JavaScript Coding Examples

Variables:

```
let name = "John Doe";
let age = 30;
let isStudent = false;
console.log(name);
console.log(age);
```

```
console.log(isStudent);
```

Explanation: In this code, we are declaring 3 variables using the let keyword. The let keyword allows us to declare variables in JavaScript. The first variable name is assigned a string value of "John Doe". The second variable age is assigned a number value of 30. The third variable isStudent is assigned a boolean value of false. Finally, we log the values of these variables to the console using the console.log method.

Arrays:

```
let names = ["John", "Jane", "Jim"];
console.log(names[0]);
console.log(names.length);
names.push("Jake");
console.log(names);
```

Explanation: In this code, we are declaring an array named names that contains three string values "John", "Jane", and "Jim". The first console.log statement logs the first element of the array to the console, which is "John". The second console.log statement logs the length of the array to the console, which is 3. The push method is then used to add another element "Jake" to

the end of the array. The final console.log statement logs the updated array to the console.

Example : Array

```
let fruits = ["apple", "banana", "cherry"];
for (let i = 0; i < fruits.length; i++) {
  console.log("I like " + fruits[i]);
}
```

Explanation: This example demonstrates the use of an array. We create an array fruits containing three strings "apple", "banana", and "cherry". We then use a for loop to iterate over each element in the array and log a message to the console using console.log(). This will log the message "I like apple", "I like banana", and "I like cherry" to the console.

Example: Object

```
let car = {
  make: "Toyota",
  model: "Camry",
  year: 2022,
};
```

```
console.log("I drive a " + car.year + " " + car.make +
" " + car.model);
```

Explanation: This example demonstrates the use of an object. We create an object car with three properties make, model, and year. We then use dot notation

Objects:

```
let person = {
  name: "John Doe",
  age: 30,
  isStudent: false
};
console.log(person.name);
console.log(person.age);
console.log(person.isStudent);
```

Explanation: In this code, we are declaring an object named person with properties name, age, and isStudent. The properties of the object store values "John Doe", 30, and false respectively. The console.log statements log the values of the properties to the console.

Conditional Statements:

```
let grade = 75;
if (grade >= 60) {
  console.log("Passed");
} else {
  console.log("Failed");
}
```

Explanation: In this code, we are declaring a variable grade with a value of 75. We then use an if...else statement to determine whether the student has passed or failed based on their grade. If the value of grade is greater than or equal to 60, the code inside the first block (i.e., console.log("Passed")) will be executed, and the message "Passed" will be logged to the console. If the value of grade is less than 60, the code inside the second block (i.e., console.log("Failed")) will be executed, and the message "Failed" will be logged to the console.

Functions:

```
function greet(name) {
  console.log("Hello, " + name);
}
greet("John");
```

Explanation: In this code, we are defining a function named greet that takes a parameter name. The function logs a greeting message to the console by concatenating the string "Hello, " with the value of the name parameter. The function is then called by passing a string argument "John" to it, which results in the message "Hello, John" being logged to the console.

Example: Simple Function

```
function greet(name) {
  console.log("Hello " + name);
}

greet("John");
```

Explanation: In this example, we have defined a simple function greet that takes a single argument name and logs a greeting to the console using console.log(). To call the function, we pass in an argument, such as "John", to the function and invoke it. This will log the message "Hello John" to the console.

Example: Conditional Statement

```
let num = 5;
if (num > 0) {
  console.log(num + " is a positive number");
} else {
  console.log(num + " is a negative number");
}
```

Explanation: This example demonstrates the use of a conditional statement using an if statement. The code checks if the value of num is greater than 0. If it is, the code inside the first set of curly braces is executed, and the message "5 is a positive number" is logged to the console. If the value of num is not greater than 0, the code inside the else block is executed, and the message "5 is a negative number" is logged to the console.

String Methods:

```
let message = "Hello World";
console.log(message.toUpperCase());
console.log(message.includes("Hello"));
```

Explanation: In this code, we are declaring a variable message with a string value of "Hello World". The first console.log statement logs the result of calling the toUpperCase method on

the message string, which returns an uppercase version of the string: "HELLO WORLD". The second console.log statement logs the result of calling the includes method on the message string, which checks if the string "Hello" is a part of the message string. Since it is, the method returns true, which is then logged to the console.

