SUNSTONE

JavaScript

LECTURE 10 NOTES

Objects

- **Objects, Methods, and Properties in Real Life**
  - In real life, a car is an object.
  - A car has properties such as weight and color, and methods such as start and stop

Object	Properties	Methods

	car.name = Fiat
	car.model = 500
	car.weight = 850kg
	car.color = white
		car.start()
		car.drive()
		car.brake()
		car.stop()

  - All cars have the same properties, but the property values differ from car to car.
  - All cars have the same methods, but the methods are performed at different times.
  - JavaScript is an Object Oriented Programming (OOP) language. A programming language can be called object-oriented if it provides four basic capabilities to developers:
    - **Encapsulation:** The capability to store related information, whether data or methods, together in an object.
    - **Aggregation:** The capability to store one object inside another object.
    - **Inheritance:** The capability of a class to rely upon another class (or number of classes) for some of its properties and methods.

- **Polymorphism:** The capability to write one function or method that works in a variety of different ways.

- Objects are composed of attributes. If an attribute contains a function, it is considered to be a method of the object, otherwise, the attribute is considered a property.

What are Objects, Methods, and Properties in JS?

Objects

- In JavaScript, objects are a fundamental data structure used to represent real-world entities or concepts. They are collections of key-value pairs, where each key is a unique identifier called a property, and the value can be any valid JavaScript data type (such as numbers, strings, arrays, functions, or even other objects).
- Objects in JavaScript are dynamic, meaning that properties can be added, modified, or removed at any time.

Object Literals

- Object literals are a convenient way to create and initialize objects in JavaScript. They allow you to define an object and its properties in a concise syntax using curly braces ({}) and key-value pairs.
- Here's an example of an object literal representing a person:

```
const person = {
  name: "John",
  age: 30,
  gender: "male"
};
```

Properties

- Properties are the individual attributes or characteristics of an object. They consist of a key and a corresponding value.

- You can access and manipulate object properties using dot notation (objectName.propertyName) or bracket notation (objectName["propertyName"]).
- Here's an example demonstrating property access and modification:

```
console.log(person.name);          // Output: John
person.age = 35;         // Modifying the value of the 'age'
propertyconsole.log(person.age);   // Output: 35
```

➢ Adding and Removing Properties

- Object properties can be added or removed dynamically.
- To add a new property to an object, you can simply assign a value to a new key:

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

- To remove a property from an object, you can use the delete keyword:

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

➢ Object Methods

- Object properties can also hold functions, which are then referred to as methods. Methods allow objects to have behavior associated with them.
- Here's an example of an object with a method:

```
const calculator = {
  add: function(a, b) {
    return a + b;
  }
};
console.log(calculator.add(2, 3)); // Output: 5
```

- **Creating Objects in JavaScript**
    - There are three ways to create objects.
        1. By object literal
        2. By creating an instance of the object directly (using new keyword)
        3. By using an object constructor (using new keyword)

1. JavaScript Object by object literal

- The syntax of creating an object using object literal is given below:

```
object={property1:value1,property2:value2.....propertyN:valueN}
```

- Property and value is separated by : (colon).
- **For example,**

```
<script>

emp={id:102,name:"Shyam Kumar",salary:40000}

document.write(emp.id+" "+emp.name+" "+emp.salary);

</script>
```

Output:

```
102 Shyam Kumar 40000
```

2. By creating an instance of the object

- The syntax of creating an object directly is given below:

```
var objectname=new Object();
```

- Here, the ‘new’ keyword is used to create the object.
- **For example,**

```
<script>
```

```
var emp=new Object();
emp.id=101;
emp.name="Ravi";
emp.salary=50000;
document.write(emp.id+" "+emp.name+" "+emp.salary);
</script>
```

