Section 3: Implementing, Testing and Modifying the Solution

Test Data

The testing phase is most crucial to testing the three primary features of the reaction: the ability to search chemical equations and add them to the reaction, the ability to change and calculate the amounts of the chemicals involved, and the ability to formulate balanced reactions.

The testing checklist essentially can be reduced to this:

- **Feature 1 - Searching:** Manually test the search algorithm, testing boundary conditions through a white-box method, use live/black-box testing to ensure that it works as intended
- **Feature 2 - Reacting:** Install a driver that tests the react() function black-box style, this process should be fully automated (apart from programming and adding some expected outputs)
- **Feature 3 - User Manipulation:** Test each aspect of the GUI from the programmer's perspective, attempting to find flaws in the way users can input information and how the program outputs information to account for such changes
- Pilot the software solution with users, to get live feedback on the operation of the solution

The primary algorithm for the first feature is unique in the fact that there is no right answer to how the function should operate. This means that an automatic driver process would not work, guaranteed that a human must look at the raw test results. This means that black-box testing the module would not be viable, however instead to acceptance test it - by ensuring that the search bar operates in an ergonomic fashion, providing appropriate suggestions. white-box function testing, also known as unit/integrated testing. This allows for the analysis of whether the boundary conditions do not break the code.

The test data used for this would simply be a set of strings that are incomplete versions of chemical names that appear within the database i.e. the query "Sodi" should produce search results "Sodium Chloride", "Sodium Ion", "Sodium Fluoride" etc. Note that the order may change depending on how often the chemical is included, however the feature where the program remembers the relevancy of each search is to be kept for a later version of the solution.

For the reaction function, a driver will be used. This, unlike the search bar, can be tested automatically, as there is an absolute, expected input for the operation of each. In order to test the react() function, however, a module/system-level black-box test must be performed. What will happen is that the driver will go through each and every possible input in the reaction storage text file, input the reactants, transfer them to the reaction, and then activate the react() function. Once an input is received, what needs to happen is that the driver needs to check:

1. That the ratios of both the reactants and products are correct. This requires an external text file input, done manually.
2. The products that have been formulated are correct.
3. The calculations of some predetermined/preset random float value and unit are correct and precise.

Note: The Driver image is shown on the next page.

From this, it is clear that the driver method also acts as a test involving large file sizes and interfaces between modules. Hence the driver is the most important testing method, allowing for the testing of pretty much a majority of the solution, as the react() function contains most of the operation - the GUI is simply an interface to talk to the reaction object.

Other tests involve simple debugging methods, specifically with the operation of the GUI, this requires manual black-box testing i.e. checking if each of the user inputs can lead to problems like attempting to include large numbers in the text boxes, or going through each possible unit selection and checking the calculation results to assure that it works. This can also help confirm that the calculate() function works adequately.

Another method that can be used to test the solution would be to get an external user to attempt to use the solution. This can help as other user inputs may test certain aspects of the solution that was not tested during the GUI testing phase.

```
function driver() {
    let faults = [];

    //Automatically add chemicals and react
    const autoReact = (chems) => {
        console.log(chems);
        for (let chem in chems) {
            addChemicalToStage({id: chems[chem]});
            addChemicalsToReaction();
        }
        reactButton();
    }

    //Check if the resulting formula object is as intended
    const checkValid = () => {
        let valid = true;
        const id = eq1.getReactDictR();

        for (let c in eq1.reactants[0]) {
            if (eq1.reactants[1][c] != parseInt(driverInst[id].ratio[c])) {
                valid = false;
            }
            if (eq1.reactants[4][c] != driverInst[id].state[c]) {
                valid = false;
            }
        }
        for (let c in eq1.products[0]) {
            console.log(driverInst[id].ratio);
            c = parseInt(c);
            if (eq1.products[1][c] != parseInt(driverInst[id].ratio[c + eq1.reactants[0].length])) {
                valid = false;
            }
            if (eq1.products[4][c] != driverInst[eq1.getId(true)].state[c + eq1.reactants[0].length]) {
                console.log(eq1.products[0][c], driverInst[eq1.getId(true)].state[c + eq1.reactants[0].length], eq1.products[4][c]);
                valid = false;
            }
        }

        return [valid, driverInst[id], reactdict[id], eq1];
    }

    //Main subroutine loop
    for (let c in reactdict) {
        if (!reactdict[c].std) {
            autoReact(c.split('+'));
            console.log("DRIVER: ", eq1);
            let check = checkValid();
            if (!check[0]) {
                faults.push(check[1], check[2], check[3]);
            }
            deleteButton();
        }
    }

    //Print Results
    if (faults.length == 0) {
        console.log("DRIVER RESULTS:\nAll good!");
    }
    else {
        console.log("DRIVER RESULTS:");
        for (let c in faults) {
            console.log("ERROR:"+c);
            console.log("inDriverInst:", faults[c][0]);
            console.log("inReactDict:", faults[c][1]);
            console.log("inReaction:", faults[c][2]);
        }
    }

}
```

