OBJECT™
TECHNOLOGIES

Java script built in objects

What will be covered

OBJECT TECHNOLOGIES

- Introduction to Date
- Creating Date
- Date methods
- Introduction to string
- String methods
- Global functions
  - parseInt, parseFloat
  - Number, String
  - isNaN, isFinite
  - encodeURL,decodeURL
  - Eval
- Form validation
  - Built-in form validation
  - JavaScript validation

Introduction to Date

OBJECT TECHNOLOGIES

- JavaScript Date objects represent a single moment in time in a platform-independent format. Date objects contain a Number that represents milliseconds since 1 January 1970 UTC.
- We can use it to store creation/modification times, to measure time, or just to print out the current date.

Creating Date

OBJECT TECHNOLOGIES

To create a new Date object call new Date() with one of the following arguments:

1. new Date()

   Without arguments – create a Date object for the current date and time:

```
let now = new Date();
alert( now ); // shows current date/time
```

2. new Date(milliseconds)

   Create a Date object with the time equal to number of milliseconds (1/1000 of a second) passed after the Jan 1st of 1970 UTC+0.

```
// 0 means 01.01.1970 UTC+0
let Jan01_1970 = new Date(0);
alert( Jan01_1970 );
// now add 24 hours, get 02.01.1970 UTC+0
let Jan02_1970 = new Date(24 * 3600 * 1000);
alert( Jan02_1970 );
```

3. new Date(datestring)

   If there is a single argument, and it's a string, then it is parsed automatically. The algorithm is the same as Date.parse uses, we'll cover it later.

```
let date = new Date("2017-01-26");
alert(date);
```

Creating Date

OBJECT

4. new Date(year, month, date, hours, minutes, seconds, ms)

Create the date with the given components in the local time zone. Only the first two arguments are obligatory.

- The year must have 4 digits: 2013 is okay, 98 is not.
- The month count starts with 0 (Jan), up to 11 (Dec).
- The date parameter is actually the day of month, if absent then 1 is assumed.
- If hours/minutes/seconds/ms is absent, they are assumed to be equal 0.

```
new Date(2011, 0, 1, 0, 0, 0, 0); // 1 Jan 2011, 00:00:00
let date = new Date(2011, 0, 1, 2, 3, 4, 567);
```

5. Using Date.now()

Returns the numeric value corresponding to the current time—the number of milliseconds elapsed since January 1, 1970 00:00:00 UTC, with leap seconds ignored.

```
let now = new Date()
// Sat Jan 09 2021 22:06:33 GMT+0100
```

Date Methods

- Access Date Component

  There are methods to access the year, month and so on from the Date object:

  getDate() – returns date value between 1- 31

  getDay() – returns weekday value between 0(Sunday) – 6

  getMonth() – returns month value between 0(January) – 11

  getFullYear() – returns 4 digit year

  getHours() – return hours between 0- 23

  getMinutes() – return minutes between 0 - 59

  getSeconds() – return seconds between 0 – 59

  geTime() – returns no of miliseconds from 1 Jan 1970

Date Methods

OBJECT TECHNOLOGIES

- Setting Date Component

  The following methods allow to set date/time components:

  setDate() - Set the day as a number (1-31)

  setFullYear() - Set the year (optionally month and day)

  setHours() - Set the hour (0-23)

  setMilliseconds() - Set the milliseconds (0-999)

  setMinutes() - Set the minutes (0-59)

  setMonth() - Set the month (0-11)

  setSeconds() - Set the seconds (0-59)

  setTime() - Set the time (milliseconds since January 1, 1970)

Date Methods

OBJECT TECHNOLOGIES

- **Parsing date**

  The call to Date.parse(str) parses the string in the given format and returns the timestamp (number of milliseconds from 1 Jan 1970 UTC+0). If the format is invalid, returns NaN.

  The string format should be: YYYY-MM-DDTHH:mm:ss.sssZ

- **Compare dates** – We can use comparison operators like < and > two compare two Date objects, and under the hood, their tin.e counters are effectively compared.

- Note: Equality operators (== and ===) don't work with Date objects, so we don't explicitly check if they're the same.

- Another way to compare two dates is by using the built-in getTime() method.

