Control Flow (If-Else, Switch)

Question 1: What is control flow in JavaScript? Explain how if-else statements work with an example.

Ans.

Control flow in JavaScript refers to the order in which statements are executed in a program. By default, JavaScript executes code sequentially from top to bottom. However, control flow statements allow altering this default order based on conditions, loops, or function calls, enabling programs to make decisions and perform repetitive actions

The if-else statement is a fundamental conditional control flow structure that allows a program to execute different blocks of code based on whether a specified condition is true or false.

How the if-else statement works:

1. if condition: The if statement evaluates a given condition.
2. Truthiness: If the condition evaluates to true (or a "truthy" value), the code block immediately following the if statement is executed
3. else block (optional): If the condition evaluates to false (or a "falsy" value), and an else block is present, the code block within the else statement is executed instead.
4. Continuation: After either the if block or the else block (if present) is executed, the program continues with the statements following the entire if-else structure.

Example :

```
let temperature = 28;

if (temperature > 25) {

console.log("It's a hot day!");

}
```

```
else {
console.log("It's a pleasant day.");
}
```

Explanation of the example:

1. The variable temperature is initialised with the value 28.
2. The if statement checks the condition temperature > 25.
3. Since 28 is indeed greater than 25, the condition evaluates to true.
4. Consequently, the code inside the if block, console.log("It's a hot day!");, is executed, and "It's a hot day!" is printed to the console.
5. The else block is skipped because the if condition was true.

Question 2: Describe how switch statements work in JavaScript. When should you use a switch statement instead of if-else?

Ans.

JavaScript, a switch statement provides a way to control the flow of execution based on the value of a single expression. It evaluates an expression once and then compares its value against multiple case clauses.

- **How switch statements work:**
  1. Expression Evaluation : The switch statement begins by evaluating the expression provided within its parentheses.
  2. Case Matching : The result of this expression is then strictly compared (using ===) with the value of each case clause in sequential order.
  3. Code Execution : If a match is found, the code block associated with that case is executed.
  4. break Statement: Crucially, a break statement is typically included at the end of each case block to exit the switch statement.

Without break, execution will "fall through" to subsequent case blocks, even if their values do not match the expression, until a break or the end of the switch statement is encountered.

5. default Clause: An optional default clause can be included. Its code block is executed if none of the case clauses match the expression's value.

Example :

```
let day = "Monday";
switch (day) {
 case "Monday":
   console.log("It's the start of the week!");
   break;
case "Friday":
   console.log("Almost the weekend!");
   break;
 default:
     console.log("Just another day.");
}
```

<u>When should you use a switch statement instead of if-else :</u>

1. Handling Multiple Fixed Values: switch statements are ideal when you need to perform different actions based on a single variable or expression having a set of distinct, fixed values (e.g., numbers, strings, enums). This often leads to more readable and organized code compared to a long chain of if-else if-else statements.

2. Improved Readability: For scenarios involving numerous specific conditions, a switch statement can enhance code clarity and make it easier to understand the different execution paths.
3. State Management: switch statements are frequently used in state management patterns, where the program's behavior depends on its current state, represented by a single variable.

<u>When if-else might be preferred :</u>

1. Complex Conditions: When dealing with complex logical conditions involving relational operators (e.g., x > 10 && y < 20) or multiple variables, if-else statements are more suitable as switch primarily checks for strict equality.
2. Range-Based Conditions: If your conditions involve checking if a value falls within a certain range (e.g., age >= 18 && age <= 65), if-else provides a more direct and expressive way to handle these.
