Assignment 2 JavaScript basics.

* Question 2:

Rewrite the following code using a ternary operator:

```
let result;
if (score >= 80) {
    result = "Pass";
} else {
    result = "Fail";
}
```

Answer:

```
// Question 1:
let result1;
let score = 80;

result1 = score >= 80 ? "True" : "False";
console.log('Is score greater than and equal to 80? ' +
result1);
```

* Question 2:

How does the optional chaining operator (?.) work, and how can it be used to access nested properties of an object?

It has three operands or elements. The first element is a conditional, where the value is used to check if it is first true then automatically assigns it to false.

* Question 3:

Compare the for...in loop and the for...of loop in terms of their use cases and the

types of values they iterate over.

For...in loop is used for running a specific task repeatedly on each element present inside a string. Similarly, For...of loop performs the same task but for each element present inside an array.

* Question 4:

Define a function calculateAverage that takes an array of numbers as an argument and returns the average value.

```
// Question 4:

function calculateAverage(numbers) {

  // Check if the array is not empty

  if (numbers.length === 0) {

    return null; // Return null for an empty array (you can
choose a different approach based on your requirements)

  }


  // Calculate the sum of numbers

  const sum = numbers.reduce((accumulator, currentNumber) =>
accumulator + currentNumber, 0);


  // Calculate the average

  const average = sum / numbers.length;
  return average;
}


const numbersArray = [10, 20, 30, 40, 50];
const result = calculateAverage(numbersArray);


console.log("Average:", result);
```

* Question 5:

Explain the concept of "closures" in JavaScript and provide an example of their practical use.

These are the functions that refer to independent variables. Meaning that even if the functions have been declared and returned, a variable can still access it or where another function can access variables outside its scope.

```
function numberGenerator() {
  // Local “free” variable that ends up within the
closure
  var num = 1;
  function checkNumber() {
    console.log(num);
  }
  num++;
  return checkNumber;
}

var number = numberGenerator();
number(); // 2
```

* Question 6:

Create an object named student with properties name, age, and grades. Add a method calculateAverage that calculates the average of the grades.

```
// Question 6:
const student = {
  name: "John",
  age: 20,
  grades: [85, 90, 78, 95, 88],

  calculateAverage: function () {
    // Check if the grades array is not empty
    if (student.grades.length === 0) {
      return null;
    }
    // Calculate the sum of grades

    const sum = student.grades.reduce((acc, curr) => acc +
curr, 0);


    // Calculate the average

    const average = sum / student.grades.length;


    return average;

  }

};


const averageGrade = student.calculateAverage();


console.log(`${student.name}'s Average Grade:`, averageGrade);
```

* Question 7:

How can you clone an object in JavaScript and also give one example each deep copy, shallow copy, and reference copy.

You can clone an object using the spread operator (...), Object.assign() method, and JSON.stringify() and JSON.parse() methods.

Shallow copy:

```
// Shallow copy example
let originalObject = { a: 1, b: { c: 2 } };

let shallowCopy = Object.assign({}, originalObject);

console.log(shallowCopy); // { a: 1, b: { c: 2 } }
console.log(originalObject === shallowCopy); // false
console.log(originalObject.b === shallowCopy.b); // true (reference copied)
```

Deep copy:

```
// Deep copy example using JSON
let originalObject = { a: 1, b: { c: 2 } };

let deepCopy = JSON.parse(JSON.stringify(originalObject));

console.log(deepCopy); // { a: 1, b: { c: 2 } }
console.log(originalObject === deepCopy); // false
console.log(originalObject.b === deepCopy.b); // false (independent copy)
```

Reference copy:

```
// Reference copy example
let originalObject = { a: 1, b: { c: 2 } };

let referenceCopy = originalObject;

console.log(referenceCopy); // { a: 1, b: { c: 2 } }
console.log(originalObject === referenceCopy); // true
console.log(originalObject.b === referenceCopy.b); //
true (reference copied)
```

* Question 8:

Write a loop that iterates over an array of numbers and logs whether each number is even or odd, using a ternary operator.

```
// Question 8
const numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9];

for (const number of numbers) {
  const isEven = number % 2 === 0;
  const message = isEven ? "Even" : "Odd";

  console.log(number + " is " + message);
}
```

* Question 9:

Describe the differences between the for loop, while loop, and do...while loop in JavaScript. When might you use each?