-

Introduction to string

OBJECT TECHNOLOGIES

- JavaScript strings are primitive values. JavaScript strings are also immutable. It means that if you process a string, you will always get a new string. The original string doesn't change.
- To create literal strings in JavaScript, you use either single quotes or double quotes like this:

```
let str = 'Hi';
let greeting = "Hello";
```

- ES6 introduced template literals that allow you to define a string backtick (`) characters:

```
let name = 'John';
let message = `Hello ${name}`;
console.log(message);
```

Output:

```
Hello John
```

String methods

OBJECT TECHNOLOGIES

- Finding a String in a String : indexOf(), lastIndexOf()
- Searching for a String in a String : search()
- Extracting String Parts : slice(), substring(), substr()
- Replacing String Content : replace()
- Converting to Upper and Lower Case : toLowerCase(), toUpperCase()
- Joining strings : concat()
- Extracting String Characters : charAt(), charCodeAt()
- Converting a String to an Array : split()

Global functions

- parseInt() - Parses a string and returns an integer
- parseFloat() - Parses a string and returns a floating point number
- String() - Converts an object's value to a string
- Number() - Converts an object's value to a number
- decodeURI() - Decodes a URI
- decodeURIComponent() - Decodes a URI component
- encodeURI() - Encodes a URI
- encodeURIComponent() - Encodes a URI component
- isFinite() - Determines whether a value is a finite, legal number
- isNaN() - Determines whether a value is an illegal number
- eval() - Evaluates a string and executes it as if it was script code

Form validation

OBJECT

- Before submitting data to the server, it is important to ensure all required form controls are filled out, in the correct format.
- This is called client-side form validation, and helps ensure data submitted matches the requirements set forth in the various form controls.
- Client-side validation is an initial check and an important feature of good user experience; by catching invalid data on the client-side, the user can fix it straight away. If it gets to the server and is then rejected, a noticeable delay is caused by a round trip to the server and then back to the client-side to tell the user to fix their data.
- There are two different types of client-side validation
- **Built-in form validation** uses HTML5 form validation features, which we've discussed in many places throughout this module. This validation generally doesn't require much JavaScript. Built-in form validation has better performance than JavaScript, but it is not as customizable as JavaScript validation.
- **JavaScript** validation is coded using JavaScript. This validation is completely customizable, but you need to create it all (or use a library).

Built-in form validation

OBJECT TECHNOLOGIES

- The simplest HTML5 validation feature is the required attribute to make an input mandatory. The form won't submit, displaying an error message on submission when the input is empty.
- Another useful validation feature is the pattern attribute, which expects a Regular Expression as its value. A regular expression (regex) is a pattern that can be used to match character combinations in text strings.
- Character length of all text fields can be constrainted on <input> or <textarea> by using the minlength and maxlength attributes.
- For number fields (i.e. <input type="number">), the min and max attributes can be used to provide a range of valid values.

JavaScript validation

OBJECT TECHNOLOGIES

You must use JavaScript if you want to take control over the look and feel of native error messages.

The Constraint Validation API

- Most browsers support the Constraint Validation API, which consists of a set of methods and properties available on the form elements like button, input, select, textarea.
- This API supports validationMessage, validity, willValidate properties which can be used for checking the current status of the form element.
- checkValidity(), reportValidity(), setCustomValidity(message) are some methods which can be used on the form elements.

Validating forms without a built-in API

- In some cases, you won't want to use the Constraint Validation API, you're still able to use JavaScript to validate your form, but you'll just have to write your own code.

JavaScript validation

OBJECT TECHNOLOGIES

- To validate a form, need to determine how to validate the data i.e. string operations ore regular expressions. If the form doesn't validate, whether to display the error message or highlight the field.
- It's very important to provide as much helpful information as possible to the user in order to guide them in correcting their inputs.

JS Built-in Objects

In this chapter, we will throw a light on very frequently needed object types and their functionality like array, date and string. We will even learn about few global functions available in javascript.

Date

JavaScript Date object is a built-in object which is used to deal with date and time. It stores a Number which represents milliseconds for the Unix Timestamp(which is nothing but milliseconds passed since 1 January 1970 UTC). We can use date to achieve following functionality

- to create a new date
- format a date
- get elapsed time between two different time values, etc.
- get a date value using milliseconds number
- get a date based on the different timezones as well.

