ISCG6420

Exercise: JavaScript Fundamentals

This exercise will explore the fundamental concepts of JavaScript, including variable types, data types, operators, type functions, and structures.

This exercise can be performed independently or supported by group discussion.

Getting Started

Download the provided HTML, CSS, and JS file from Moodle. Open an existing website project or begin a new project in your IDE (VSCode, etc). Add the downloaded files to your project inside a folder called “week3”. Double-click the HTML file in the week3 folder to open it.

With the Live Server (Five Server) extension enabled, click the “go live” button at the bottom-right of VSCode to launch a live version of the website. Your web browser should open a web page like this:

The webpage has some element created for you:

- nav bar at the top to return to your project home page
- Console log output redirect (displays the console log output on the webpage)
- Sample data for testing
- Basic calculator UI components

Starting with JavaScript

In VSCode, double-click the “add.js” file in the week3 folder to open the file. Inside you will find the following code:

```
window.onload = function () {
    // ignore this for now
    consoleRedirect();

    // Add your code below this line


    // Add your code above this line
```

The first line of code is declaring a function (with no name) and telling the window to run the function when it loads (*onload* is the event that occurs when the web page loads).

The second line of code calls a function called *consoleRedirect*(). This function moves some of the console log output from the web browser dev tools to the text area on the web page. It is not relevant to this exercise.

All of your code can be written between the two comment lines at the bottom of the *onload* function.

Variable types

There are three main variable types in JavaScript: let, const, var.

- let: Block scope variable declaration. Used for variables that can change value.
- const: Block scope variable declaration. Used for variables which can only have their value set once.
- var: Function scope variable declaration. Very commonly used in older JS code. No longer recommended to use.

Variable Declaration

Create an if statement with a condition of “true”. Inside the if statement block, create one variable of each type:

```
if (true) {
    let myLet = "I am a let";
    const myConst = "I am a const";
    var myVar = "I am a var";
}
```

1) Below the if statement (outside the curly braces) attempt to print *myLet* to the console:

```
if (true) {
    let myLet = "I am a let";
    const myConst = "I am a const";
    var myVar = "I am a var";
}

console.log(myLet);
```

2) Running this code will result in an error:

Console log output

```
Error line 13: ReferenceError: myLet is not defined
```

This error occurs because the let variable only exists within the code block it's declared in. Once the if statement block has completed running, *myLet* no longer exists.

3) Replace the logged variable with *myConst*. It will also produce an error as it is also a block scope variable:

Console log output

```
Error line 13: ReferenceError: myConst is not defined
```

4) Replace the logged variable with *myVar*. It will work without error:

Console log output

```
I am a var
```

This is because var is a function scope variable, and will exist inside the nearest ancestor function regardless of declaration location. This is also the downside of using var, as it can exist outside of intended scope unintentionally.

Rule of thumb: avoid using var where possible.

Value integrity

5) Convert your written code to a comment by highlighting the code, holding the CTRL key down, then press and release 'K', then press and release 'C', and release CTRL. This will stop the code from running and causing errors without deleting the code.

```
// if (true) {
//     let myLet = "I am a let";
//     const myConst = "I am a const";
//     var myVar = "I am a var";
// }

// console.log(myVar);
```

6) Create a new const variable called *myConst* and set its value to "Hello ".

7) Underneath, attempt to append "World!" to *myConst using the += operator*:

```
const myConst = "Hello ";
myConst += "World!";
```

8) Save the file and check the console log. It will contain an error:

Console log output

```
Error line 16: TypeError: invalid assignment to const 'myConst'
```

Once a const type has its value assigned, it cannot be changed. By appending "World!" to *myConst* we attempted to change its value and it resulted in a TypeError.

9) Perform the same steps using a *let* variable instead of a const, and log the result:

```
let myLet = "Hello ";
myLet += "World!";
console.log(myLet);
```

Console log output

```
Hello World!
```

Operators

Operators are actions that relate to variables and result with a comparative output. Such as:

Basic operators	A + B, A – B, A * B, A / B
Assignment operators	A += B, A -= B, A *= B, A /= B
Incremental operators	A++, ++A, A--, --A
Exponential operator	A ** B
Modulus operator	A % B

10) Create two variables called A and B. Attempt to use at least one of each operator category on A and B and log the output. It may help to add additional information to the log to identify information more easily:

```
let a = 10;
let b = 5;

console.log("a + b = ")
console.log(a + b);

console.log("a += b = ");
console.log(a += b);

console.log("++a");
console.log(++a);

console.log("a ** b");
console.log(a ** b);

console.log("a % b");
console.log(a % b);

console.log(a);
console.log(b);
```

Console log output

a + b =
15
a += b =
15
++a
16
a ** b
1048576
a % b
1
16
5

11) Replace the starting value of variable B with a string “5” and check the output. The operations still occur, but the behaviour and output are different. The same is true if the value of variable B is set to true:

b = “5”;

Console log output

a + b =
105
a += b =
105
++a
106
a ** b
13382255776
a % b
1
106
5

b = true;

Console log output

a + b =
11
a += b =
11
++a
12
a ** b
12
a % b
0
12
true

String methods and properties

JavaScript has many functions for working with strings. To use string methods, use dot notation on a string and append the method call.

