Lesson:

Switch Case

PW SKILLS

Topics Covered

1. Introduction to switch case.
2. Switch Syntax.
3. Switch with return statement
4. What happens when we skip the break keyword?
5. Handling multiple cases having same logiç
6. When to use switch statements or if/else statements?
7. Example: Weekday
8. When to use switch statements or if/else statements.

Introduction to switch case.

Since we have been talking about conditionals from past lectures. Let’s look at a real-life scenario, assume it's lunchtime and you walk to your favorite restaurant. The attendant offers you the menu. On the menu are different delicious items made for special people like you. You go through the menu and choose one or more meals from the menu and have yourself a good lunch. That is what **switch** statements help us do in JavaScript

Switch Syntax

Let’s look at the syntax of the switch statement

```javascript
switch (expression) {
  case value1:
    // code to execute;
    break;
  case value2:


    // code to execute;
    break;
 case value3:
    // code to execute;
    break;
  default:
  // default code;
}
```

- switch keyword defines a switch block.
- case keyword defines different case blocks, which will be executed only when case value matches with expression value. Case values can be a number, string or boolean etc.
- break keyword is used at the end of every case block to terminate switch case evaluation further.
- default keyword defines a default case block. When no case value is matched then the default block will be executed. It is not mandatory to include it.

Let’s compare the switch syntax with the if statement

```JavaScript
switch (expression) {
   case value1: // if (expression === value1 then execute this
block)
   // code to execute;
   break;

   case value2: // if (expression === value2 then execute this
block)
   // code to execute;


   break;

  case value3: // if (expression === value3 then execute this
block)
  // code to execute;
  break;

  default: // if (expression === none of the previous values
matched execute this block)
 // default code;
}
```

Example: Weekday

Let’s now implement a switch statement considering an example.

We represent our weekdays in both number notation and text notation like 1 for Sunday, 2 for Monday, and so on.

Let’s take the number notation as input and provide text notation as output using the switch statement

In this case, the only condition is that the number notation of the day must be between 1 to 7. If any other input is given then it will be considered invalid input.

Let’s now implement a switch statement considering an example.

```javascript
var day = 5;

switch (day) {
  case 1: // if (day === 1) then execute this block)
    console.log("Sunday");
    break;


  case 2: // if (day === 2) then execute this block)
    console.log("Monday");
    break;

  case 3: // if (day === 3) then execute this block)
    console.log("Tuesday");
    break;

  case 4: // if (day === 4) then execute this block)
    console.log("Wednesday");
    break;

  case 5: // if (day === 5) then execute this block)
    console.log("Thursday");
    break;

  case 6: // if (day === 6) then execute this block)
    console.log("Friday");
    break;

  case 7: // if (day === 7) then execute this block)
    console.log("Saturday");
    break;

  default: // if (expression === none of the previous conditions
then execute this block)
    console.log("Day doesn't exist");
}

// Output: Thursday
```

This code will output **Thursday** to the console because the value of the variable **day** is **5**.

As we have passed the day to the switch statement, the day matches case **5** in the switch statement. When this case is executed, it will run the code block associated with it, which is console.log(**Thursday**).

The break statement at the end of the case ensures that the code execution exits the switch statement after the matching case has been executed, so the code in the other cases and the default block will not be executed.

In the above example, we have used numbers as case values, 1,2,3,4, but be careful if you use “1” as your case value it will be treated as a different case because “1” is a string and 1 is a number datatype and switch cases use strict comparison (===). The values must be of the same type to match. A strict comparison can only be true if the operands are of the same type.

```
JavaScript
console.log(1 === '1') // false
```

Switch with return statement

We can use the **return** keyword instead of **break** for every case. It will be helpful, if we are using switch statements inside

the **function**, because it will serve two purposes, one to come out of the **switch** block (like a **break**) and at the same time it returns the result from the function.

Lets see one example using return statement,

```
JavaScript
let grade = 'B';

function getValue(grade){
   switch (grade) {
    case 'A':
       return "Excellent";
    case 'B':
       return "Average"
    case 'C':
       return "Below than average";
    default:
       return "No Grade";
   }
}


console.log(getValue(grade)); // Average
```

In the above example, the getValue function returns keywords “Excellent”, “Average”, “Below Average” based on grades A, B, C respectively. It uses switch statements and instead of the break keyword in every case we are returning value using the return keyword.

What happens when we skip the break keyword?

In JavaScript, when you skip the break keyword in a switch statement, the program will continue executing the statements in the following case(s) without any further

evaluation of the case conditions. This behavior is known as "fall-through."

Example

```javascript
let color = "red";

switch (color) {
  case "red":
    console.log("The color is red.");
  case "blue":
    console.log("The color is blue.");
    break;
  case "green":
    console.log("The color is green.");
     break;
  default:
    console.log("The color is unknown.");
}

//Output
/*
In this case, if the value of color is "red," but the program
will print:

The color is red.
The color is blue.
*/
```

Since the break keyword is not used after the first case, the program will continue executing the statements in the subsequent cases. This is because JavaScript does not

automatically exit the switch statement after a matching case unless a break statement is encountered.

Handling multiple cases having same logiç

Sometimes we have situations where, we have same logic for different cases, here is an example,

Suppose in an organization, salary is incremented annually based on their role,

- If the role is an employee, manager or HR increment will be 5%.
- If the role is CEO, CIO, CTO increment will be 10%.

Lets see how we can implement this using switch,

```JavaScript
let role = "CEO";
let salary = 100000;

switch (role) {
  case "employee":
  case "hr":
  case "manager":
     salary += 0.05*salary;
     break;
  case "CEO":
  case "CIO":
  case "CTO":
     salary += 0.1*salary;
     break;
  default:
    console.log("Unknown Role");
}


console.log(salary);
//Output - 110000
```

When to use switch statements or if/else statements?

- if/else conditional branches are great for variable conditions that result in a Boolean, whereas switch statements are great for fixed data values.
- In a situation where more than one choice is preferred, the switch is a better choice than an if/else statement.
- Considering the speed of execution it is advised if the number of cases is more than 5 use a switch, otherwise, you may use if-else statements.
- Switch is more readable and looks much cleaner when you have to combine cases.