Ternary Operators:

```
let grade = 75;
let result = (grade >= 60) ? "Passed" : "Failed";
console.log(result);
```

Explanation: In this code, we are declaring a variable grade with a value of 75 and using a ternary operator to assign a value to the result variable based on the value of grade. The expression (grade >= 60) ? "Passed" : "Failed" says that if grade is greater than or equal to 60, the value of result should be "Passed", otherwise it should be "Failed". The final console.log statement logs the value of result to the console.

Using if statements:

```
let age = 25;
if (age >= 18) {
```

```
  console.log("You are an adult.");
} else {
  console.log("You are not an adult.");
}
```

Explanation: In this code, we are declaring a variable age with the value 25. We are then using an if statement to determine if the value of age is greater than or equal to 18. If it is, the first console.log statement is executed, logging "You are an adult." to the console. If not, the else block is executed, logging "You are not an adult." to the console.

Using the switch statement:

```
let day = 2;
switch (day) {
  case 1:
    console.log("Monday");
    break;
  case 2:
    console.log("Tuesday");
    break;
  case 3:
    console.log("Wednesday");
```

```
    break;
  default:
    console.log("Invalid day");
}
```

Explanation: In this code, we are using a switch statement to control the flow of the program based on the value of a variable. The switch statement takes an expression as an argument and compares it to the case labels. If a match is found, the code inside the corresponding case block is executed. If no match is found, the code inside the default block is executed. In this case, the value of day is 2, so the code inside the case 2 block is executed and the result "Tuesday" is logged to the console.

Example of switch statements:

```
let grade = "A";
switch (grade) {
  case "A":
    console.log("Excellent");
    break;
  case "B":
    console.log("Good");
    break;
```

```
  case "C":
    console.log("Average");
    break;
  default:
    console.log("Invalid grade");
    break;
}
```

Explanation: In this code, we are declaring a variable grade with the value "A". We are then using a switch statement to match the value of grade with different cases. If the value of grade is "A", the first console.log statement is executed, logging "Excellent" to the console. If the value of grade is "B", the second console.log statement is executed, logging "Good" to the console. If the value of grade is "C", the third console.log statement is executed, logging "Average" to the console. If the value of grade does not match any of the cases, the default block is executed, logging "Invalid grade" to the console.

Using an object to store data:

```
let person = {
  name: "John",
  age: 30,
```

```
  occupation: "Teacher"
};
console.log(person.name);
```

Explanation: In this code, we are using an object to store data about a person. An object is a collection of key-value pairs, and in this case, the keys are name, age, and occupation. The values are the corresponding data for each key. To access the data stored in an object, we use dot notation, such as person.name to access the value of the name key. The value of person.name is logged to the console using the console.log() method.

Using a function to define a reusable piece of code:

```
function sayHello(name) {
  console.log(`Hello, ${name}!`);
}
sayHello("John");
```

Explanation: In this code, we are using a function to define a reusable piece of code. A function is a block of code that can be executed repeatedly with different arguments. In this case, the function sayHello takes an argument name and logs a greeting to

the console using the console.log() method. To call the function, we use its name followed by a set of parentheses, such as sayHello("John"). This causes the function to run and the greeting "Hello, John!" is logged to the console.

Anonymous Functions:

```
let greet = function(name) {
  console.log("Hello, " + name);
};
greet("John");
```

Explanation: In this code, we are declaring an anonymous function and assigning it to the variable greet. The function takes a parameter name and logs a greeting message to the console by concatenating the string "Hello, " with the value of the name parameter. The function is then called by passing a string argument "John" to it, which results in the message "Hello, John" being logged to the console.

Arrow Functions:

```
let add = (a, b) => a + b;
console.log(add(2, 3));
```

Explanation: In this code, we are declaring an arrow function named add that takes two parameters, a and b, and returns their sum. The final console.log statement calls the add function, passing the values 2 and 3 as arguments, and logs the result to the console, which is 5.

Example: For Loop

```
for (let i = 0; i < 5; i++) {
  console.log("The value of i is: " + i);
}
```

Explanation: This example demonstrates the use of a for loop. The loop initializes the variable i to 0 and runs the loop as long as i is less than 5. The loop increments the value of i by 1 with each iteration. This will log the message "The value of i is: 0", "The value of i is: 1", "The value of i is: 2", "The value of i is: 3", and "The value of i is: 4" to the console.