In For loop, you can declare and initialize all three elements needed for running a loop statement, such as a looping variable (let i;), condition (i <= 20;), and an increment or decrement to change each loop cycle, it can have both pre and post increment (++i, i++, --i, i--). All these are declared in an order within the parameters (), and may or may not be omitted such as:

Original syntax:

```
for (let i = 0; i <= arr.length; i++){}
```

Modified syntax:

```
let i = 0;
```

```
for (; i<= arr.length; i++){}
```

Here the first element, (i) is omitted because it is already initialized before the For loop. However, it is important to write (;) in the parameters as that the value of (i) is recognized by the For loop and runs smoothly.

In While loop, you can only initialize and declare a condition inside the parameters. Whereas, the looping variable is used outside the loop and the incrementation or decrementation is performed inside.

Original syntax:

```
let i = 0;

while(i < 20){

    i++;

}
```

In Do while loop, you first write the task, which will run as a loop then write the condition. Hence it will run the loop once even if the condition is false, then exit the loop.

Original syntax:

```
let i = 1;

do {

    console.log("Hi there!");

}

while(i < 1)
```

Do while loop can be used for checking if there is an error in the condition by running a false or wrong value.

* Question 10:

Provide an example of using optional chaining within a loop to access a potentially missing property of an object.

```
// Question 10

const students = [

  { name: 'Alice', grades: { math: 90, english: 85 } },

  { name: 'Bob', grades: { math: 88 } },

  { name: 'Charlie' } // Charlie has no grades property

];


for (const student of students) {

  // Using optional chaining to access the math grade property

  const mathGrade = student?.grades?.math;


  // Checking if the property is defined before logging

  if (mathGrade !== undefined) {

    console.log(student.name +  "'s " + 'math grade: ' +
mathGrade);

  } else {

    console.log(student.name + ' has no math grade
available.');

  }

}
```

* Question 11:

Write a for...in loop that iterates over the properties of an object and logs each property name and value.

```
// Question 11
const person = {
  name: 'John',
  age: 30,
  gender: 'male'
};

for (const key in person) {
  if (person.hasOwnProperty(key)) {
    const value = person[key];
    console.log('Property: ' + key, ' Value: ' + value);
  }
}
```

* Question 12:

Explain the use of the break and continue statements within loops. Provide scenarios where each might be used.

Break statement is used to exit a code block inside a case in a switch case loop. Whereas, the continue is used to carry on with by executing the remaining switch case code present.

When a condition is running infinitely or if a person doesn't have the permission or meet the criteria, then break is used.

When an authorized user wants to view all the data stored in the cases, then continue is used.

- Question 13:

Write a function calculateTax that calculates and returns the tax amount based on a given income. Use a ternary operator to determine the tax rate.

```
// Question 13:

function calculateTax(income) {

  const taxRate = income <= 50000 ? 0.1 : income <= 100000 ? 0.15 : 0.2;

  return income * taxRate;

}


const income1 = 45000;

const tax1 = calculateTax(income1);

console.log('Tax for ' + income1 + ' : ' + tax1);


const income2 = 75000;

const tax2 = calculateTax(income2);

console.log('Tax for ' + income2 + ' : ' + tax2);
```

- Question 14:

Create an object car with properties make, model, and a method startEngine that logs a message. Instantiate the object and call the method.

```
const car = {
  make: 'Toyota',
  model: 'Camry',
  startEngine: function() {
    console.log('Engine started! Vroom vroom!');
  }
};

car.startEngine();
```

* Question 15:

Explain the differences between regular functions and arrow functions in terms of scope, this binding, and their use as methods.

A regular function needs parameters () and curly braces {} to run the code block, whereas the arrow function only needs parameters. A regular function needs to have its name or identifier, whereas the arrow function can have an anonymous name.