Creating date object

1. Using no-arg constructor
   the default way is using the new keyword with the Date constructor function.

   ```
   let newdate = new Date();
   ```

   This will create a new Date object with the current date and time stored in it. The format of the date stored in this object will be:
   Tue May 05 2020 21:16:17 GMT+0530 (India Standard Time)
2. Getting unix timestamp
   To get the standard UNIX timestamp value we can use the following code:

   ```
   let newdate = Date.now();
   ```

   1588693792007
   `now() is a static function` for the Date object. This will give us the Unix timestamp for current date and time which is nothing but the number of milliseconds passed since 1 January 1970.
3. Using constructor with int argument
   Using integer as an argument - which represents milliseconds - and sets the date object to the given number of milliseconds after 1/1/1970. For example, new date(7000) would set the object to 7 seconds after 1/1/1970

3. Using constructor with int argument

   Using integer as an argument - which represents milliseconds - and sets the date object to the given number of milliseconds after 1/1/1970. For example, new date(7000) would set the object to 7 seconds after 1/1/1970.

```
const d = new Date(24 * 60 * 60 * 1000);
```

   This creates date as January 01 1970 plus 24 hours

4. Using constructor with string argument

   Takes a string representation of a date, in a format accepted by the date.parse() method, as an argument.

```
const d = new Date("October 13, 2014 11:13:00");
```

   Creates a date as Mon Oct 13 2014 11:13:00 GMT+0530 (India Standard Time)

```
const d = new Date("2022-03-25");
```

   Creates a date as Fri Mar 25 2022 05:30:00 GMT+0530 (India Standard Time)

5. Using contructor with date parts

   Takes arguments in the order like year, month, day, hours, minutes, seconds, each specifying a part of the date. Out of this only year and month is compulsory and rest arguments are optional.

```
const d = new Date(2018, 11, 24, 10, 33, 30, 0);
```

   Creates a date as Mon Dec 24 2018 10:33:30 GMT+0530 (India Standard Time)

Date functionality

Once a date object is instantiated, there are several methods that can be used with it. Some are get methods - which enable you to retrieve specific information from a date, while others are set methods, which let you edit the information in a date object.

Getter methods

getDay()	Returns an integer which represents the day (0-6)- 0 means Monday, 1 means Tuesday and so on.
getDate()	Returns an integer which represents the date
getMonth()	Returns an integer which represents the month (0-11) - 0 means January onwards
getFullYear()	Returns a 4 digit integer which represents the year
getHours()	Returns an integer that represents the hours in the date object.(0-23)

Date functionality

Once a date object is instantiated, there are several methods that can be used with it. Some are get methods - which enable you to retrieve specific information from a date, while others are set methods, which let you edit the information in a date object.

Getter methods

getDay()	Returns an integer which represents the day (0-6)- 0 means Monday, 1 means Tuesday and so on.
getDate()	Returns an integer which represents the date
getMonth()	Returns an integer which represents the month (0-11) - 0 means January onwards
getFullYear()	Returns a 4 digit integer which represents the year
getHours()	Returns an integer that represents the hours in the date object.(0-23)
getMinutes()	Returns an integer representing minutes in the date object.(0-59)
getSeconds()	Returns an integer representing seconds in the date object.(0-59)
getMilliseconds()	Returns an integer representing seconds in the date object.(0-999)
getTime()	Get time (milliseconds since January 1, 1970)

Setter methods

setDate()	Set the day as a number (1-31)
setFullYear()	Set the year (optionally month and day)
setHours()	Set the hour (0-23)
setMilliseconds()	Set the milliseconds (0-999)
setMinutes()	Set the minutes (0-59)
setMonth()	Set the month (0-11)
setSeconds()	Set the seconds (0-59)
setTime()	Set the time (milliseconds since January 1, 1970)

Date formats

There are generally 3 types of JavaScript date input formats:

1. ISO Dates

   ISO dates can be written with added hours, minutes, and seconds (YYYY-MM-DDTHH:MM:SSZ):

   ```
   const d = new Date("2015-03-25T12:00:00Z");
   ```

Date formats

There are generally 3 types of JavaScript date input formats:

1. ISO Dates

   ISO dates can be written with added hours, minutes, and seconds (YYYY-MM-DDTHH:MM:SSZ):

   const d = new Date("2015-03-25T12:00:00Z");

   Date and time is separated with a capital T. UTC time is defined with a capital letter Z.

2. JavaScript Short Dates.

   Short dates are written with an "MM/DD/YYYY" syntax lie this:

   const d = new Date("03/25/2015");

3. JavaScript Long Dates.

   Long dates are most often written with a "MMM DD YYYY" syntax like this:

   const d = new Date("Mar 25 2015");

   Month and day can be in any order and month can be written in full (January), or abbreviated (Jan):

Time zones

When setting a date, without specifying the time zone, JavaScript will use the browser's time zone. When getting a date, without specifying the time zone, the result is converted to the browser's time zone.

String

In JavaScript, a string is a data type representing a sequence of characters that may consist of letters, numbers, symbols, words, or sentences. Strings are used to represent text. So, basically, anything that is a Unicode character.

