Hands-on Lab: JavaScript - Browser Console

Estimated Time: 25 minutes

The purpose of this lab is to practice using JavaScript in the browser console, to reinforce your understanding of certain concepts.

Objectives

After completing this lab, you will be able to:

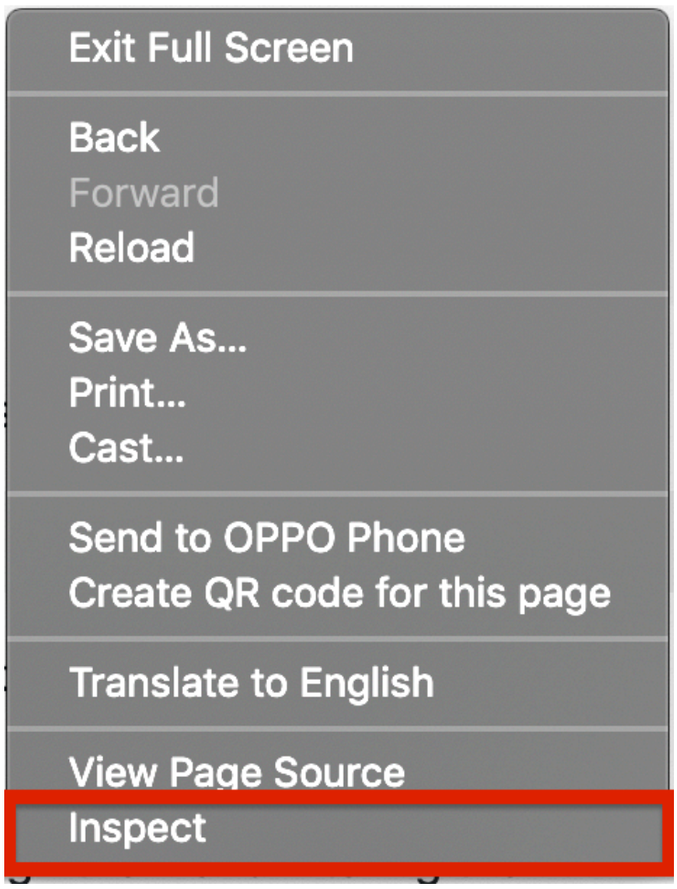
1. Write and run JavaScript on the browser console
2. Create variables, work with conditional statements, create loops, and define methods in JavaScript.

Task 1 - Open the browser console

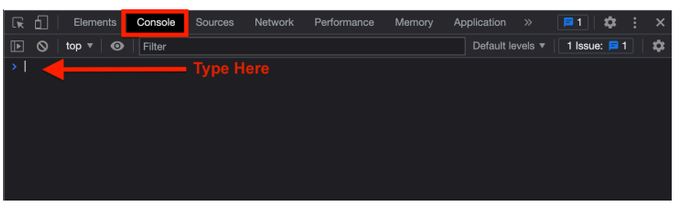
In this task, we are going to run JavaScript code in the browser console. The Chrome browser uses V8, which is Google's open source high-performance JavaScript engine.

1. Open a new, blank tab in your browser. You can do this by clicking on `Ctrl` + `T` (Windows) or `Command` + `T` (Mac).

2. Right-click anywhere on the new blank browser tab and choose **Inspect** or **Inspect Element**, depending on the browser you are using. The image below is for the Chrome Browser.



3. A developer window should open on your screen. Go to the **Console** tab, as shown below. You will see a command prompt. You can run the rest of the tasks there.



4. If your console has any logs printed, clear it by running the following command. This is not mandatory, but will you help have a fresh start.

```
clear()
```

If your browser does not support this, use the `console.clear()` command instead.

Task 2 - Running JavaScript commands

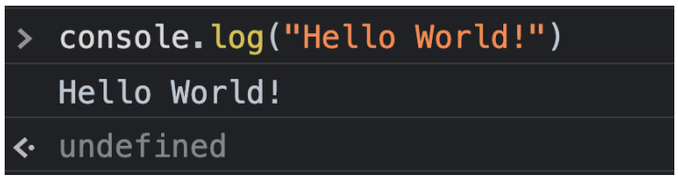
> **NOTE:** At any point of time, when you want to clear the console run the `clear()`, or `console.clear()` command.

