JAVASCRIPT

OBJECT

JavaScript Objects

What are Objects?

Objects are a fundamental data structure in JavaScript. They are collections of key-value pairs, which are used to store and access data. Objects are similar to arrays, but instead of using indexes to access data, they use keys. Keys can be strings, numbers, or symbols. Values can be any data type, including objects. Objects are declared using curly braces, and the key-value pairs are separated by commas. Keys are always followed by a colon, and values can be followed by a comma. Here is an example of an object:

index.js

```
const person = {
  name: 'John',
  age: 50,
  gender: 'male'

};
```

To access the values stored in an object, we use dot notation. We can also use bracket notation, which is similar to array notation. Here is an example of accessing the values stored in an object:

index.js

```
const name = person.name; // John

const age = person['age']; // 50
```

To add a new key-value pair to an object, we use dot notation. Here is an example of adding a new key-value pair to an object:

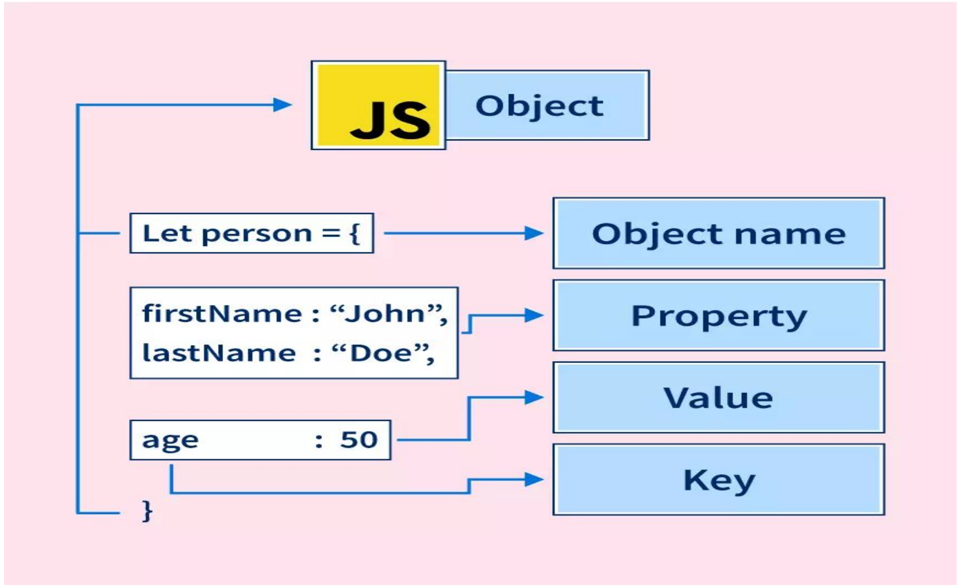
index.js

```
person.country = 'United States';
```

To delete a key-value pair from an object, we use the delete keyword. Here is an example of deleting a key-value pair from an object:

index.js

```
delete person.age;
```



Creating Objects in JavaScript

There are 3 ways to create objects.

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

1) JavaScript Object by object literal

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

```
object={property1:value1,property 2:value2.....property N:value N}
```

As you can see, property and value is separated by : (colon).

Let’s see the simple example of creating object in JavaScript.

index.js

```
emp={id:102,
name:"Bhuvanesh",
salary:40000}
console.log(emp.id+" "+emp.name+" "+emp.salary);
//  102 Bhuvanesh 40000
```

2) By creating instance of Object

The syntax of creating object directly is given below:

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

Here, **new keyword** is used to create object.

Let’s see the example of creating object directly.

index.js

```
var emp=new Object();

emp.id=101;

emp.name="Ravi Malik";

emp.salary=50000;
```

```
console.log(emp.id+" "+emp.name+" "+emp.salary);

//101 Ravi Malik 50000
```

3) By using an Object constructor

Here, you need to create function with arguments. Each argument value can be assigned in the current object by using this keyword.

The **this keyword** refers to the current object.