Creating strings in JavaScript

In JavaScript, you can create strings by wrapping the text inside single quotes ('), double quotes ("), or backticks (`).

```
//strings example
const name = 'Peter';
const name1 = "Jack";
const result = `The names are ${name} and ${name1}`;
```

Points to note :

- A string created using single quotes, double quotes, or backticks is generated as a primitive value, similar to numbers and

Creating strings in JavaScript

In JavaScript, you can create strings by wrapping the text inside single quotes ('), double quotes ("), or backticks (`).

```
//strings example
const name = 'Peter';
const name1 = "Jack";
const result = `The names are ${name} and ${name1}`;
```

Points to note :

- A string created using single quotes, double quotes, or backticks is generated as a primitive value, similar to numbers and boolean values. Primitive data are immutable, which means they cannot be changed. Also, they do not have any methods or properties.
- Backticks are generally used when you need to include variables or expressions into a string.
- You can also write a quote inside another quote. For example,

```
const name = 'My name is "Peter".';
```

- The quote should not match the surrounding quotes. For example,

```
const name = 'My name is 'Peter'.'; // error
```

String functionality

Method	Description	Example
charAt()	Returns character at a specified index in string	const string = "Hello World!"; let index1 = string.charAt(1); //e
charCodeAt()	Returns Unicode of the character at given index	const greeting = "Good morning!"; let result = greeting .charCodeAt(5); //109

toUppeCase()	Converts all characters of a string in capitals letter	const message = "javascript is fun";
		const upperMessage = message.toUpperCase();
		// Output: JAVASCRIPT IS FUN

toLowerCase()	Converts all characters of a string in lower case letter	const message = "JAVASCRIPT IS FUN";
		const lowerMessage = message.toLowerCase();
		// Output: javascript is fun

indexOf()	Returns the first index of	const message = "JavaScript is not Java";

String functionality

Method	Description	Example
charAt()	Returns character at a specified index in string	const string = "Hello World!"; let index1 = string.charAt(1); //e
charCodeAt()	Returns Unicode of the character at given index	const greeting = "Good morning!"; let result = greeting .charCodeAt(5); //109

toUppeCase()	Converts all characters of a string in capitals letter	const message = "javascript is fun";
		const upperMessage = message.toUpperCase();
		// Output: JAVASCRIPT IS FUN

toLowerCase()	Converts all characters of a string in lower case letter	const message = "JAVASCRIPT IS FUN";
		const lowerMessage = message.toLowerCase();
		// Output: javascript is fun

indexOf()	Returns the first index of occurrence of a value	const message = "JavaScript is not Java";
		// returns index of 'v' in first occurrence of 'va'
		const index = message.indexOf("va"); //2

lastIndexOf()	Returns the last index of occurrence of a value	// defining a string
		var str = "Programming";
		var substr = "g";
		// find last occurrence of "g" in str
		var result = str.lastIndexOf(substr); //10

slice()	Extracts and returns a section of the string between the given indices. It can accept even negative values	const message = "JavaScript is fun.";
		// get the substring starting from index 0 to 10(excluded)
		let result = message.substring(0, 10); //JavaScript

substring()	Returns a specified part of the string	const message = "JavaScript is fun.";
		// get the substring starting from index 0 to 10(excluded)
		let result = message.substring(0, 10); //JavaScript

substr()	Returns a specified no of characters starting with a given index	let str = "Apple, Banana, Kiwi"; let s_str = str.substr(7,6); //Banana
concat()	Appends the string at the end of	let emptyString = "";

slice()	Extracts and returns a section of the string between the given indices. It can accept even negative values	const message = "JavaScript is fun.";
		// get the substring starting from index 0 to 10(excluded)
		let result = message.substring(0, 10); //JavaScript

substring()	Returns a specified part of the string	const message = "JavaScript is fun.";
		// get the substring starting from index 0 to 10(excluded)
		let result = message.substring(0, 10); //JavaScript

substr()	Returns a specified no of characters starting with a given index	let str = "Apple, Banana, Kiwi"; let s_str = str.substr(7,6); //Banana

concat()	Appends the string at the end of the given string	let emptyString = "";
		// joint arguments string
		let joinedString = emptyString.concat("JavaScript", " is", " fun.");
		console.log(joinedString);
		// Output: JavaScript is fun.

split()	Returns the string divided into list of substring	const message = "JavaScript::is::fun"; let result = message.split("::"); //['JavaScript','is','fun']

serach()	Searches for specified value in the string	let sentence= "I love JavaScript.";
		// pattern that searches the first occurence of an uppercase character
		let regExp = /[A-Z]/;
		// searching for a match between regExp and given string
		let indexReg = sentence.search(regExp); //0

replace()	Replaces the searched string with the given string	const message = "ball bat";
		// replace the first b with c
		let result = message.replace('b', 'c');
		console.log(result);
		// Output: call bat

JS Global Functions

The JavaScript global properties and methods can be used with all

JS Global Functions

The JavaScript global properties and methods can be used with all JavaScript objects. Since these methods are global, and global the object is the browser window, these methods are actually window methods:
isNaN() is the same as window.isNaN().

