Conditions and Loops

Assignment Questions

Q1. What are conditional statements? Explain conditional statements with syntax and examples.

Answer: In JavaScript, conditional statements are used to execute different blocks of code based on whether a specified condition is true or false. They help control the flow of the program based on certain conditions.

Types of Conditional Statements in JavaScript:

if statement: Executes a block of code if the condition is true.
Syntax:

```
if (condition) {
    // code to execute if condition is true
}
```

Example:

```
let age = 18;
if (age >= 18) {
    console.log("You are an adult.");
}
```

else statement: Executes a block of code if the condition is false. This must be used after an `if` statement.
Syntax:

```
if (condition) {
    // code to execute if condition is true
} else {
    // code to execute if condition is false
}
```

Example:

```
let age = 16;
if (age >= 18) {
    console.log("You are an adult.");
} else {
    console.log("You are a minor.");
}
```

else if statement: Allows checking multiple conditions. It's used after an `if` statement to check additional conditions if the first one is false.
Syntax:

```
if (condition1) {
    // code to execute if condition1 is true
} else if (condition2) {
    // code to execute if condition2 is true
} else {
    // code to execute if all conditions are false
}
```

Example:

```
let age = 20;
if (age >= 65) {
    console.log("You are a senior citizen.");
} else if (age >= 18) {
    console.log("You are an adult.");
} else {
    console.log("You are a minor.");
}
```

switch statement: Allows testing multiple conditions for a single variable. It's an alternative to multiple `if-else` statements when checking many conditions based on the same expression.

Syntax:

```
switch (expression) {
    case value1:
        // code to execute if expression equals value1
        break;
    case value2:
        // code to execute if expression equals value2
        break;
    default:
        // code to execute if no case matches
}
```

Example:

```
let day = 3;
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");
}
```

Summary:

- if: Executes a block of code if the condition is true.
- else: Executes a block of code if the condition is false.
- else if: Checks additional conditions if the previous ones are false.
- switch: Tests a variable against multiple values and executes code for the matching case.

Conditional statements in JavaScript allow you to make decisions in your programs and control the flow of execution.

Q2. Write a program that grades students based on their marks

If greater than 90 then A Grad If
between 70 and 90 then a B grad
If between 50 and 70 then a C
grad Below 50 then an F grade

Answer:

```
// Input: Get the student's marks
let marks = parseFloat(prompt("Enter the student's marks:"));

// Conditional statements to grade based on marks
let grade;
if (marks > 90) {
    grade = 'A';
} else if (marks >= 70) {
    grade = 'B';
} else if (marks >= 50) {
    grade = 'C';
} else {
    grade = 'F';
}
```

```
// Output: Display the grade
console.log(`The student's grade is: ${grade}`);
```

Q3. What are loops, and what do we need them? Explain different types of loops with their syntax and examples.

Answer: Loops in programming allow us to repeatedly execute a block of code as long as a specified condition is true. Loops are useful when we want to perform repetitive tasks without writing the same code multiple times. They help save time and reduce errors in programs.

Why do we need loops?

- Efficiency: Loops help you execute repetitive tasks automatically.
- Flexibility: With loops, you can iterate over collections of data (arrays, objects, etc.).
- Readability: Using loops makes the code cleaner and more concise.

Types of Loops in JavaScript:

1. for loop

The `for` loop is used when you know how many times you want to execute a statement or a block of code.

Syntax:

```
for (initialization; condition; increment/decrement) {
    // code to be executed
}
```

Example:

```
for (let i = 1; i <= 5; i++) {
    console.log(i);
}
```

2. while loop

The `while` loop runs as long as the specified condition is true. It's useful when you don't know beforehand how many times the loop should run.

Syntax:

```
while (condition) {
    // code to be executed
}
```

Example:

```
let i = 1;
while (i <= 5) {
    console.log(i);
    i++;  // Increment i
}
```

3. do...while loop

The `do...while` loop is similar to the `while` loop, except that it always executes the block of code at least once before checking the condition.

Syntax:

```
do {
    // code to be executed
} while (condition);
```

Example:

```
let i = 1;
do {
    console.log(i);
    i++;
} while (i <= 5);
```

4. for...in loop

The `for...in` loop is used to iterate over the properties (keys) of an object.

Syntax:

```
for (let key in object) {
    // code to be executed
}
```

Example:

```
const person = { name: "Alice", age: 25, city: "New York" };
for (let key in person) {
    console.log(key + ": " + person[key]);
}
```

5. for...of loop

The `for...of` loop is used to iterate over the values of an iterable (like arrays or strings).

Syntax:

```
for (let value of iterable) {
    // code to be executed
}
```

Example:

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

Q4. Generate numbers between any 2 given numbers. Ex

const num1 = 10
const num2 = 25;
Output: 11, 12, 13,
...., 25

Answer: To generate numbers between two given numbers in JavaScript, you can use a `for` loop to iterate through the range and print each number. Here's an example program:

```
const num1 = 10;
const num2 = 25;

// Generate numbers between num1 and num2
for (let i = num1 + 1; i < num2; i++) {
    console.log(i);
}
```

Explanation:

- num1 and num2 are the two given numbers.
- The loop starts at `num1 + 1` (i.e., one number greater than `num1`) and goes up to `num2 - 1`

(i.e., one number less than `num2`).

- The loop prints each number in between `num1` and `num2`.

Example Output (for num1 = 10, num2 = 25):

```
12
13
14
15
16
17
18
19
20
21
22
23
24
```

This approach will work for any two given numbers and will print all numbers between them, excluding `num1` and `num2`. If you want to include `num2`, you can adjust the loop condition accordingly.

Q5. Use the while loop to print numbers from 1 to 25 in ascending and descending order.
Answer: Here's how you can use a `while` loop to print numbers from 1 to 25 in both ascending and descending order in JavaScript:

Ascending Order (1 to 25):

```
let i = 1;
while (i <= 25) {
    console.log(i);
    i++;  // Increment by 1
}
```

Descending Order (25 to 1):

```
let j = 25;
while (j >= 1) {
    console.log(j);
    j--;  // Decrement by 1
}
```

Explanation:

- Ascending order: The loop starts with `i = 1` and continues as long as `i` is less than or equal to 25. After each iteration, `i` is incremented (`i++`).
- Descending order: The loop starts with `j = 25` and continues as long as `j` is greater than or equal to 1. After each iteration, `j` is decremented (`j--`).