**Intro to the Debugging Challenges**

Debugging is the process of finding and fixing or resolving the problems within a computer program. It is a frustrating task to do but it helps the developer to review the code and check if the code works as intended.

There are basically three types of errors:

a) Syntax errors. Grammar errors in the use of the programming language that prevent a program from running.

b) Runtime errors. Errors that occur during the execution of the program. Asks the program to do something that is unable to do. These kinds of errors are difficult to find and fix than syntax errors.

c) Logical or Semantic errors. Errors related to the logic of the program. A code with logic or semantic errors may produce wrong or different results from what it is expected.

Here are my solutions to the challenges:

**1. Use the JavaScript Console to Check the Value of a Variable**

let a = 5;

let b = 1;

a++;

console.log(a); //ans

let sumAB = a + b;

console.log(sumAB);

**2. Understanding the Differences between the freeCodeCamp and Browser Console**

let outputTwo = "This will print to the browser console 2 times";

console.log(outputTwo);

let outputOne = "Try to get this to log only once to the browser console";

console.clear();

console.log(outputOne);

**3. Use typeof to Check the Type of a Variable**

let seven = 7;

let three = "3";

console.log(seven + three);

console.log(typeof seven);

console.log(typeof three);

**4. Catch Misspelled Variable and Function Names**

let receivables = 10;

let payables = 8;

let netWorkingCapital = receivables - payables;

console.log("Net working capital is: ${netWorkingCapital}");

**5. Catch Unclosed Parentheses, Brackets, Braces and Quotes**

let myArray = [1, 2, 3];

let arraySum = myArray.reduce((previous, current) => previous + current);

console.log(`Sum of array values is: ${arraySum}`);

**6. Catch Mixed Usage of single and Double Quotes**

let innerHtml = "<p>Click here to <a href=\"#Home\">return home</a></p>";

console.log(innerHtml);

**7. Catch Use of Assignment Operator Instead of Equality Operator**

let x = 7;

let y = 9;

let result = "to come";

if(x == y) {

result = "Equal!";

} else {

result = "Not equal!";

}

console.log(result);

**8. Catch Missing Open and Closing Parenthesis After a Function call**

function getNine() {

let x = 6;

let y = 3;

return x + y;

}

let result = getNine();

console.log(result);

**9. Catch Arguments Passed in the Wrong Order When Calling a Function**

function raiseToPower(b, e) {

return Math.pow(b, e);

}

let base = 2;

let exp = 3;

let power = raiseToPower(base, exp);

console.log(power);

**10. Catch Off By One Errors When Using Indexing**

Off by one errors(sometimes called OBOE) crop up when you're trying to target a specific index of a string or array (to slice or access a segment), or when looping over the indices of them. JavaScript indexing starts at zero, not one, which means the last index is always one less than the length of the item. If you try to access an index equal to the length, the program may throw an "index out of range" reference error or print undefined.

function countToFive() {

let firstFive = "12345";

let len = firstFive.length;

for (let i = 0; i < len; i++) {

console.log(firstFive[i]);

}

}

countToFive();

**11. Use Caution When Reinitializing Variables Inside a Loop**

function zeroArray(m, n) {

// Creates a 2-D array with m rows and n columns of zeroes

let newArray = [];

// let row = [];

for (let i = 0; i < m; i++) {

// Adds the m-th row into newArray

let row = []; //moved this object

for (let j = 0; j < n; j++) {

// Pushes n zeroes into the current row to create the columns

row.push(0);

}

// Pushes the current row, which now has n zeroes in it, to the array

newArray.push(row);

}

return newArray;

}

let matrix = zeroArray(3, 2);

console.log(matrix);

**12. Prevent Infinite Loops with a Valid Terminal Condition**

function myFunc() {

for (let i = 0; i <= 4; i += 2) {

console.log("Still going!");

}

}