Peer Report (From Connor IW)

```
BEGIN HCF(nums)
	index = 0
	factors = Empty Array

	FOR i = 0 TO nums.length - 1 STEP 1
		IF nums(i) = 0 AND index = 0 THEN
			increment index
		ENDIF
		IF nums(i) > 0 AND nums(i) < nums(index) THEN
			index = i
		ENDIF
	NEXT

	FOR i = 0 TO nums(index) STEP 1
		div = true

		FOR n = 0 TO nums.length - 1 STEP 1
			IF nums(n) % i != 0 THEN
				div = false
			ENDIF
		NEXT

		IF div = true THEN
			Add i to factors
		ENDIF
	NEXT

	RETURN factors(factors.length - 1)
END HCF
```

The above algorithm is a binary search which searches through an ordered list in order to find a given number.

Desk check 1:
Check if array is working and edge cases
Test input:
array = [-1000, -256, -10, -3.14 , 0 , 3.14, 10, 256, 1000]
num = 10

run	index	Found	min	max	mid	array(mid)
0	-1	false	0	8	4	0
1			5	8	7	256
2			5	6	6	10
3	5	true				

This will return 5 when it should have returned 6 which is wrong

This can be solved by changing ‘index = min’ to ‘index = mid’

Desk check 2
Test for if the number is not in the array
Test input:
array = [-1000, -256, -10, -3.14 , 0 , 3.14, 10, 256, 1000]
num = 42

run	index	Found	min	max	mid	array(mid)
0	-1	false	0	8	4	0
1			5	8	7	256
2			5	6	6	10
3	-1		7	6	7	256

Algorithm will return -1 which would suggest that the number is not in the array, this would rely on the routine calling this subroutine to be able to interpret -1 as not part of the array

Overall this algorithm works well, however It is fundamentally broken as it will return the wrong answer majority of the time due to the error of index = min when it should be index = mid. This is most likely due to a simple typo however has created a logical error. The round function here rounds up if a the midpoint lies between two integers however this could be better specified

```
BEGIN binarySearch(num, array)
    min = 0
    max = length of array - 1
    found = false
    mid = Average of (min, max) Rounded
    index = -1
    WHILE !found && max >= min
        IF num == array(mid) THEN
            found = true
            index = min
        ELSE IF num < array(mid) THEN
            max = mid - 1
        ELSE IF num > array(mid) THEN
            min = mid + 1
        ENDIF
        mid = Average of (min, max) Rounded
    ENDWHILE
    RETURN index
END binarySearch
```

This code first locates the lowest number in the array and then brute forces divisibility checks of all the numbers ranging from 0 to the lowest number. If this number is divisible by all 3 then it will add to the array factors. It then returns the last element in the divisibility array.