To run the commands we will use the command prompt on the browser control. Type or paste each command and press **enter** to run the command.

1. Let's start with some simple code to print **Hello World!** to the console. Run the following command.

```
console.log("Hello World!")
```

The output should be the same as below.



The `undefined` value means that your command doesn't return any value.

2. Let's create some variables and print them. Run the following commands.

```
let num = 5
var mystr = "John"
console.log(num)
console.log(mystr)
```

Both `let` and `var` can be used to create variables. `var` is used when you want the variable to have global scope and `let` is used when you want the variable to have scope within the block where it is created.

3. Let's create a constant and print it. Run the following command.

```
const pi_val = 3.147
console.log(pi_val)
```

`const` is used to declare variables whose values can never change.

4. Let's create function which prints any value that is input to it.

```
function printMyInput(user_input) {
    console.log("The parameter passed is " + user_input)
}
```

5. Call the function you created in the previous step, once with a number and once with a string.

- *Note: `9` & `John` are the 2 input parameters passed to the **printMyInput** function.*

```
printMyInput(9)
printMyInput("John")
```

6. Let us see another approach of writing the **printMyInput** function according to the ES6 standard. This syntax is also called `arrow functions` and provide a shorthand to write functions.

```
let printMyInputES6 = (user_input) => {
    console.log(user_input)
}
```

7. Call the function you created in the previous step once with a number and once with a string.

```
printMyInputES6(9)
printMyInputES6("John")
```

Since the function is passed a single value and the body of the function is a single line, the brackets can be omitted. The code can also be written as below.

```
let printMyInputES6Short = user_input => console.log(user_input)
```

Now when we call it, the output should remain the same.

```
printMyInputES6Short(9)
printMyInputES6Short("John")
```

Task 3 - Operators, Conditions, Loops

In this task, you will be running some JavaScript from which you can learn how to use operators, controls, and loops.

Ensure that you understand the code in each file. These are primitive and foundational for your understanding of JavaScript.

1. **Arithmetic operators** are operators that we use to perform arithmetic operations.
   - + (plus) operator is used to add
   - - (minus) operator is used to subtract
   - * (star or asterisk) operator is used to multiply
   - / (slash) operator is used to divide
   - ** (double star) operator is used for exponentiation/power
   - % (percentage) operator is used for modulus operation (the remainder left over after division)

```
console.log("5 + 3 = ", 5 + 3)
console.log("7 - 3 = ", 7 - 3)
console.log("8 * 2 = ", 8 * 2)
console.log("27 / 3 = ", 27 / 3)
console.log("4 to the power of 3 = ", 4 ** 3)
console.log("19 mod 4 = ", 19 % 4)
```

The plus (+) operator is also used for string concatenation. When using a `+` with both a number and a string, they both get treated as a string, and get concatenated rather than added.
Expressions are read left to right, so adding two numbers and then a string will interpret the first + as addition, and the second + as concatenation.

```
console.log("5 + 3 = ", 5 + 3)
console.log("5 + \"3\" = ", 5 + "3")
console.log("5 + 5 + \"3\" = ", 5 + 5 + "3")
console.log("\"3\" + 5 + 5 = ", "3" + 5 + 5)
console.log("5 + 5 + \"3\" + 5 = ", 5 + 5 + "3" + 5)
```

2. **Assignment operators** are operators that are used to assign values to variables