12) Create a new let variable and assign a string value to it:

```
let myString = "Hello World!";
```

13) Call the length property using dot notation:

```
let myString = "Hello World!";
myString.length
```

14) Wrap the second line in a console log:

```
let myString = "Hello World!";
console.log(myString.length);
```

Save your work and check the console log output. It will show the number of characters in your string:

Console log output

```
12
```

15) Create a new console log call, and inside use the substring() method:

```
let myString = "Hello World!";
console.log(myString.length);
console.log(myString.substring());
```

The output will show the first character of your string. The substring method can also take parameters to specify which character sequence you wish to grab.

Console log output

```
12
H
```

16) Add two parameters to the substring method, 2 and 10:

```
let myString = "Hello World!";
console.log(myString.length);
console.log(myString.substring(2, 10));
```

Which results in the substring from character [2](inclusive) to character [10](exclusive) :

Console log output

```
12
llo Worl
```

17) Create a new console log call, and inside use the array index notation on the string to access index 3:

```
console.log(myString.substring(2, 10));
console.log(myString[3]);
```

This will access a single character, as strings are able to be treated as a read-only array of characters.

Console log output

```
12
llo Worl
l
```

Note: Attempting to access an array index outside the range of characters will result in an undefined value returned:

```
Error line 84: TypeError: content is undefined
```

18) Create a new console log call, and inside use the **toUpperCase()** method on the string:

```
let myString = "Hello World!";
console.log(myString.toUpperCase());
```

This results in all string characters being converted to upper case:

```
HELLO WORLD!
```

19) Replace the **toUpperCase()** method with **toLowerCase()** :

```
let myString = "Hello World!";
console.log(myString.toLowerCase());
```

This results in all string characters being converted to lower case:

```
hello world!
```

20) Create a new console log call, and inside use the **split()** method with " " as a parameter:

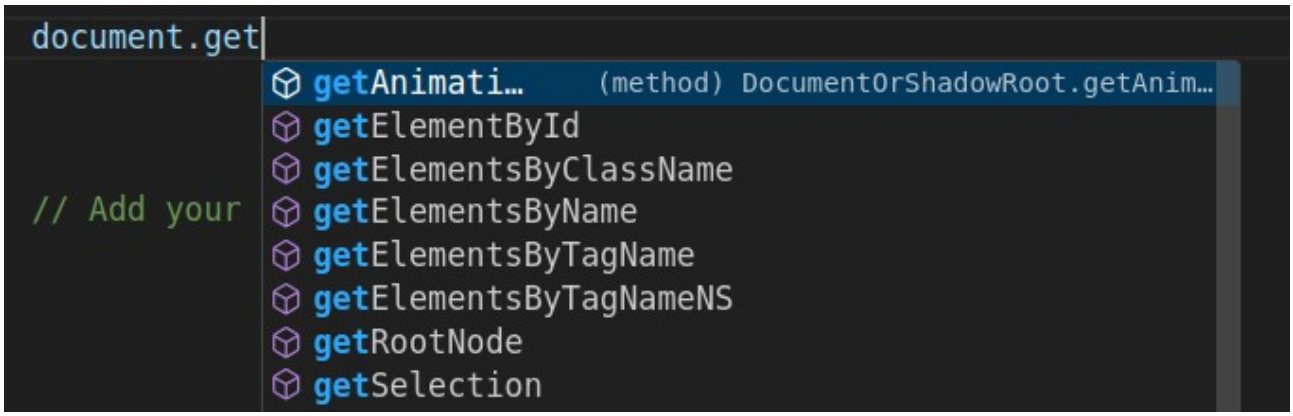
```
let myString = "Hello World!";
console.log(myString.split(" "));
```

The split method will separate the string at each provided character (space) and return an array of substrings:

```
array: [ Hello, World! ]
```

DOM access

JavaScript can be used to read and write content of the website via the Document Object Model (DOM). Accessing element of the website can be achieved using a selector:



21) Fetch the element with ID “sampleSentance” using the getElementsById(“ID”) method. Log the output:

```
let element = document.getElementById("sampleSentance");
console.log(element);
```

```
[object HTMLParagraphElement]
```

22) Inside the element object are properties and methods we can use. To access the text content of a paragraph, call the innerText property:

```
let element = document.getElementById("sampleSentance");
console.log(element.innerText);
```

```
This is sample text for you to use while testing JavaScript.
```

23) Modify the content of the selected paragraph by setting the innerText equal to a new string:

```
let element = document.getElementById("sampleSentance");
element.innerText = "This is a new string.";
```

This will update the contents of the selected web page element:

Data to work with

This is a new string.

24) Select all elements that belong to class “sampleData” using the getElementsByClassName(“class”) and convert the result to an array using Array.from(collection) :

```
let elements = Array.from(document.getElementsByClassName("sampleData"));
console.log(elements);
```

```
array: [ [object HTMLParagraphElement], [object HTMLParagraphElement],
[object HTMLParagraphElement], [object HTMLParagraphElement] ]
```

Instead of logging the array we can loop over the array elements and read the text content of each element.

25) Remove the console log line and create a for loop from 0 to the array length:

```
let elements = Array.from(document.getElementsByClassName("sampleData"));

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

}
```

26) Inside the for loop, console log the innerText of each element using the index i:

```
for (let i = 0; i < elements.length; i++) {
    console.log(elements[i].innerText);
}
```

This will output the following to the console:

Console log output

This is a new string.
123456
true
a,b,c,d,e,f

Exercise complete.

Save your work and commit the changes to your repository.