While Loops:

```
let i = 0;
while (i < 5) {
  console.log(i);
  i++;
}
```

Explanation: In this code, we are using a while loop to log the values 0 to 4 to the console. The loop initializes a variable i with a value of 0. The condition in the loop checks if i is less than 5. If the condition is true, the code inside the loop will be executed, and the value of i will be logged to the console.

Using a for loop to iterate through an array:

```
let numbers = [1, 2, 3, 4, 5];
for (let i = 0; i < numbers.length; i++) {
  console.log(numbers[i]);
}
```

Explanation: In this code, we are using a for loop to iterate through an array of numbers. The loop uses the variable i as a counter, and it continues to run as long as i is less than the length of the numbers array. On each iteration of the loop, the current value of numbers[i] is logged to the console using the console.log() method.

Using a while loop to calculate the factorial of a number:

```
let number = 5;
let factorial = 1;
while (number > 1) {
  factorial *= number;
  number--;
}
console.log(factorial);
```

Explanation: In this code, we are using a while loop to calculate the factorial of a number. The while loop continues to run as long as the value of number is greater than 1. On each iteration of the loop, the value of factorial is multiplied by number, and then number is decremented by 1. The final value of factorial is the factorial of the original number, and it is logged to the console using the console.log() method.

Loops:

```
for (let i = 0; i < 5; i++) {
  console.log(i);
}
```

Explanation: In this code, we are using a for loop to log the values 0 to 4 to the console. The loop initializes a variable i with a value of 0. The condition in the loop checks if i is less than 5. If the condition is true, the code inside the loop will be executed, and the value of i will be logged to the console. The final statement in the loop increments the value of i by 1. This process continues until i is no longer less than 5, at which point the loop terminates. The result is that the values 0 to 4 are logged to the console.

Using the for loop:

```
for (let i = 0; i < 10; i++) {
  console.log(i);
}
```

Explanation: In this code, we are using a for loop to iterate over a range of values. The loop starts by declaring a variable i with the value 0. The loop will then continue to execute as long as i is less than 10. At the end of each iteration, the value of i is incremented by 1. This means that on each iteration, the value of i is logged to the console using the console.log() method. The result will be the numbers 0 through 9 logged to the console.

Using the while loop:

```
let i = 0;
while (i < 10) {
  console.log(i);
  i++;
}
```

Explanation: In this code, we are using a while loop to iterate over a range of values. The loop starts by declaring a variable i with the value 0. The loop will then continue to execute as long as i is less than 10. At the end of each iteration, the value of i is incremented by 1. This means that on each iteration, the value of i is logged to the console using the console.log() method. The result will be the numbers 0 through 9 logged to the console.

For-of Loop:

```
let numbers = [1, 2, 3, 4, 5];
for (let number of numbers) {
  console.log(number);
}
```

Explanation: In this code, we are declaring an array named numbers with 5 elements. We are then using a for-of loop to log each element of the numbers array to the console. The for-of

loop creates a variable named number that takes on the value of each element in the numbers array one at a time, and the code inside the loop logs the value of number to the console.

Array Methods:

```
let numbers = [1, 2, 3, 4, 5];
console.log(numbers.length);
console.log(numbers.slice(1, 3));
```

Explanation: In this code, we are declaring an array named numbers with 5 elements. The first console.log statement logs the length of the array, which is 5. The second console.log statement logs the result of calling the slice method on the numbers array, which returns a new array containing elements from the original array at indices 1 to 2 (the second and third elements).

Object Destructuring:

```
let person = {
  name: "John Doe",
  age: 30,
  isStudent: false
```

```
};
let { name, age } = person;
console.log(name);
console.log(age);
```

Explanation: In this code, we are declaring an object named person with properties name, age, and isStudent. Then, using object destructuring, we are extracting the name and age properties from the person object and assigning them to separate variables with the same names. The final console.log statements log the values of the name and age variables to the console.

Spread Operator:

```
let a = [1, 2, 3];
let b = [4, 5, 6];
let numbers = [...a, ...b];
console.log(numbers);
```

Explanation: In this code, we are declaring two arrays a and b with 3 elements each. We are then using the spread operator to concatenate the two arrays into a new array named numbers. The final console.log statement logs the numbers array to the console, which contains elements from both a and b.

