Objects and DOM Manipulation

Introduction to Objects and the DOM

An **object** in JavaScript is a standalone entity composed of properties and methods. Properties are key-value pairs that hold data, while methods are functions associated with the object.

💡 Objects allow you to group related data and functionality, making them an essential tool for building dynamic and reusable components in JavaScript.

For instance, an object representing a person might have properties like `name` and `age`, and methods like `greet()` to display a message. Objects are created using object literals, constructors, or classes.

1. Constructor Functions

- Constructor functions are used to create objects with shared properties and methods.
- **Definition**:

```
function ConstructorName(param1, param2) {
    this.property1 = param1;
    this.property2 = param2;
}
```

- **Example**:

```
const person = {
  firstName: "John",
  lastName: "Doe",
  age: 30,
  isEmployed: true,
  greet: function() {
    return `Hello, my name is ${this.firstName} ${this.las
tName}.`;
  }
};

console.log(person.greet());
```

---

2. Arrays and Array Methods

- Arrays store multiple values in a single variable.
- **Common Methods**:
  - `forEach`: Loops through each item in the array.

    ```
    array.forEach((item) => {
        // Process each item
    });

    guests.forEach((guest) => {
          console.log(`Welcome ${guest.name} to the Grand H
    otel!`);
    });
    ```

  - `some`: Checks if at least one item meets a condition. Returns true or false.

    ```
    array.some((item) => condition);

    const validateGuest = guests.some((guest) => guest.name
    == 'Andrea')
    console.log(validateGuest) // output: false

    const validateGuest = guests.some((guest) => guest.name
    ```

```
== 'Bob')
console.log(validateGuest) // output: true
```

- **Example**:

```
const guests = [
    new Guest('Alice', 101, 2),
    new Guest('Bob', 102, 4),
];

guests.forEach((guest) => {
      console.log(`Welcome ${guest.name} to the Grand Hote
l!`);
});
```

---

3. DOM Manipulation

The **DOM (Document Object Model)** is a programming interface that represents the structure of an HTML document as a tree of objects. Each element in the HTML is a node in this tree, and the DOM allows JavaScript to access and manipulate these nodes dynamically. This enables developers to modify content, styles, and structure in response to user interactions or other events, creating interactive web pages.

```
Document
└── html
├── head
|    ├── title
|    |    └── "Sample DOM Tree"
|
└── body
├── header
```

```
|     └── h1
|          └── "Welcome to My Website"
|
├── main
|     ├── section
|     |     ├── h2
|     |     |     └── "About Us"
|     |     └── p
|     |           └── "This is a sample paragraph about the website
|     |
|     └── section
|           ├── h2
|           |     └── "Contact"
|           └── ul
|                 ├── li
|                 |     └── "Email: contact@website.com"
|                 └── li
|                       └── "Phone: +123456789"
|
└── footer
└── p
└── "© 2025 My Website"
```

- **Selecting Elements**:

```
const element = document.getElementById('elementId');
```

- **Modifying Content**:

```
<p> Whatever inside this is text. </p>

element.innerHTML = '<u> New Content </u>';
```

```
<p> New Content </p> // Output: New Content
```

- **Creating Elements**:

```
const newElement = document.createElement('div');
document.createElement('ul'); // Output: <ul> </ul>
document.createElement('p'); // Output: <p></p>

const listElement = document.createElement('ol'); // Outpu
t: <ol> </ol>
const listItemElement = document.createElement('li'); // O
utput: <li></li>
```

- **Appending Elements**:

```

Before appendChild:
<ol></ol>
<li></li>
<li></li>



Syntax - parentElement.appendChild(newElement);

listElement.appendChild(listItemElement);

<ol>
   <li></li>
   <li></li>
</ol>
```

4. Event Listeners

- Respond to user interactions (e.g., clicks, form submissions).

- **Syntax**:

```
element.addEventListener('event', function (event) {
    // Handle the event
});
```

- **Example**:

```
document.getElementById('form').addEventListener('submit',
function (event) {
    event.preventDefault(); // Prevent default action
    // Custom logic
});
```

---

5. Template Literals

- Template literals simplify string concatenation and allow embedded expressions.
- **Syntax**:

```
`String with embedded ${expression}`
```

- **Example**:

```
const message = `Hello, ${name}! You are in room ${room}.
`;
```

---

6. Type Conversion

- Convert data between types as needed.
- **Common Methods**:
    - `parseInt`: Converts strings to integers.

```
const num = parseInt('123');

"10" + 1  = 101
"Andrea" + "Mendoza" = "AndreaMendoza"

parseInt("10") + 1 = 11
```

  - `Number`: Converts values to numbers.

```
const num = Number('123.45');
```

---

7. Preventing Duplicates or Conflicts

- Use `Array.some()` to check for conflicts in an array.

```
const isDuplicate = array.some((item) => item.property ===
value);

if (isDuplicate) {
    console.log('Duplicate detected!');
}
```

---

8. Form Handling

- Capture user input and validate it.
- **Access Input Values**:

```
const value = document.getElementById('inputId').value;
```

- **Validate Input**:

```
if (value === '') {
    console.log('Input cannot be empty!');
}
```

9. Reusable Functions

- Create modular functions for repetitive tasks.
- **Example**:

```
function renderItems(array) {
    array.forEach((item) => {
        console.log(item);
    });
}
renderItems(guests);
```

10. Error Handling

- Display errors to users or log them for debugging.
- **Example**:

```
const errorElement = document.getElementById('error');
errorElement.style.display = 'block';
errorElement.textContent = 'An error occurred!';

// or...

errorElement.classList.remove('d-none'); // Makes previous
ly invisible element visible now if class d-none was used
in HTML.
```

11. Dynamic Styling

- Use classes to dynamically style elements.
- **Example**:

```
element.className = 'new-class';

<div class="container">

element.className = 'container';

// or...

element.classList.add('new-class');

element.classList.add('mb-5');

<div class="container mb-5">
```