Output:

```
101 Ravi 50000
```

3. By using an object constructor

- Here, you need to create a function with arguments. Each argument value can be assigned in the current object by using this keyword.
- ‘this’ keyword refers to the current object.
- **For example,**

```
<script>
function emp(id,name,salary){
this.id=id;
this.name=name;
this.salary=salary;
}
e=new emp(103,"Vimal Jaiswal",30000);
document.write(e.id+" "+e.name+" "+e.salary);
</script>
```

Output:

```
103 Vimal Jaiswal 30000
```

➢ **Defining method in JavaScript object**

- We can define methods in JavaScript object. But before defining a method, we need to add property in the function with the same name as the method.
- The example of defining a method in object is given below.

```
function emp(id,name,salary){

this.id=id;

this.name=name;

this.salary=salary;

this.changeSalary=changeSalary;


function changeSalary(otherSalary){

this.salary=otherSalary;  }}


e=new emp(103,"Sonoo Jaiswal",30000);

document.write(e.id+" "+e.name+" "+e.salary);

e.changeSalary(45000);

document.write("<br>"+e.id+" "+e.name+" "+e.salary);
```

Output:

```
103 Sonoo Jaiswal 30000

103 Sonoo Jaiswal 45000
```

- **Prototypes and Inheritance**

➢ **Prototypes**

- In JavaScript, objects can inherit properties and methods from other objects through a mechanism called prototypes. This allows for code reuse and the creation of object hierarchies.

- Every object in JavaScript has a prototype, which is a reference to another object. The prototype object serves as a fallback for properties and methods that the current object doesn't have.
- Prototypes form a chain-like structure known as the prototype chain, allowing objects to inherit properties and methods from their prototypes.
- The prototype of an object can be accessed using the __proto__ property or the Object.getPrototypeOf() method.

➢ Inheritance

- In JavaScript, inheritance is achieved through prototypes. Objects can inherit properties and methods from their prototypes.
- To create an object that inherits from another object, you can use the Object.create() method or the class syntax introduced in ECMAScript 2015 (ES6).
- For example,

```
const personPrototype = {
  greeting: function() {
    console.log(`Hello, my name is ${this.name}.`);
  }};
const john = Object.create(personPrototype);
john.name = "John";
john.greeting();       // Output: Hello, my name is John.
```

➢ Constructor Functions and Prototypes

- Constructor functions are a traditional way of creating objects and establishing inheritance in JavaScript.
- Constructor functions are invoked with the new keyword to create instances of objects.

- The prototype property of a constructor function is used to add properties and methods that will be shared by all instances created with that constructor.
- For example,

```
function Person(name) {
  this.name = name;
}
Person.prototype.greeting = function() {
  console.log(`Hello, my name is ${this.name}.`);
};
const john = new Person("John");
john.greeting();        // Output: Hello, my name is John.
```

➢ Inheriting from Constructor Functions

- To establish inheritance from a constructor function, the prototype property of the child constructor is set to an instance of the parent constructor.
- For example,

```
function Student(name, grade) {
  Person.call(this, name);
  this.grade = grade;
}
Student.prototype                                              =
Object.create(Person.prototype);Student.prototype.constructor  =
Student;
Student.prototype.displayGrade = function() {
  console.log(`I am in grade ${this.grade}.`);
};
const jane = new Student("Jane", 5);
jane.greeting();        // Output: Hello, my name is Jane.
jane.displayGrade();    // Output: I am in grade 5.
```

➢ **Class Syntax and Inheritance**

- ECMAScript 2015 (ES6) introduced the class syntax, which provides a more convenient and readable way to define constructor functions and establish inheritance.
- The **'extends'** keyword is used to specify the parent class from which a child class inherits.
- For example,

```
class Person {
  constructor(name) {
    this.name = name;   }
  greeting() {
    console.log(`Hello, my name is ${this.name}.`);
  }
}
class Student extends Person {
  constructor(name, grade) {
    super(name);
    this.grade = grade;
  }
  displayGrade() {
    console.log(`I am in grade ${this.grade}.`);
  }
}
const jane = new Student("Jane", 5);
jane.greeting();

// Output: Hello, my name is Jane.
```