Using the Date object:

```
let now = new Date();
console.log(now);
console.log(now.toLocaleString());
```

Explanation: In this code, we are creating a Date object with the current date and time using the new Date() constructor. The first console.log statement logs the Date object as a string. The second console.log statement logs the same Date object, but formatted as a human-readable string using the toLocaleString() method.

Example of Date object:

```
let date = new Date();
console.log(date);
```

Explanation: In this code, we are using the Date object to get the current date and time. The new Date() constructor creates a new Date object, which represents the current date and time. The result is logged to the console using the console.log() method.

Using the Math object:

```
let number = Math.round(2.5);
console.log(number);
```

Explanation: In this code, we are using the Math object to perform mathematical operations. The Math.round() method takes a number as an argument and returns the rounded value. In this case, the number 2.5 is rounded to 3. The result is logged to the console using the console.log() method.

Example of Math object:

```
console.log(Math.PI);
console.log(Math.ceil(3.14));
console.log(Math.floor(3.14));
```

Explanation: In this code, we are using the Math object to perform mathematical operations. The first console.log statement logs the value of π (Pi) from the Math object. The second console.log statement logs the smallest integer that is greater than or equal to 3.14 using the ceil() method. The third

console.log statement logs the largest integer that is less than or equal to 3.14 using the floor() method.

Using try and catch statements:

```
try {
  let x = y;
  console.log(x);
} catch (error) {
  console.error(error);
}
```

Explanation: In this code, we are using a try and catch statement to handle errors. Within the try block, we are attempting to assign the value of y to a variable x. However, y has not been defined, so this operation will result in a ReferenceError. This error will be caught by the catch block, which logs the error to the console using the console.error() method. This is useful for handling unexpected errors in your code and allowing your program to continue running, rather than crashing.

Try-catch statement Example:

```
try {
  let num = Number("hello");
} catch (error) {
  console.log(error.message);
}
```

Explanation: In this code, we are using the try-catch statement to handle errors. The try block contains code that might throw an error, while the catch block contains code that is executed when an error is thrown. In this case, the Number("hello") expression tries to convert the string "hello" to a number, but since this is not possible, an error is thrown. The error is caught by the catch block and its message property is logged to the console using the console.log() method.

Using the map() method:

```
let numbers = [1, 2, 3, 4, 5];
let doubled = numbers.map(function(num) {
  return num * 2;
});
console.log(doubled);
```

Explanation: In this code, we are using the map() method to transform an array of numbers. The map() method takes a callback function as an argument and applies that function to each element in the array. In this case, the callback function doubles each element in the numbers array. The result is a new array doubled that contains the doubled values. The final console.log statement logs the doubled array to the console.

Using the filter() method:

```
let numbers = [1, 2, 3, 4, 5];
let evens = numbers.filter(function(num) {
  return num % 2 === 0;
});
console.log(evens);
```

Explanation: In this code, we are using the filter() method to select certain elements from an array based on a condition. The filter() method takes a callback function as an argument and applies that function to each element in the array. In this case, the callback function checks if each element in the numbers array is even. The result is a new array evens that contains only the even numbers from the numbers array. The final console.log statement logs the evens array to the console.

Using the reduce() method:

```
let numbers = [1, 2, 3, 4, 5];
let sum = numbers.reduce(function(accumulator,
currentValue) {
  return
let numbers = [1, 2, 3, 4, 5];
let sum = numbers.reduce(function(accumulator,
currentValue) {
  return accumulator + currentValue;
});
console.log(sum);
```

Explanation: In this code, we are using the reduce() method to reduce an array of numbers to a single value. The reduce() method takes a callback function as an argument and applies that function to each element in the array. The first argument to the callback function is an accumulator, which is initialized to the first value in the array. The second argument is the current value being processed. In this case, the callback function adds the current value to the accumulator. The result is a single value sum that is the sum of all the numbers in the numbers array. The final console.log statement logs the sum to the console.