   - = operator is used to assign value on the right to the variable on left
   - += operator is used to increment the value stored in the left operand by the value of the right operand and store it back to the left operand (the same as writing `tmp = tmp + val` where `tmp` is a variable and `val` is some arbitrary value)
   - -= operator is used to decrement the value stored in the left operand by the value of the right operand and store it back to the left operand (the same as writing `tmp = tmp - val` where `tmp` is a variable and `val` is some arbitrary value)
   - *= operator is used to multiply the value stored in the left operand by the value of the right operand and store it back to the left operand (the same as writing `tmp = tmp * val` where `tmp` is a variable and `val` is some arbitrary value)
   - /= operator is used to divide the value stored in the left operand by the value of the right operand and store it back to the left operand (the same as writing `tmp = tmp / val` where `tmp` is a variable and `val` is some arbitrary value)
   - **= operator is used to raise the value stored in the left operand to the power value of the right operand and store it back to the left operand (the same as writing `tmp = tmp ** val` where `tmp` is a variable and `val` is some arbitrary value)
   - %= operator is used to get modulus of the value stored in the left operand by value of the right operand and store it back to the left operand (the same as writing `tmp = tmp % val` where `tmp` is a variable and `val` is some arbitrary value)

```
x = 5
console.log("Old value of x: ", x)
x += 3
console.log("New value of x: ", x)
y = 5
console.log("Old value of y: ", y)
y -= 3
console.log("New value of y: ", y)
a = 6
console.log("Old value of a: ", a)
a *= 3
console.log("New value of a: ", a)
b = 6
console.log("Old value of b: ", b)
b /= 3
console.log("New value of b: ", b)
c = 6
console.log("Old value of c: ", c)
c %= 3
console.log("New value of c: ", c)
```

```
d = 6
console.log("Old value of d: ", d)
d **= 3
console.log("New value of d: ", d)
```

3. **Comparison Operators** are used to compare values or variables against other values or variables

- == operator checks if the operand on the left is of equal value to the operand on right
- === operator checks if the operand on the left is of equal value *and equal type* to the operand on right
- != operator checks if the operand on the left is not of equal value to the operand on right
- > operator checks if the operand on the left is greater than that on the right
- < operator checks if the operand on the left is lesser than that on the right
- >= operator checks if the operand on the left is greater than or equal to that on the right
- <= operator checks if the operand on the left is lesser than or equal to that on the right

```
//Checking equality of 5 (number type) and 5 (string type)
console.log("5=='5' ", 5=='5')
console.log("5==='5' ", 5==='5')
console.log("5===5 ", 5===5)
console.log("5 != 5 ", 5 !== 5)
console.log("5 != 6 ", 5 != 6)
console.log("5 != '5' ", 5 !== '5')
console.log("5 > 2 ", 5 > 2)
console.log("5 > 7 ", 5 > 7)
console.log("5 > 5 ", 5 > 5)
console.log("5 < 7 ", 5 < 7)
console.log("5 < 2 ", 5 < 2)
console.log("5 < 5 ", 5 < 5)
console.log("5 >= 5 ", 5 >= 5 )
console.log("5 <= 5 ", 5 <= 5 )
```

4. **Logical Operators** are used to combine more than one conditions.

- && operator checks if the condition on left and right are true. Returns true only of both conditions are true. Else returns false.
- || operators checked if either the condition on the left is true or right is true. Returns true even if one of the two conditions is true.
- ! operator checks if the condition is not met.

```
var raining = false
var cloudy = true
console.log("It is raining: ", raining)
console.log("It is cloudy: ", cloudy)
console.log("It is raining AND cloudy: ", raining && cloudy)
console.log("It is raining OR cloudy: ", raining || cloudy)
console.log("It is not raining: ", !raining)
console.log("It is not cloudy: ", !cloudy)
```

Short-Circuit Evaluation

Short-circuit evaluation is a concept in which the compiler will skip checking sub-expressions in a compound statement (a statement with logical operators) once the value is determined.

- exp1 && exp2 will not evaluate exp2 if exp1 is **false** because if even one expression is false with an &&, the entire expression is false
- exp1 || exp2 will not evaluate exp2 if exp1 is **true** because if even one expression is true with an ||, the entire expression is true

This can be very useful when evaluating certain expressions, and should be taken advantage of where needed.

```
var chocolate = true
var candy = false
console.log("There is chocolate: ", chocolate)
console.log("There is candy: ", candy)
console.log("There is candy AND chocolate: ", chocolate && candy, " -- Only candy is evaluated")
console.log("There is chocolate OR candy: ", chocolate || candy, " -- Only chocolate is evaluated")
```