Input data: nums = [36, 12, 24]

Run	index	nums(index)	nums(i)	i	n	nums(n)	factors	div
0	0	36	36	0			[ ]	
1	1	12	12	1				
2	1	12	24	3				
3 (note: fist loop has ended)	1	12		0	0	36		true

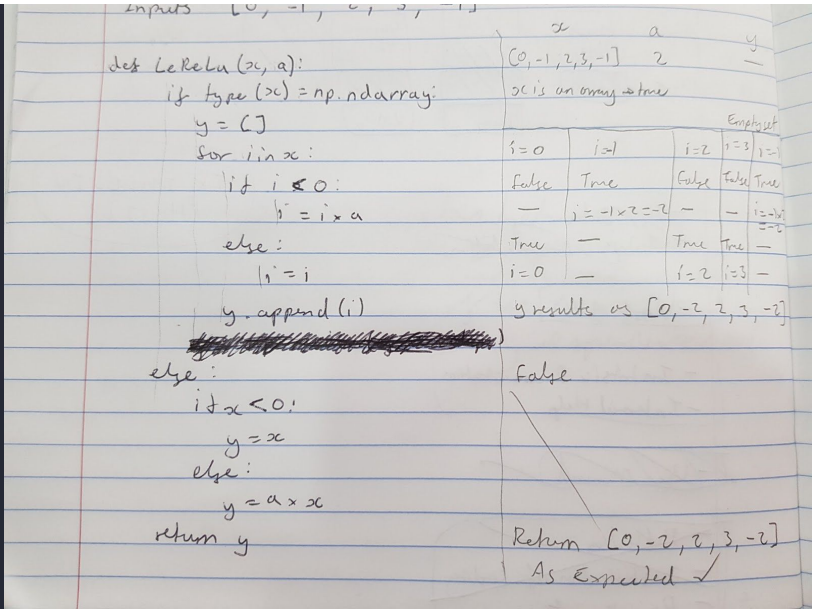
Division by 0 error would then halt this code on line ‘ IF nums(n) % i != 0 THEN’ as at this point i = 0 and nums n = 36, 36 % 0 does not work as 36 cannot be divided by 0,
To fix this problem, i should start at 1 instead of 0

Some other improvements that could be made would be to write ‘==’ instead of ‘=’ on lines that are comparing values. Also major speed improvements could be made as since you are only finding the highest factor, there is no need to start at i = 0, instead you should start at i= nums(index) and step back to 0 so once you find a factor that works for all elements in nums

then it can simply return that number and stop iterating through. This will mean you no longer need to check every value and can save many cycles.

Peer Report (To Connor IW)

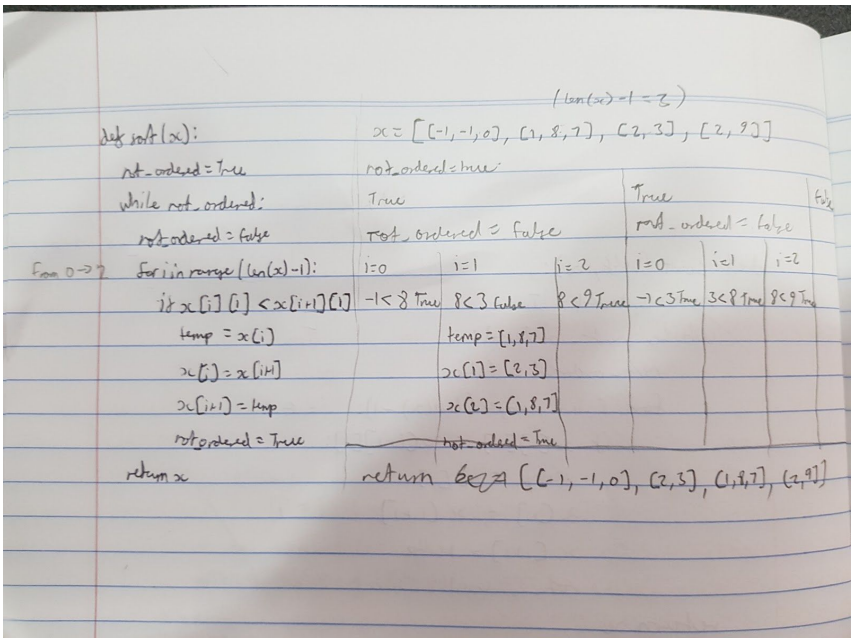
```
# Leaky Rectified Linear units
def LeReLu(x, a):
    if type(x) == np.ndarray:
        y = []
        for i in x:
            if i<0:
                i = i*a
            else:
                i=i
            y.append(i)
        y = np.resize(y, x.shape)
    else:
        if x<0:
            y=x
        else:
            y=a*x
    return y
```



