[2] OBJECTS

Objects in JavaScript allow you to structure and organize your data by creating collections of key-value pairs.

Creating Objects

You can create objects in several ways:

1. Object Literal Notation: The simplest way to create an object is using literal notation, where you define key-value pairs within curly braces `{}`.

```
const person = {
name: "John",
age: 30,
city: "New York"
};
```

2. Constructor Function: You can use constructor functions to create objects, typically when you need to create multiple instances of a similar object.

```
function Person(name, age, city) {
this.name = name;
this.age = age;
this.city = city;
}
const john = new Person("John", 30, "New York");
```

3. Object.create(): You can create objects with a specified prototype using `Object.create()`.

```
const personPrototype = {
greet: function() {
console.log(`Hello, my name is ${this.name}.`);
}
};

const john = Object.create(personPrototype);
john.name = "John";
john.age = 30;
```

Accessing Object Properties

You can access object properties using dot notation or bracket notation:

```
console.log(person.name); // "John"
console.log(person["age"]); // 30
```

Bracket notation is useful when the property name is dynamic or contains special characters.

Modifying Object Properties

You can modify object properties directly:

```
person.age = 31;
```

You can also add new properties:

```
person.email = "john@example.com";
```

Checking for Object Properties

To check if an object has a specific property, you can use the `in` operator or `hasOwnProperty()` method:

```
console.log("name" in person); // true
console.log(person.hasOwnProperty("email")); // false
```

Removing Object Properties

You can delete object properties using the `delete` operator:

```
delete person.city;
```

Iterating Over Object Properties

You can iterate over object properties using `for...in` loops:

```
for (let key in person) {
console.log(key, person[key]);
}
```

However, be cautious when using `for...in`, as it iterates over all enumerable properties, including properties inherited from the prototype chain. To avoid this, you can use `Object.hasOwnProperty()` to filter out inherited properties.

Object Methods

Objects can also contain functions, which are called methods. Methods can be used to perform actions associated with the object:

```
const person = {
name: "John",
age: 30,
sayHello: function() {
console.log(`Hello, my name is ${this.name}.`);
}
};
person.sayHello(); // "Hello, my name is John."
```

Object Destructuring

You can use object destructuring to extract object properties into variables:

```
const { name, age } = person;
console.log(name, age); // "John" 30
```

Object Cloning

To create a new object that is a copy of an existing object, you can use various methods:

- Spread Operator: This creates a shallow copy.

```
const clone = { ...person };
```

- Object.assign()**: This method copies enumerable properties from one or more source objects to a target object.

```
const clone = Object.assign({}, person);
```

Object Prototypes and Inheritance

JavaScript uses prototypes for object inheritance. You can create object hierarchies and inherit properties and methods from other objects.

```
// Parent object
const animal = {
speak: function() {
console.log("Some sound.");
}
```

```
// Child object inheriting from parent
const cat = Object.create(animal);
cat.speak(); // "Some sound."
```