JavaScript interview questions with answers:

What is closure in JavaScript and how does it work?

Answer: Closure is a feature in JavaScript where a function has access to its outer scope even after the outer function has returned. It is created when a function is defined inside another function, and the inner function retains access to the variables in the outer function's scope.

Closure Example 1:

```
function outerFunction(x) {
  return function innerFunction(y) {
    return x + y;
  };
}
const add5 = outerFunction(5);
console.log(add5(3)); // 8
```

Closure Example 2:

```
function outerFunction(x) {
  return function innerFunction(y) {
    return x + y;
  };
}

const add5 = outerFunction(5);
console.log(add5(3)); // 8
```

Closure Example A counter function:

```
function createCounter() {
  let count = 0;
  return function() {
```

```
    return ++count;
  };
}
const counter = createCounter();
console.log(counter()); // 1
console.log(counter()); // 2
```

Closure Example A function returning another function:

```
function greeting(message) {
  return function(name) {
    console.log(`${message} ${name}!`);
  };
}

const sayHi = greeting('Hi');
const sayHello = greeting('Hello');
sayHi('John'); // Hi John!
sayHello('Jane'); // Hello Jane!
```

Closure Example A timer:

```
function setTimer(duration) {
  let timeoutId;
  return function() {
    clearTimeout(timeoutId);
    timeoutId = setTimeout(function() {
      console.log(`${duration} seconds have passed.`);
    }, duration * 1000);
  };
}
const timer1 = setTimer(3);
const timer2 = setTimer(5);
```

```
timer1();
timer2();
```

Closure Example A module:

```
function createModule(name) {
  let module = {};
  return function(data) {
    module[name] = data;
    return module;
  };
}

const createPerson = createModule('person');
const createAddress = createModule('address');
const person = createPerson({ name: 'John', age: 32 });
const address = createAddress({ street: '123 Main St',
city: 'San Francisco' });
console.log(person); // { person: { name: 'John', age:
32 } }
console.log(address); // { address: { street: '123 Main
St', city: 'San Francisco' } }
```

Closure Example A function to maintain state:

```
function createToggler() {
  let state = false;
  return function() {
    state = !state;
    return state;
  };
}
const toggler = createToggler();
console.log(toggler()); // true
```

```
console.log(toggler()); // false
```

Can you explain hoisting in JavaScript?

Answer: Hoisting is a behavior in JavaScript where variable and function declarations are moved to the top of their scope, regardless of where they are declared in the code. This means that variables and functions can be called before they are declared, but the value of variables will be undefined until they are assigned a value.

`Hoisting` Example 1:

```
console.log(x); // undefined
var x = 5;
```

What is the difference between == and === in JavaScript?

Answer: The double equals (==) performs type coercion, which means it converts the data type of one operand to match the data type of the other operand before making a comparison. The triple equals (===) does not perform type coercion and only returns true if both operands have the same data type and value.

`== vs ===` Example 1:

```
console.log(5 == "5"); // true
console.log(5 === "5"); // false
console.log(null == undefined); // true
console.log(null === undefined); // false
```

How do you declare a variable in JavaScript?

Answer: Variables in JavaScript can be declared using the "var", "let" or "const" keywords. The "var" keyword is used to declare a variable with function scope, while "let" and "const" are used to declare variables with block scope.

`Variable declaration 1:`

```
var x = 5;
let y = 10;
const z = 15;
```

Variable declaration 2:

```
let x = 5;
x = 10;
console.log(x); // 10
const z = [1, 2, 3];
z.push(4);
console.log(z); // [1, 2, 3, 4]
```

Can you explain the event loop in JavaScript?

Answer: The event loop in JavaScript is a mechanism that allows the execution of code to be scheduled in a non-blocking way. The event loop continuously checks the message queue for new messages, and if there is a message, it executes the corresponding callback function. This allows the JavaScript engine to handle multiple events and execute code in a responsive and efficient manner.

Event loop 1:

```
console.log('Start');
setTimeout(function() {
  console.log('Timeout');
}, 0);
console.log('End');
```

Event loop 2:

```
console.log('Start');
setTimeout(function() {
  console.log('Timeout');
```

```
}, 2000);
console.log('End');
```

Event loop Using setTimeout with a queue:

```
console.log('Start');
setTimeout(() => console.log('Timeout 1'), 0);
setTimeout(() => console.log('Timeout 2'), 0);
console.log('End');
```

This will log:
Start
End
Timeout 1
Timeout 2

Event loop Handling multiple callbacks with setInterval:

```
let count = 0;
const intervalId = setInterval(() => {
  console.log(`Interval: ${count++}`);
  if (count === 5) {
    clearInterval(intervalId);
  }
}, 1000);
```

This will log:
Interval: 0
Interval: 1
Interval: 2
Interval: 3
Interval: 4

Event loop Chaining promises:

```
const delay = (duration) => new Promise(resolve =>
setTimeout(resolve, duration));
console.log('Start');
delay(1000)
  .then(() => console.log('Promise 1'))
  .then(() => delay(1000))
  .then(() => console.log('Promise 2'));
console.log('End');
```

This will log:

Start
End
Promise 1
Promise 2

Event loop Processing multiple tasks with setImmediate:

```
console.log('Start');
setImmediate(() => console.log('Immediate 1'));
setImmediate(() => console.log('Immediate 2'));
console.log('End');
```

This will log:

Start
End
Immediate 1
Immediate 2

What is the difference between null and undefined in JavaScript?

