Web Programming 2x1

JavaScript Numbers | Strings | Dates

Outcomes

Students should understand the following outcomes, upon successful completion of this module:

- Using Numbers
- Using String
- Using Date

Numbers

In modern JavaScript, there are two types of numbers:

1) Regular numbers

- Regular numbers" are represented by the number data type, encompassing both integers and floating-point numbers.
- 64-bit format IEEE-754, also known as “double precision floating point numbers”.
- These are numbers that used most of the time, and topic of discussion in this section.

```
let integerNumber = 42;          // Integer
let floatingPointNumber = 3.14 // Floating-point number
```

Numbers

2) BigInt numbers

- BigInt is a relatively recent addition (introduced in ECMAScript 2020).
- They allows us to work with integers of arbitrary (as large as needed) precision, beyond the limitations of the traditional number type which is 64-bit.
- They are sometimes needed, because a regular number can't exceed 253 or be less than -253.
- BigInts are declared by appending *n* to the end of an integer literal or by calling the *BigInt()* function:

```
1   const bigNumber = 1234567890123456789012345678890n;
2   const bigNumber2 = BigInt("90071992547409991"); // Using BigInt function
3
4   console.log(bigNumber);
5   console.log(bigNumber2);
6
7   console.log(bigNumber + bigNumber2);    //Arithmentic operations
```

Numbers

3) Scientific Notation (e notation)

- It is used to represent floating-point numbers, not integers or BigInts.

- Scientific notation uses e or E to indicate powers of 10, typically used with floating-point numbers to represent very large or very small numbers compactly.

- For example, 1.23e6 represents 1.23 multiplied by 10^6, which is 1230000.

```
let smallNumber = 1.23e-4; // 0.000123 (1.23 multiplied by 10^-4)
let largeNumber = 6.02e23; // 6.02 multiplied by 10^23

console.log(smallNumber);
console.log(largeNumber);
```

Methods of Number

The Number object provides several built-in methods that allow you to perform various operations and manipulations on numbers.

a) Conversion Methods - Number.parseInt() and Number.parseFloat():

These methods are used to parse strings and convert them into integers (parseInt) or floating-point numbers (parseFloat):

```
// Converts string "42" to integer 42
let intNum = Number.parseInt("42");

// Converts string "3.14" to float 3.14
let floatNum = Number.parseFloat("3.14");


console.log(intNum);
console.log(floatNum);
```

Methods of Number

b) Number Validation Methods

Number.isNaN() and Number.isFinite():

```
console.log(Number.isNaN(123));   //false
console.log(Number.isNaN("hello")); //false (not a number)
console.log(Number.isNaN(NaN));   //true
```

c) Conversion and Formatting Methods

Number.toFixed(), Number.toPrecision() & Number.toString():

```
let num = 3.14159;
console.log(num.toFixed(2));   // Output: "3.14"
console.log(num.toPrecision(4));   // Output: "3.142"
console.log(num.toString());   // Output: "3.14159"
```

Rounding Numbers

- Rounding numbers in JavaScript can be achieved using several methods provided by the Math object or directly through the Number object's methods.

```
let num = 7.12345;
console.log(Math.round(num)); // Output: 7

let num2 = 10.9875;
console.log(Math.floor(num2)); // Output: 10

let number = 3.001;
console.log(Math.ceil(number));  // Output: 4
```

- Math.round() is often used when you want to round to the nearest integer.
- Math.floor() and Math.ceil() are useful when you specifically need to round down or round up, respectively.

Math Object

- The Math object in JavaScript provides a set of built-in mathematical functions and constants that allow you to perform complex mathematical operations.
- These functions are static, meaning you call them directly on the *Math* object itself, rather than on instances of the object.

i. Math.PI: *Represents the ratio of the circumference of a circle to its diameter, approximately 3.14159.*

i. Math.random() *generates a pseudo-random number between 0 (inclusive) and 1 (exclusive).*

ii. Math.sqrt() *for square root.*

iii. Math.min() & Math.max() for min and max numbers.

```
console.log(Math.PI);
console.log(Math.random());
console.log(Math.sqrt(36));
console.log(Math.min(4, 9));
console.log(Math.max(2, 30));
```

Exercise

Q1: The formula for the surface area A of a cylinder with radius cylinder with radius r and height h is given by:

$$A = 2\pi r^2 + 2\pi rh$$

Using a Math object, create a JavaScript app to calculate the surface area when r = 24.67 and h = 12.13. Area must be correct to 4 sig figures.