The above algorithm, written in python is meant to:

1. Check if the inputted parameter x is an array or an integer
2. If it's a array go through each item in the array, if it's an integer, just check the single value
3. If the value is less than zero, and x is in an array, the value is multiplied by the second parameter a
4. If the value is greater than or equal to zero value, and x is an integer, the value is multiplied by the second parameter a
5. For all other combinations of the data type of x and the value, the value remains the same
6. The new value is appended to a return array/integer y

Given this algorithm and the desk, it is clear that the operation of it works as intended. The code developed is also quite modular, in terms of the data types used, as x can be both an integer or an array. And specifically for python, x can be a numpy array, meaning that it could also be a matrix, hence the np.resize function. One area of improvement, would be to explain the operation of the function using commented lines. This would help people to understand how the function works, as the fact that the rules (i.e. what happens to the value) swap around depending on the data type. This can be confusing as if x is an integer, it will output x but if x was an array, containing the same integer, it would output x*a.




```python
def sort(self, x):
    ''' sort an array of arrays in decending order based off the second item '''
    not_ordered = True
    while not_ordered:
        not_ordered = False
        for i in range(len(x)-1):
            if x[i][1] < x[i+1][1]:
                temp = x[i]
                x[i] = x[i+1]
                x[i+1] = temp
                not_ordered = True
    return x
```

The above algorithm, written in python is meant to sort an array of integer arrays by the second item of each array element.

This algorition also works well, however due to its development in python, it is important to remember that range(len(x)-1) would produce the numbers 0, 1, 2, … len(x)-2. This may be confusing, and might require some commenting. The code is quite good at fulfilling this purpose in an efficient manner, using a bubble sort to achieve a simple solving method. The modularity of this program can be improved by allowing a third parameter that determines what item in the array the sorting will be based on, setting it to a default `index = 1`. This would mean that the 7th line would be `if x[i][index] < x[i+1][index]`.

Justification of Code

There are three key aspects that need to be considered when justifying the code (or essentially the quality of the solution): the intrinsic documentation; the features and the meeting of the user requirements; the functionality of the solution. In assessing these factors, I will use the example of a specific function addChemicalsToReaction() from main.js

The intrinsic documentation, or the maintainability of the solution is the most important part of the code, allowing for other people to easily understand the operation of the solution. Developing a set of whitespaces and control structures are involved in this process. As seen from the example, there is a neat progression of logic from the beginning, which seeks to prevent the user from entering incorrect inputs (Duplicate values and Null values), then to create an instance of an object, fill that object with information, append that information to other important lists/variables in the document, update the screen display with the new information by removing it from the ‘stage’ to the ‘auxiliary results display’ (the ‘stage’ being where you can manipulate the information about the chemical i.e. type, amount, units. And the ‘auxiliary results display’ being the place where the reaction details are stored and presented). There are also comments, which aid in the ability for people to understand what exactly the function is doing. It is also a one task per subroutine function, as the goal of this function is simply to transfer information about the user-inputted chemical from one section of the GUI, the searching stage, to the reaction interface, where the primary function of the software is carried out.

The features of the solution are also shown below, a primary feature of the solution is your ability to search for a chemical and add it to a reaction, from which you can perform reaction calculations on. This function is involved in the search feature as it is part of the GUI section where chemicals can be searched and added to the reaction. It acts essentially as an interface between the features, allowing a clean transfer with the press of a button labeled ‘Add’. This is what allows the solution to meet the user requirements, as part of the requirements involves the ability to search for chemicals, originating from a .txt file, manipulate the information about the chemical, and then adding it to the reaction so that it may be reacted. This code helps achieve the user requirement, as it allows the user to add the chemicals to the reaction.

The functionality of the solution is also shown below as well. Essentially, the functionality is the ability for the solution to work effectively and without error (or minimal error). Due to testing the module, using techniques such as drivers or desk checks, I have been able to ensure that this section of the code prevents users from damaging the solution, or the solution from damaging itself.