1. isNaN

   In JavaScript NaN is short for "Not-a-Number". The isNaN() method returns true if a value is NaN. The isNaN() method converts the value to a number before testing it.

```
document.write("<br/>" + isNaN(19));  //false
document.write("<br/>" + isNaN(0));  //false
document.write("<br/>" + isNaN(-9.6));   //false
document.write("<br/>" + isNaN("he98hello"));   //true
document.write("<br/>" + isNaN(9 - 8));   //false
document.write("<br/>" + isNaN("24/5/2019"));    //true
```

2. isFinite

   The isFinite() method returns true if a value is a finite number. Infinite (not finite) values are Infinity, -Infinity, or NaN. Note that isFinite method will always return opposite value as that of isNaN for the same argument.

3. parseInt

   The parseInt method parses a value as a string and returns the first integer. Method trims the spaces from both the sides. Whenever a non numeric character is found, it stops further parsing. A radix parameter specifies the number system to use: 2 = binary, 8 = octal, 10 = decimal, 16 = hexadecimal. If radix is omitted, JavaScript assumes radix 10

```
document.write("<br/>" + parseInt("  99  "));   //99
document.write("<br/>" + parseInt("  99 99 888 "));   //99
document.write("<br/>" + parseInt("  99.99 "));      //99
document.write("<br/>" + parseInt("  99hello  "));   //99
document.write("<br/>" + parseInt("hello99"));    //NaN
```

4. parseFloat

   The parseFloat() method parses a value as a string and returns the first number. The only difference from parseInt method is dot dymbol(decimal point) is considered valid character for parsing

5. eval

   The eval() method evaluates or executes an argument. If the argument is an expression, eval() evaluates the expression. If

5. eval

   The eval() method evaluates or executes an argument. If the argument is an expression, eval() evaluates the expression. If the argument is one or more JavaScript statements, eval() executes the statements.

```
document.write("<br/>" + eval("1*3"));    //4
document.write("<br/>" + eval("a=10;b=20;a+b"));    //30
```

6. encodeURI

   The encodeURI() method encodes a URI. The encodeURI() method does not encode characters like:

   , / ? : @ & = + $ * #

```
var str = "http://localhost:80/login.php?pwd=bakul%123";
var en_str = encodeURI(str);
document.write("<br/> Encoded : " + en_str);
```

7. decodeURI

   The decodeURI() method decodes a URI.

```
var de_str = decodeURI(en_str);
document.write("<br/> Decoded : " + de_str);
```

Assignments

1. Design an HTML form to accept a string from the user and perform the following functions:
   1. Calculate the length of the string.
   2. Convert the string to upper case, lower case.
   3. Check if letter ?w? is present in the given string. If yes at which index. Use search from beginning as well as from the end.
   4. Check which alphabet is present at index 4, also get its ASCII code.
   5. Accept two strings from the form and perform their concatenation.
   6. Display all the words in the given string by splitting the words separated by spaces.
2. Accept the date of birth of the user using a form and perform the following operations:
   1. Find the day, month and year separately
   2. Find the age of user.
3. Display digital clock on the web page. It should refresh automatically after every second.
4. Display a stop watch on the web page. It should have buttons like start, pause and reset. When page gets loaded it should

Assignments

1. Design an HTML form to accept a string from the user and perform the following functions:
    1. Calculate the length of the string.
    2. Convert the string to upper case, lower case.
    3. Check if letter ?w? is present in the given string. If yes at which index. Use search from beginning as well as from the end.
    4. Check which alphabet is present at index 4, also get its ASCII code.
    5. Accept two strings from the form and perform their concatenation.
    6. Display all the words in the given string by splitting the words separated by spaces.
2. Accept the date of birth of the user using a form and perform the following operations:
    1. Find the day, month and year separately
    2. Find the age of user.
3. Display digital clock on the web page. It should refresh automatically after every second.
4. Display a stop watch on the web page. It should have buttons like start, pause and reset. When page gets loaded it should display 00:00:00 . Start button should start counting the time. Reset should start from 00 again.
5. Create the slide show of images after the interval of 1000 ms (User event is not required).