Using the Array.includes() method:

```
let numbers = [1, 2, 3, 4, 5];
let result = numbers.includes(3);
console.log(result);
```

Explanation: In this code, we are using the Array.includes() method to check if an array contains a certain value. The Array.includes() method takes a value as an argument and returns true if the array contains that value and false otherwise. In this case, the value 3 is included in the numbers array, so the result is `true

Using a class to define a blueprint for creating objects:

```
class Person {
  constructor(name, age) {
    this.name = name;
    this.age = age;
  }
  sayHello() {
    console.log(`Hello, my name is ${this.name} and I
am ${this.age} years old.`);
```

```
  }
}
let john = new Person("John", 30);
john.sayHello();
```

Explanation: In this code, we are using a class to define a blueprint for creating objects. A class is a blueprint for creating objects with similar properties and methods. In this case, the Person class has a constructor that takes two arguments, name and age, and sets them as properties of the object being created. The class also has a method sayHello that logs a message to the console using the console.log() method. To create an object using a class, we use the new keyword, such as let john = new Person("John", 30). This creates a new Person object with the properties name set to "John" and age set to 30. To call the sayHello method on the object, we use dot notation, such as john.sayHello(). This causes the message "Hello, my name is John and I am 30 years old." to be logged to the console.

Example: Higher-Order Function

```
function multiplyBy(factor) {
  return function (number) {
    return number * factor;
```

```
  };
}
let double = multiplyBy(2);
console.log(double(5)); // 10
```

Explanation: This example demonstrates the use of a higher-order function. A higher-order function is a function that returns another function. In this example, the multiplyBy function takes a factor as its argument and returns a new function that takes a number as its argument. The returned function calculates and returns the product of the number and factor. We then assign the returned function to the variable double and call it with an argument 5. This will log the result 10 to the console.

Example: Closure

```
function outerFunction() {
  let counter = 0;

  return function innerFunction() {
    counter++;
    console.log(counter);
  };
```

```
}
let counterFunction = outerFunction();
counterFunction(); // 1
counterFunction(); // 2
counterFunction(); // 3
```

Explanation: This example demonstrates the use of a closure. A closure is a function that has access to variables in its outer scope even after the outer function has returned. In this example, the outerFunction returns the innerFunction which logs the value of counter to the console. The variable counter is declared in the outer scope and is accessible to the inner function. We assign the returned function to the variable counterFunction and call it multiple times. This will log the values 1, 2, and 3 to the console, indicating that the inner function still has access to the counter variable even after the outer function has returned.

Example: Destructuring

```
let person = {
  name: "John Doe",
  age: 30,
  address: {
```

```
    street: "123 Main St",
    city: "San Francisco",
    state: "CA",
  },
};
let { name, age, address: { city } } = person;
console.log(name); // "John Doe"
console.log(age); // 30
console.log(city); // "San Francisco"
```

Explanation: This example demonstrates the use of destructuring in JavaScript. Destructuring allows you to extract values from objects and arrays and assign them to separate variables. In this example, we have an object person with three properties: name, age, and address. We use destructuring to extract the values of name, age, and city properties and assign them to separate variables. This will log the values "John Doe", 30, and "San Francisco" to the console.

Example: Promises

```
let promise = new Promise(function (resolve, reject) {
  setTimeout(function () {
    resolve("Success!");
```

```
  }, 1000);
});
promise
  .then(function (result) {
    console.log(result); // "Success!"
  })
  .catch(function (error) {
    console.error(error);
  });
```

Explanation: This example demonstrates the use of Promises in JavaScript. A Promise is a returned object representing the eventual completion or failure of an asynchronous operation. In this example, we create a new Promise using the Promise constructor. The constructor takes a callback function with resolve and reject parameters. We use the setTimeout function to wait for 1 second and then call resolve with a string value of "Success!". The returned Promise object has a then method that takes a success callback function as its argument. This function will be called if the Promise is resolved. The Promise object also has a catch method that takes an error callback function as its argument. This function will be called if the Promise is rejected. In this example, the Promise is resolved, so the then method logs the value "Success!" to the console.

Example: Generators

```
function* generator() {
  yield 1;
  yield 2;
  yield 3;
}
let iterator = generator();
console.log(iterator.next().value); // 1
console.log(iterator.next().value); // 2
console.log(iterator.next().value); // 3
```

Explanation: This example demonstrates the use of generators in JavaScript. A generator is a special type of function that can be paused and resumed multiple times. In this example, we create a generator function using the function* syntax. The generator function uses the yield keyword to return a value each time it is resumed. We create an iterator using the generator function and use the next method to get the next value from the iterator. This will log the values 1, 2, and 3 to the console, indicating that the generator has been resumed and returned the values one by one.

