Lesson:

Introduction to Object

PW SKILLS

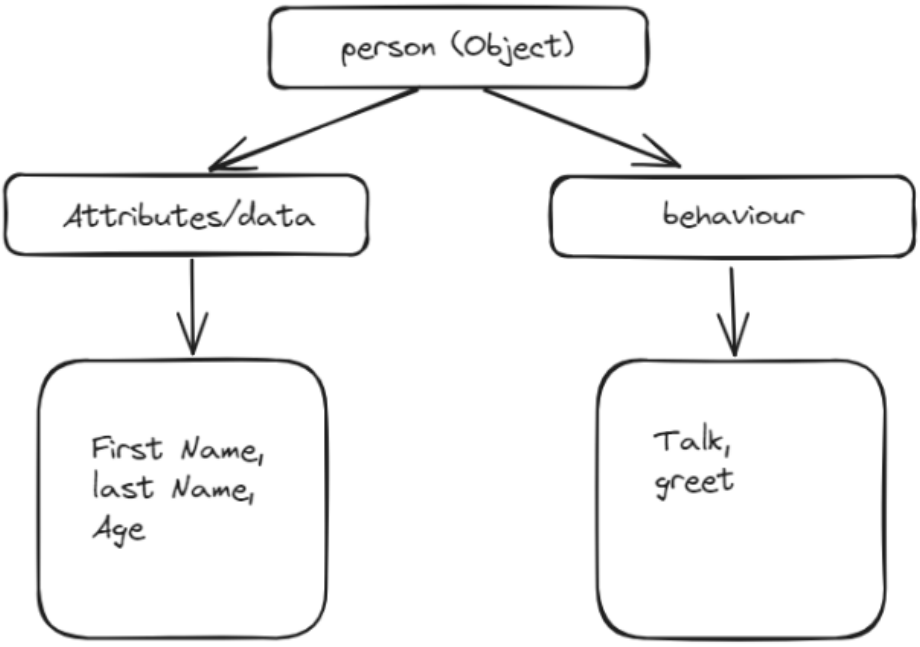
Topics

- What is an Object?
- Properties access via (objectName.propertyName, objectName["propertyName"] & objectName[expression])
- Add/remove properties/methods
- for...in
- in operator
- ComputedPropertyName
- optional chaining

What is an Object in JavaScript?

In JavaScript, an object is a powerful data structure that allows you to group related information and functions together. Imagine an object as a container that can hold different types of data and even behaviors.
Take, for instance, a car. A car can have a name, manufacturer, and fuel capacity which are examples of data. A car can be started, stopped, or accelerated. These are examples of behavior.

Pictorially, you can understand it as



Creating an object

There are majorly 3 ways to create an object in javascript :

1. By object literal
2. By creating an instance of Object directly (using a new keyword)
3. By using an object constructor (using a new keyword)

1. By object literal

This is the simplest and most common way to create an object. You define the properties and methods of the object directly within curly braces **{}**.

Syntex:

```
Object = { name1:value1,
    name2:value2,
... ,
nameN:valueN  };
```

Example

```
const person = {
    firstName: "John",
    lastName: "Doe",
    age: 30,
    greet: function() {
        console.log("Hello!");
    };
};
```

2. Creating an Object Using Object Constructor:

You can also create an object using the Object constructor function and the new keyword:

Syntex:

```
objectName = new Object();
```

Example

```
const person = new Object();
person.firstName = "John";
person.lastName = "Doe";
person.age = 30;
person.greet = function() {
    console.log("Hello!");
};
```

While this method works, it's less commonly used compared to object literal notation.

3. Creating an Object Using Object Constructor Function:

You can create custom constructor functions to create objects with shared properties and methods. This is a way to define a "blueprint" for creating objects.

Example

```
function Person(firstName, lastName, age) {
    this.firstName = firstName;
    this.lastName = lastName;
    this.age = age;
    this.greet = function() {
        console.log("Hello!");
    };
}

const person = new Person("John", "Doe", 30);
```

This method is especially useful when you want to create multiple objects with the same structure and behavior.

Properties access via (objectName.propertyName, objectName["propertyName"] & objectName[expression])

Properties in JavaScript objects can be accessed in different ways: via dot notation, square bracket notation, and using expressions. Let's explain this with an example:

```
const person = {
    firstName: "John",
    lastName: "Doe",
    age: 30,
    greet: function() {
        console.log("Hello!");
    }
};
```

1. Dot Notation

You can access object properties using dot notation, like this:

```
console.log(person.firstName); // Outputs: "John"
console.log(person.age);       // Outputs: 30
person.greet();              // Calls the greet method and
outputs: "Hello!"
```

Dot notation is the most common and straightforward way to access object properties.

2. Square Bracket Notation:

You can also access object properties using square brackets, like this:

```
console.log(person["lastName"]); // Outputs: "Doe"
```

Square bracket notation allows you to use a string (in quotes) to access the property. This can be helpful when property names contain special characters, or spaces, or are stored as variables.

Lets access properties with special characters:

```
const person = {
  "first-name": "John",
  "last-name": "Doe",
  "#id": 12345
};

// Using square bracket notation to access properties with
special characters
console.log(person["first-name"]); // "John"
console.log(person["last-name"]);  // "Doe"
console.log(person["#id"]);        // 12345
```

In this example, we have properties with special characters, such as hyphens and a hashtag. Square bracket notation allows us to access these properties by providing the property name as a string within square brackets.