The example of creating object by object constructor is given below.

index.js

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

this.id=id;

this.name=name;

this.salary=salary;

}

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

console.log(e.id+" "+e.name+" "+e.salary);

// 103 Vimal Jaiswal 30000
```

Object Properties

In JavaScript, objects can contain properties that can be accessed and modified. The properties of an object are defined as a name-value pair. The name is a string that is used to access the value. The value can be of any type, including functions, arrays, and other objects. Properties are typically added to an object when it is created, but they can also be added or removed at any time. To access a property of an object, use the dot notation: `objectName.propertyName` To set a property of an object, use the assignment operator: `objectName.propertyName = value` To delete a property of an object, use the delete keyword: `delete objectName.propertyName` Here is an example of an object with two properties:

index.js

```
const person = {
  name: 'John',
  age: 30
};
```

```
console.log(person.name); // 'John'
console.log(person.age); // 30

person.name = 'Jane';
person.age = 25;

console.log(person.name); // 'Jane'
console.log(person.age); // 25

delete person.age;


console.log(person.age); // undefined
```

Enumerate Object Properties

Using a for...in loop:

The `for...in` loop allows you to iterate over all enumerable properties of an object, including properties inherited from its prototype chain.

index.js

```
const person = {
  firstName: "John",
  lastName: "Doe",
  age: 30
};

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

```
// firstName John

// lastName Doe

// age 30
```

This loop will output the property names and their corresponding values in the `person` object.

Object.keys():

The `Object.keys()` method returns an array of the object's own enumerable property names. It allows you to iterate through the property names more easily.

index.js

```
const person = {
  firstName: 'John',
  lastName: 'Doe',
  age: 30
};

const keys = Object keys(person);

for (let key of keys) {              // firstName John
  console.log(key, person[key]);   // lastName Doe
}                                  // age 30
```

Object.values():

The `Object.values()` method returns an array of the object's own enumerable property values.

index.js

```
const
  firstName: 'John',
  lastName: 'Doe',
  age: 30
};
```

```
const values = Object.values(person);


for (let value of values) {
  console.log(value);
}
// John
// Doe
// 30
```

Object.entries():

The `Object.entries()` method returns an array of key-value pairs, where each pair is represented as an array with two elements: the property name and its corresponding value.

index.js

```
const person = {
  firstName: 'John',
  lastName: 'Doe',
  age: 30
};
```

```
const entries = Object.entries(person);
for (let [key, value] of entries) {
  console.log(key, value);
}
// firstName John
// lastName Doe
// age 30
```

Keep in mind that these methods will only enumerate the object's enumerable properties. Some properties, like those added via `Object.defineProperty()` with `{enumerable: false}`, won't be included in the enumeration. Also, properties from the object's prototype chain won't be enumerated unless you specifically check for them.

Nested Objects:

Nested objects are commonly used to organize and structure data more effectively, allowing for a clearer representation of complex information. You can nest objects within objects to multiple levels, creating a tree-like structure.

```javascript
// Creating a nested object
let person = {
  name: 'Alice',
  age: 30,
  address: {
  street: '123 Main St',
  city: 'Anytown',
  country: 'USA'
  }
};
```

```
// Accessing nested object properties

console.log(person.name); // Output: Alice

console.log(person.age); // Output: 30

console.log(person.address.city); // Output: Anytown
```

In this example, the `person` object contains a property called `address`, which is another object with its own properties like `street`, `city`, and `country`. You can access these nested properties using dot notation.

This

The `this` keyword in JavaScript is a reference to the object it belongs to. It is commonly used when defining object methods, to refer to the object of which the method is a property.

In JavaScript, the value of `this` is determined by how a function is called. In most cases, it refers to the object that owns the executing code. It can be assigned a different value when calling a function.

In the global execution context (outside of any function), this refers to the global object, whether in the browser or in Node.js.

Example:

index.js

```
const person = {
  name: 'John Doe',
  age: 30,
  greet() {
    console.log('Hello, my name is ' + this.name);
  }
};

person.greet(); //Hello, my name is John Doe
```

In the example above, `this` refers to the object `person`. In the method `greet`, we can use `this` to refer to the `person` object, and access its properties.

Delete operator

Once an object is created in JavaScript, it is possible to remove properties from the object using the `delete` operator. The `delete` keyword deletes both the value of the property and the property itself from the object. The `delete` operator only works on properties, not on variables or functions.

index.js

```
const person = {
 firstName: "Matilda",
 age: 27,
 hobby: "knitting",
 goal: "learning JavaScript"
};
```

```
delete person.hobby; // or delete person[hobby];

console.log(person);
/*
{
 firstName: "Matilda"
 age: 27
 goal: "learning JavaScript"
}
*/
```