Example: Asynchronous Iteration

```
async function asyncIteration() {
  let array = [1, 2, 3];
  for await (const value of array) {
    console.log(value);
  }
}
asyncIteration();
// Output:
// 1
// 2
// 3
```

Explanation: This example demonstrates the use of asynchronous iteration in JavaScript. Asynchronous iteration allows us to iterate over asynchronous data sources, such as an async function or a Promise, in a synchronous-like manner. In this example, we create an async function that contains a for-await-of loop. The for-await-of loop is used to iterate over the array of values. The loop logs each value to the console. The use of the async keyword ensures that the loop will wait for each iteration to complete before moving on to the next one.

Example: Map and Set

```
let map = new Map();
map.set("key1", "value1");
map.set("key2", "value2");
console.log(map.get("key1")); // "value1"
console.log(map.size); // 2
let set = new Set();
set.add("value1");
set.add("value2");
console.log(set.has("value1")); // true
console.log(set.size); // 2
```

Explanation: This example demonstrates the use of the Map and Set data structures in JavaScript. A Map is an ordered collection of key-value pairs, while a Set is an unordered collection of unique values. In this example, we create a Map and use the set method to add two key-value pairs. We use the get method to retrieve the value associated with a specific key and the size property to get the number of elements in the Map. We also create a Set and use the add method to add two values. We use the has method to check if a value exists in the Set and the size property to get the number of elements in the Set.

Example: WeakMap and WeakSet

```
let weakMap = new WeakMap();
let key = {};
weakMap.set(key, "value");
console.log(weakMap.has(key)); // true
key = null;
console.log(weakMap.has(key)); // false
let weakSet = new WeakSet();
let value = {};
weakSet.add(value);
console.log(weakSet.has(value)); // true
value = null;
console.log(weakSet.has(value)); // false
```

Explanation: This example demonstrates the use of the WeakMap and WeakSet data structures in JavaScript. A WeakMap is a collection of key-value pairs where the keys are objects and the values can be any value. A WeakSet is a collection of objects. Unlike Map and Set, the entries in a WeakMap or WeakSet do not prevent the objects from being garbage collected. In this example, we create a WeakMap and use the set method to add a key-value pair. We use the has method to check if the key exists in the WeakMap. We then set the key to null, which makes it eligible for garbage collection. We check the has method again

and it returns false, as the key has been garbage collected. We also create a WeakSet and use the add method to add a value. We use the has method to check if the value exists in the WeakSet. We then set the value to null, which makes it eligible for garbage collection. We check the has method again and it returns false, as the value has been garbage collected.

Example: Object Destructuring

```
let obj = { x: 1, y: 2, z: 3 };
let { x, y, z } = obj;
console.log(x, y, z); // 1 2 3
```

Explanation: This example demonstrates the use of object destructuring in JavaScript. Object destructuring allows us to extract values from an object and assign them to variables with the same name as the object properties. In this example, we create an object with properties x, y, and z. We then use destructuring to extract the values of these properties and assign them to variables with the same name. The destructured variables x, y, and z now hold the values of the respective properties.

Example: Class and Inheritance

```
class Shape {
  constructor(width, height) {
    this.width = width;
    this.height = height;
  }
  getArea() {
    return this.width * this.height;
  }
}
class Rectangle extends Shape {
  constructor(width, height) {
    super(width, height);
  }
}
let rectangle = new Rectangle(10, 20);
console.log(rectangle.getArea()); // 200
```

Explanation: This example demonstrates the use of classes and inheritance in JavaScript. A class is a blueprint for creating objects with similar properties and methods. In this example, we create a Shape class with a constructor method that takes width and height as arguments and sets them as properties of the object. We also add a getArea method that returns the product of

width and height. We then create a Rectangle class that extends the Shape class. The Rectangle class has its own constructor method that calls the super method to pass the width and height arguments to the Shape class. We create an instance of the Rectangle class and use the getArea method to get the area of the rectangle.