In conclusion, I think that, from this specific example, my code is justified as it makes part of a quality solution that is maintainable, functional, contains many features (i.e. is expansive in its scope), and meets the user requirements.

```
function addChemicalsToReaction() {
    //Prevent Duplicates/Null inputs
    let sameflag = false;
    for (let c in eq1.reactants[0]) {
        if (eq1.reactants[0][c].formula === formulaOnStage) {
            sameflag = true;
        }
    }
    if (formulaOnStage === "") {
        alert("You need to search and add a chemical to the stage.");
        return null;
    }
    if (sameflag) {
        alert("You can't double up on chemicals. Select a different chemical.");
        return null;
    }

    //Create Chemical Instance
    let addition = new chemical(formulaOnStage, "none", "none");
    addition.name = addition.getDriver("name");
    addition.type = addition.getDriver("type");
    addition.ion = addition.getDriver("ion");
    addition.state = addition.getDriver("state");

    console.log(addition);

    //Input into reation object
    addConditions(eq1);
    eq1.reactants[0][auxcounter] = addition;
    eq1.reactants[1][auxcounter] = 1;
    eq1.reactants[2][auxcounter] =
    eq1.convertUnits(addition, parseFloat(document.getElementById("chem_n").value), document.getElementById("chem_u").value, "mol");
    eq1.reactants[3][auxcounter] = "mol";
    eq1.reactants[4][auxcounter] = addition.state;



    //AuxillaryCondSet
    auxcondset.push("reactants_" + auxcounter);
    auxcounter++;
    //console.log("lol", auxcondset);

    console.log(eq1.reactants);

    //Update displayReact/output
    displayResults(eq1.reactants, 'reactants');
    output.innerText = displayReact(eq1, false);
    ConditionCheck(true);


    //Remove chemical from stage
    let textbox = document.getElementById("chem_n");
    let selectbox = document.getElementById("chem_u");
    textbox.value = 1;
    selectbox.value = "mol";

    let stagename = document.getElementById("chemicalonstage");
    stagename.innerHTML = ".";
    stagename.style.color = "transparent";
    formulaOnStage = "";
}
```

Reflection

In reflection, I think that this software project provided an interesting experience into the world of software development. This is because there were many unique challenges that I faced in developing this solution that was not present in previous projects.

The first notable difference was how I organised myself timewise. This project was unique because I had more freedom to decide how I would complete the project. This is combined with the fact that the project was planned over the course of three terms, an incredibly large amount of time. In the beginning, I think that I was able to keep on track substantially well, I was ahead by a few weeks. However, due to more and more interference from other subjects, I fell behind in the development of the solution. This led to a very stressful last two weeks. This affected my ability to construct a well-maintained solution. However, I think that because of my earlier experience in software development, I was able to handle intense time pressure.

In terms of my ability to keep to a more structured approach, the development of the solution very quickly developed into a RAD development. This was as expected, as I am the only person programming the solution. The scope of this project also led to a lot of bodging, which meant that the resultant solution would emulate a prototype more than the final product.

A positive area of work would be in the complexity of the solution. This is probably the only available method to calculate chemical reactions. This, still required a bodged method, as the equalize() function essentially runs on a trial and error basis, but considering that the only proper algorithm to perform such a task is the subject of a PhD thesis (yes I did my research), I think the solution I developed would already represent well-enough my skills.

Another positive note is that this solution really acts as the combination of all of my knowledge in software development. This is probably the only project where I used almost all aspects of my Y9-12 knowledge of IST/SDD programming and SDD theory knowledge. For example, the use of a driver to test the react() function.

I think that improvements could be made in the way I program the solution, the specific methods and methodologies that I employ in developing the solution. For example, I did not know how to use promises, which led to a lot of worrying about asynchronicity catching up to how I developed my solution. This is compounded with the fact that I need to improve how I prepare for the development of such a (relatively) large project. This is evident in the fact that I should have been more rudimentary in the development of my code, however, errors are inevitable and that is what led to an italian kitchen level of spaghetti in my repository.

In conclusion, I think that I did a great job in developing my solution. And even if there were to still be errors, I think that it gave me a great learning experience in how to program larger solutions. So, the ultimate summary is this: next time, I'll use C#.