5. **if - else if - else** Conditional statements are very useful to control the flow of your code.

```
//Accept a input from the user. If it is a number, print the multiplication table for the number. Else print the input and the length of the input.
let user_input = prompt('Enter a value');
//Check if the user did not input anything
if (!user_input) {
    console.log("You did not input anything")
}
//Check if the user input is not a number
else if (isNaN(user_input)) {
    console.log("Your input is ", user_input)
    console.log("The length of your input is ", user_input.length)
}
//The only option remaining is that the input is a number
else {
    console.log(user_input, " X 1 = ", user_input*1)
    console.log(user_input, " X 2 = ", user_input*2)
    console.log(user_input, " X 3 = ", user_input*3)
    console.log(user_input, " X 4 = ", user_input*4)
```

```
    console.log(user_input, " X 5 = ", user_input*5)
    console.log(user_input, " X 6 = ", user_input*6)
    console.log(user_input, " X 7 = ", user_input*7)
    console.log(user_input, " X 8 = ", user_input*8)
    console.log(user_input, " X 9 = ", user_input*9)
    console.log(user_input, " X 10 = ", user_input*10)
}
```

6. **switch-case** statements are used to replace multiple if - else if conditions which check the same variable. After one of the conditions is satisfied and the block of code is executed, the control should explicitly **break** out of the switch block. Otherwise, all other conditions will be executed until either a **break** statement is found, or until there is no more code.

```
let user_input = prompt('Enter a number between 1 to 7');
//Using logical OR operator to check if the input is a number and it is between 1 to 7
if(isNaN(user_input) || user_input< 1 || user_input>7) {
    console.log("Invalid input")
} else {
    user_input = parseInt(user_input)
    switch(user_input){
        case 1: console.log("Sunday"); break;
        case 2: console.log("Monday"); break;
        case 3: console.log("Tuesday"); break;
        case 4: console.log("Wednesday"); break;
        case 5: console.log("Thursday"); break;
        case 6: console.log("Friday"); break;
        case 7: console.log("Saturday"); break;
        default: console.log("Invalid entry");
    }
}
```

7. **Loops** can be used when the same block of code needs to be executed many times.

for loops have an initial value, condition based on which the loop is executed, and an incremental value.

```
//Accept a input from the user. If it is a number print the multiplication table for the number.
let user_input = prompt('Enter a number');
//Check if the user input is a number
if(!isNaN(user_input)) {
    //Using for loop for the repetitive statement
    for (let i=0; i<10; i++) {
        console.log(user_input, " X ", i, " = ", user_input*i)
    }
}
```

while loops have only expression: a condition based on which a block of code is executed. This is the same type of expression as the second one in a **for loop**

```
//The code below is to find the length of the words the user is entering. The loop will go on and on until the user chooses not to continue by pressi
let do_more = true
while(do_more) {
    //Accept a input from the user.
    let user_input = prompt('Enter a word');
    //Check if the user input is not a number and then print the length of the input
    if(isNaN(user_input)) {
        console.log("Length of the word you entered is " + user_input.length)
    } else {
        console.log("You entered a number. Only words are allowed")
    }
    let should_continue = prompt("Do you want to continue. Press n to stop")

    if(should_continue === "n") {
        do_more = false
    }
}
```

Task 4 - Collections

1. Array is an indexed collection. The index positions start from 0. The element in first position is at index 0, the second element is at position 1, and so on. The index of the last position will always be one less than the length of the array.

```
let myArray = ["Jack","Jill",4,5,true,"John"]
console.log(myArray[0]);
console.log(myArray[5]);
```

2. To iterate through arrays, there is a special type of for loop, **forEach**, which gets executed for each value in the given array.

```
let myArray = ["Jack","Jill",4,5,true,"John"]
myArray.forEach(element => {
    console.log(element)
})
```

3. To find the index position and the value, we can use the generic Object.entries method, which can be used with all collection objects. This maps each index position to the value.

```
let myArray = ["Jack","Jill",4,5,true,"John"]
for (const [index, value] of Object.entries(myArray)) {
  console.log(index, " - ", value);
}
```

4. **Map** object maps a key to a value. The keys have to be unique. The values can be string, int, float, or any other valid JavaScript datatype. An empty Map object can be create with the `new` keyword.

```
let myMap = new Map();
//Add a key-value pair to the map, with a key of "name" and a value of "John".
myMap.set("name", "John")
//Add another key-value pair to the map, with a key of "age" and a value of 22.
myMap.set("age", 22)
myMap.forEach((val,key) => {
    console.log(key, " - ", val)
})
```

Author(s)

Lavanya

Other Contributor(s)

Michelle Saltoun