hand the innerHTML will not only put the text assign but also consider the html in the text assign. For instance via the .innerHTML , you could assign ‘The text is <strong>Bold</strong>’ to an element and the <strong> will actually apply . This will not be the case with .innerText.

This is how to call functions in conditions :

if (isMonsterHit()) {

   }

On

The logical OR operator will use the first value if it is truthy – that is, anything apart from NaN, null, undefined, 0, -0, 0n, "", and false. Otherwise, it will use the second value.

For example: num < 10 || num > 20.

**\n** Just like in c and c++ , it pushes any new content to new line.

**.push()** // adds element at the end of a list

**.pop()** // deletes element at the end of a list and return

**.shift()** // does the inverse of pop() ie. Removes first element of list and returns it.

**.includes(*value*)** // checks weather value passed in the brackets is in the list. It is the equivalent of ‘ value in list ‘ in python

That is it for arrays.

**Math.floor()** //rounds down to the nearest integer  
**Math.ceil()** // rounds up the number to the nearest integer  
**Math.random()** //gives a random number between 0(inclusive) and 1(exclusive)

That was it for that section.

Give your button element a type attribute set to button to prevent automatic form submission.

In programming, it is a **standard** practice to prefix a variable with **has**  or **is**  to indicate that it is a boolean.

Javascript kind of receives everything like a string. So wheather you defined a number in the html form or whatever, it will always arrive in js as string.

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

    if (!["+", "-", " "].includes(strArray[i])) {

      cleanStrArray.push(strArray[i])

    }

  }

**Regular expressions (regex)**

In regex, shorthand character classes allow you to match specific characters without having to write those characters in your pattern. Shorthand character classes are preceded with a backslash (\). The character class \s will match any whitespace character. Add this to your regex pattern.

const regex = /\+-/;

**\s** is used to represent space.

Use **\** to skip shorthand characters. Eg \+  
enclose class characters in square brackets   
 The g flag, which stands for "global", will tell the pattern to continue looking after it has found a match.

const regex = /[0-9]/; //matches string with numbers from 0 to 9.

The + modifier in a regex allows you to match a pattern that occurs one or more times. To match your digit pattern one or more times, add a plus after each of the digit character classes. For example: [0-9]+

Const regex = /\d/; // stands for ‘any digit’

Strings have a .match() method, which takes a regex argument. .match() will return an array of match results – containing either the first match, or all matches if the global flag is used.

Use can directly inject variables in strings by using **template litterals** . basically it functions like PHP injection. Example:

*Sentence = `My name is ${user\_name} , I am happy to be here}`*

Notice how the variable user\_name was injected in the in the string without using concatination.

Also, notice I am using ` and not ‘ neither “.

Use the **.querySelectorAll()** to select all the elements that match the selector. It return a NodeList . Don’t confuse with querySelector which return **only**  the first element which match the selector.

How ever you can also pass parameters to that .querySelectorAll() look at this example:

const entryNumber = targetInputContainer.querySelectorAll('input[type="text"]');

Here we are really just specifying that we wanna select all targetInputContainers but having input **of type** text.

Also , check this :

const entryNumber = targetInputContainer.querySelectorAll('input[type="text"]').length();

String literals are so cool.

One of those burnout days.. [heartbrokenemoji]

Hey check this out.  
targetInputContainer.innerHTML += HTMLString;

The above adds the html defined inside the **HTMLString** to the html of the targetInputContainer. It is worth nothing that HTMLString content was assigned using string litterals. Also, it updates inner HTML directly .

targetInputContainer.insertAdjacentHTML("beforeend",HTMLString);

On the other hand we have **.insertAdjacentHtml()** which we can use to specify the position of the new html to be inserted. (With this technique if there were inputs they won’t be erased. Ie it doesn’t change the state of the element) It takes two arguments.. the first is a string that specifies the position of the new html content. The second is the html content.

**setInterval()** function:

This function is so nice.. It is a javascript in-built function that repeatedly calls a function or executes a code snippet at specified intervals.

It’s basically a loop but in **asynchronous** .

It takes two parameters. The first is the function or code snippet and the second is the interval in milliseconds. It is a good practice assign the setInterval to a const so that later you can use the **clearInterval()** function to stop the loop. The clearInterval function takes one parameter which is the setInterval object. (that is why it should be assigned to a variable so that now we can just pass it to the clearInterval function.)

The best part of all this is that it is **Asynchronous.**  Meaning while the code is looping, the program still respond to other events in the code. Insane!

To avoid form submission of a button nested in a form set It’s type to **button**.