Answer: Undefined means a variable has been declared but has not been assigned a value, while null is a value assigned to a variable that represents no value or no object.

null vs undefined example:

```
let x;
console.log(x); // undefined
x = null;
console.log(x); // null
```

What is a JavaScript callback function?

Answer: A callback function is a function passed as an argument to another function, which is then executed at a later time. Callback functions are commonly used in JavaScript to handle asynchronous operations, such as making a network request or setTimeout().

Callback function example 1:

```
function sayHello(name, callback) {
  console.log(`Hello, ${name}!`);
  callback();
}
sayHello('John', function() {
  console.log('Callback executed');
});
```

Callback function example 2:

```
function calculate(a, b, callback) {
  const result = a + b;
  callback(result);
```

```
}
calculate(5, 3, function(result) {
  console.log(result); // 8
});
```

A simple callback with setTimeout:

```
const log = (message) => console.log(message);
setTimeout(() => log('Timeout'), 1000);
```

A callback with Array.forEach:

```
const numbers = [1, 2, 3, 4, 5];
numbers.forEach((number) => console.log(number));
```

A callback with Array.map:

```
const numbers = [1, 2, 3, 4, 5];
const doubledNumbers = numbers.map((number) => number *
2);
console.log(doubledNumbers);
```

A callback with Array.filter:

```
const numbers = [1, 2, 3, 4, 5];
const evenNumbers = numbers.filter((number) => number % 2
=== 0);
console.log(evenNumbers);
```

A callback with Array.reduce:

```
const numbers = [1, 2, 3, 4, 5];
const sum = numbers.reduce((total, number) => total + number,
0);
console.log(sum);
```

A callback with Array.sort:

```
const names = ['John', 'Jane', 'Jim', 'Jake'];
const sortedNames = names.sort((a, b) => a.localeCompare(b));
console.log(sortedNames);
```

A callback with Object.keys:

```
const person = { name: 'John', age: 30, city: 'London' };
Object.keys(person).forEach((key) => console.log(`${key}:
${person[key]}`));
```

A callback with Promise.then:

```
const delay = (duration) => new Promise(resolve =>
setTimeout(resolve, duration));
console.log('Start');
delay(1000)
  .then(() => console.log('Promise'));
console.log('End');
```

A callback with EventTarget.addEventListener:

```
const button = document.getElementById('button');
button.addEventListener('click', () => console.log('Clicked'));
```

A callback with XMLHttpRequest.onload:

```
const request = new XMLHttpRequest();
request.open('GET',
'https://jsonplaceholder.typicode.com/posts');
request.onload = () => {
  if (request.status === 200) {
    console.log(JSON.parse(request.responseText));
  } else {
    console.error(request.statusText);
  }
```

```
};
request.send();
```

How do you declare a function in JavaScript?

Answer: Functions in JavaScript can be declared using the "function" keyword followed by the function name, its parameters in parentheses, and its code block in curly braces. Functions can also be declared using function expressions and arrow functions.

Function declaration 1:

```
function sayHello(name) {
  console.log(`Hello, ${name}!`);
}
const greet = function(name) {
  console.log(`Hello, ${name}!`);
};
const hello = name => console.log(`Hello, ${name}!`);
```

Function declaration 2:

```
function sayHi(name) {
  console.log(`Hi, ${name}!`);
}
const greet = function(name) {
  console.log(`Hello, ${name}!`);
};
const hello = name => console.log(`Hello, ${name}!`);
sayHi('John'); // Hi, John!
greet('John'); // Hello, John!
hello('John'); // Hello, John!
```

What is the difference between let and var in JavaScript?

Answer: The "let" and "var" keywords are used to declare variables in JavaScript, but they have different scoping rules. Variables declared with "var" have function scope, which means they can be accessed within the entire function, while "let" variables have block scope, which means they can only be accessed within the block they are declared in.

let vs var example 1:

```
var x = 5;
if (true) {
  var x = 10;
}
console.log(x); // 10
let y = 5;
if (true) {
  let y = 10;
}
console.log(y); // 5
```

let vs var example 2:

```
for (var i = 0; i < 5; i++) {
  console.log(i);
}
console.log(i); // 5
for (let j = 0; j < 5; j++) {
  console.log(j);
}
console.log(j); // ReferenceError: j is not defined
```

Can you explain the "this" keyword in JavaScript?