3. Using Expressions:

You can use expressions within square brackets to access properties dynamically:

```
const propertyName = "age";
console.log(person[propertyName]); // Outputs: 30
```

Here, we use the propertyName variable within square brackets to access the "age" property dynamically.

Adding and Removing Properties/Methods

Objects in JavaScript are flexible; you can modify them by adding new properties or methods.

1. Adding Properties:

To add a new property to an object, assign a value to a new or existing property name.

Continuing the above example let's add more properties to a person object:

```
person.city = "New York";
console.log(person.city); // Output: New York
```

2. Adding Methods:

Methods are functions that belong to an object. You can add them just like properties.

```
person.sayHello = function() {
    console.log("Hello!");
};
person.sayHello() // Hello!
```

3. Removing Properties/Methods:

Use the delete keyword to remove a property or method from an object.

```
delete person.city;
console.log(person.city); // Output: undefined
```

Iterating through Object Properties with for ... in.

The **for...in** loop in JavaScript is used to iterate over the properties (keys) of an object. It's particularly useful for iterating over the keys of objects, such as object properties or array indices. Here's how it works:

Syntax

```
for (variable in object) {
    // code to be executed for each property
}
```

variable: This is a variable that represents the current property key in each iteration.

object: The object you want to loop through.

Example

```
const person = {
    firstName: "John",
    lastName: "Doe",
    age: 30
};
for (const key in person) {
    console.log(key + ": " + person[key]);
}
```

In this example, we use a for...in loop to iterate through the properties of the person object, and we log each property name and its value. The output will be:

```
firstName: John
lastName: Doe
age: 30
```

In operator

The in operator is used to check if a particular property exists in an object. It returns true if the property is found in the object and false otherwise.

Here's the basic syntax of the in-operator:

```
propertyKey in object
```

propertyKey: The name of the property you want to check for existence.

object: The object in which you want to check for the property.

Example

```
const person = {
  firstName: "John",
  lastName: "Doe",
  age: 30
};
// Check if 'firstName' property exists in 'person'
const hasFirstName = "firstName" in person;
console.log(hasFirstName); // true

// Check if 'email' property exists in 'person'
const hasEmail = "email" in person;
console.log(hasEmail); // false
```

In this example:

- We use **"firstName"** and **"email"** as property keys to check if they exist in the person object.
- **hasFirstName** is **true** because the **firstName** property is present in the **person** object.
- **hasEmail** is **false** because the **email** property is not found in the **person** object.

The **in** operator is a useful way to determine if a property is defined in an object, which can be especially helpful for conditional logic or dynamic property access in your JavaScript code.

ComputedPropertyName

Computed property names in JavaScript are a feature that allows you to dynamically compute the name of an object's property using an expression. These expressions are enclosed in square brackets ([]) within the object literal. Computed property names are particularly useful when you want to create object properties with names determined at runtime. Here's how they work:

syntax

```
const obj = {
  [expression]: value,
};
```

- **expression:** This is an expression that computes the property name dynamically.
- **value:** The value associated with the computed property name.

Example

```
const propertyName = "age";
const person = {
  name: "John",
  [propertyName]: 30,
};

console.log(person.age); // Outputs: 30
```

In this example, the property name "age" is computed using the propertyName variable, which contains the string "age." This allows you to create a property with a dynamic name based on the value of propertyName.

optional chaining

Optional chaining in JavaScript is a feature that allows you to safely access properties and methods of objects without causing errors when the properties or methods are missing or undefined. It uses the ?. syntax to handle potentially missing properties and gracefully return undefined instead of causing runtime errors. This is particularly useful for dealing with nested data structures and making your code more resilient.

Syntax

```
object?.property
object?.method()
object?.[expression]
```

Here's how it works:

1) Property Access (.property):

Normally, if you try to access a property that doesn't exist in an object, it results in an error.
With optional chaining, you can use ?. after the object to access the property safely. If the property exists, its value is returned. If it doesn't, undefined is returned instead of causing an error.

```
const user = {
  id: 1,
  name: "John",
  address: {
    street: "123 Main St",
    city: "Nagpur",
  },
};

// Let's use optional chaining to access the 'city' property
within the 'address' object.

const userCity = user.address?.city;
const userState = user.address?.state

console.log(userCity); // This will log 'Nagpur' since 'city'
exists in the 'address' object.

console.log(userState); // This will log 'undefined' since
'state' does not exists in the 'address' object.
```

2) Method Call (.method()):

Similarly, you can use optional chaining to call a method on an object. If the method exists, it's called; if not, it doesn't cause an error.

Example

```
};

// Let's use optional chaining to call the
'getNotificationStatus' method.

const notificationStatus =
user.preferences?.getNotificationStatus?.();
const onlineStatus = user.preferences?.online?.();



console.log(notificationStatus); // This will log
'Notifications enabled' since the method exists.

console.log(onlineStatus); // this will log "undefined" since
the method does not esists.
```

3) Dynamic Property Access ([expression]):

You can also use optional chaining with dynamic property names, allowing you to access properties using variables.

```
const user = {
  id: 1,
  name: "John",
  address: {
    street: "123 Main St",
    city: "Nagpur",
  }
};
// First Example: Accessing an existing property
const propertyName = "city";
const userCity = user.address?.[propertyName];

console.log(userCity); // This will log 'Nagpur' because the
'city' property exists in 'address'.

// Second Example: Accessing a non-existing property
const propertyName = "state";
const userState = user.address?.[propertyName];

console.log(userState); // This will log 'undefined' because
the 'state' property doesn't exist in 'address'.
```