Q2: Consider the following expressio

```
let num = Math.floor(Math.random() * (max - min + 1)) + min;
```

What will console.log(num) output if Math.random() generates 0.458, min = 10 and max = 100?

Strings

- In JavaScript, the textual data is stored as strings. There is no separate type for a single character.
- Strings can be enclosed within either single quotes, double quotes or backticks:
  - let single = 'single-quoted';
  - let double = "double-quoted";
  - let backticks = `backticks`;
- Backticks allow us to embed any expression into the string, by wrapping it in *${…}*.
- The following JavaScript statement that demonstrates *string interpolation* within a *template literal*:

```
let num1 = 1, num2 = 2;

console.log(`1 + 2 = ${num1 + num2}.`);
```

String Interpolation vs Concatenation

- *String expression interpolation* in JavaScript refers to the ability to embed expressions directly within strings using *template literals*: *`${...}`*.

- This feature allows you to easily concatenate variables, function calls, or even complex expressions into a string without resorting to traditional *concatenation* operator (+).

```
let fname = 'John';
let age = 21;

// Using template literals for string interpolation
let message = `Hello, my name is ${fname} and I am ${age} years old.`;
console.log(message);


// Using concatenation with +
let message2 = 'Hello, my name is ' + fname + ' and I am ' + age + ' years old.';
console.log(message2);
```

Manipulating Strings

- JavaScript strings offer various methods and properties for manipulation and retrieval of data.
- Modern JavaScript (ES6+) introduced template literals for cleaner multiline strings and enhanced readability.

1) Multiline Strings

- We can use Escape characters (\n) for multiline strings, but with ES6, template literals provide a cleaner way to handle multiline strings:

```
2   let guestList = "Guests:\n * John\n * Pete\n * Mary";
3   console.log(guestList);
4
5
6   let multilineString = `
7       Guests:
8       1 - Andrew
9       2 - Deen
10      3 - Joseph
11  `;
12
13  console.log(multilineString);
```

Manipulating Strings

- Create and define the output of each the following:

```
let str = "Belgium Campus!";

console.log(str.length);
console.log(str[0]);
console.log(str[str.length - 1]);
console.log(str.toLowerCase());
console.log(str.toUpperCase());
console.log(str.substring(1, 5));
console.log(str.indexOf("Campus"));
console.log(str.slice(8));
console.log(str.slice(0, 8));
```

- The *length* property of a string returns the number of characters in the string.
- You can access characters in a string using *bracket* notation (*[]*). Indices start at 0 for the first character.
- Return a new string with all characters converted to *lowercase* or *uppercase*.
- The *substring()* method extracts characters from a string between two indices (start inclusive, end exclusive).
- The *indexOf()* method returns the index of the first occurrence of a specified substring within the string.
- The *slice()* method extracts a section of a string and returns it as a new string, without modifying the original string.

Immutable Strings

- Strings in JavaScript are immutable sequences of characters.
- Immutable means that once a string is created, its content cannot be changed.
  - Any operation that appears to modify a string actually creates and returns a new string with the modified content, leaving the original string unchanged:

```
let str = "WPR2X1";
str[0] = 'Web Programming';

console.log(str);   // Output: "WPR2X1"
```

  - Operations like concatenation, substring extraction, or methods like toUpperCase() and toLowerCase() do not alter the original string:

```
let originalStr = "Belgium Campus";
let newStr = originalStr.toUpperCase();

console.log(originalStr);
console.log(newStr);
```

Date Object

- The Date object is used for working with dates and times.
- It provides methods for creating, manipulating, and formatting dates and times.
- Create a new Date object:

```
let currentDate = new Date();
console.log(currentDate);

let date1 = new Date("2020-10-07");
console.log(date1);

let date2 = new Date(2024, 7, 22); //Month is 7 for August (0-indexed)
console.log(date2);

let dateTime = new Date(2024, 3, 19, 12, 30, 0, 0); //July 18, 2024, 12:30:00
console.log(dateTime);
```

Date Methods and Properties

- The Date object provides various methods to work with dates and times:

```
let now = new Date();

console.log(now.getFullYear());
console.log(now.getMonth());
console.log(now.getDate());
console.log(now.getHours());
console.log(now.getMinutes());
console.log(now.getSeconds());
console.log(now.getMilliseconds());
console.log(now.getDay());
```

Thank You!

The End

info@belgiumcampus.ac.za +27 10 593 53 68

/belgiumcampusSA

#Belgium Campus

/belgiumcampus