Answer: The "this" keyword in JavaScript refers to the object that the function is a method of. Its value depends on how the function is called, and it can be set using bind, call or apply methods. If a function is not a method of an object, "this" refers to the global object (window in the browser).

"this" keyword example 1:

```
const person = {
  name: 'John',
  sayHello: function() {
    console.log(`Hello, I'm ${this.name}`);
  }
};
person.sayHello(); // Hello, I'm John
const greet = person.sayHello;
greet(); // Hello, I'm undefined
```

"this" keyword example 2:

```
const car = {
  brand: 'Tesla',
  drive: function() {
    console.log(`Driving a ${this.brand} car`);
  }
};
car.drive(); // Driving a Tesla car
const driveCar = car.drive;
driveCar(); // Driving a undefined car
```

examples of the this keyword in JavaScript:

this in an object method:

```
const person = {
  name: 'John',
  sayHello: function () {
```

```
    console.log(`Hello, my name is ${this.name}`);
  }
};

person.sayHello();
```

this in a class method:

```
class Person {
  constructor(name) {
    this.name = name;
  }
  sayHello() {
    console.log(`Hello, my name is ${this.name}`);
  }
}

const john = new Person('John');
john.sayHello();
```

this in an arrow function:

```
const person = {
  name: 'John',
  sayHello: () => {
    console.log(`Hello, my name is ${this.name}`);
  }
};

person.sayHello();
```

this in a function passed as a callback:

```
const person = {
  name: 'John'
```

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

setTimeout(logName.bind(person), 1000);
```

this in an event handler:

```
const button = document.getElementById('button');
button.addEventListener('click', function () {
  console.log(`Button with ID "${this.id}" was clicked`);
});
```

this in an object constructor:

```
function Person(name) {
  this.name = name;
}
const john = new Person('John');
console.log(john.name);
```

this in a closure:

```
const person = {
  name: 'John',
  sayHello: function () {
    const that = this;
    setTimeout(function () {
      console.log(`Hello, my name is ${that.name}`);
    }, 1000);
  }
};
person.sayHello();
```

this in a prototype method:

```
function Person(name) {
  this.name = name;
}
Person.prototype.sayHello = function () {
  console.log(`Hello, my name is ${this.name}`);
};
const john = new Person('John');
john.sayHello();
```

this in a dynamic method:

```
const person = {
  name: 'John'
};
const method = 'sayHello';
person[method] = function () {
  console.log(`Hello, my name is ${this.name}`);
};

person.sayHello();
```

this in a nested function:

```
const person = {
  name: 'John',
  sayHello: function () {
    const nested = function () {
      console.log(`Hello, my name is ${this.name}`);
    };
    nested.call(person);
  }
};
person
```

JavaScript interview questions with code examples:

What is hoisting in JavaScript and how does it work?

```
console.log(hoistedVariable); // undefined
var hoistedVariable = 'This is a hoisted variable';
console.log(notHoisted); // ReferenceError: notHoisted is not defined
let notHoisted = 'This is not a hoisted variable';
```

What is closure in JavaScript and how is it useful?

```
function outerFunction(x) {
  return function innerFunction(y) {
    return x + y;
  };
}
const add5 = outerFunction(5);
console.log(add5(3)); // 8
```

What is the difference between == and === in JavaScript?

```
console.log(1 == '1'); // true
console.log(1 === '1'); // false
```

What is the difference between null and undefined in JavaScript?

```
let variable1;
console.log(variable1); // undefined
let variable2 = null;
console.log(variable2); // null
```

How does asynchronous code work in JavaScript?

```
console.log('Before setTimeout');
setTimeout(function () {
  console.log('Inside setTimeout');
}, 1000);
console.log('After setTimeout');
```

What is the difference between let and var in JavaScript?

```
if (true) {
  var variable1 = 'This is a var variable';
  let variable2 = 'This is a let variable';
}
console.log(variable1); // This is a var variable
console.log(variable2); // ReferenceError: variable2 is not defined
```

How do you declare a variable in JavaScript?

```
var variable1 = 'This is a var variable';
let variable2 = 'This is a let variable';
const variable3 = 'This is a const variable';
```

What is the difference between forEach and map in JavaScript?

```
const numbers = [1, 2, 3, 4, 5];
numbers.forEach(function (number) {
  console.log(number);
});
const doubledNumbers = numbers.map(function (number) {
  return number * 2;
});
console.log(doubledNumbers);
```

What is the difference between function and arrow function in JavaScript?

```
function regularFunction(x, y) {
  return x + y;
}
const arrowFunction = (x, y) => x + y;
console.log(regularFunction(1, 2)); // 3
console.log(arrowFunction(1, 2)); // 3
```

How do you declare an object in JavaScript?

```
const objectLiteral = {
  key1: 'value1',
  key2: 'value2'
};
const objectConstructor = new Object();
objectConstructor.key1 = 'value1';
objectConstructor.key2 = 'value2';
```