# 4

# Logic Statements

Up to this point, our code has been rather static. It will do the same thing every time we execute it. In this chapter, that is all going to change. We will be dealing with logical statements. Logical statements allow us to make multiple paths in our code. Depending on the outcome of a certain expression, we will follow one code path or another.

There are different logic statements, and we will go over them in this chapter. We will start with `if` and `if else` statements. After that, we will be dealing with the ternary operator, and the final one we will be dealing with is the `switch` statement.

Along the way, we will cover the following topics:

- if and if else statements
- else if statements
- Conditional ternary operators
- switch statements

Note: exercise, project and self-check quiz answers can be found in the *Appendix*.

# if and if else statements

We can make decisions in our code using if and if else statements. It is very much like this template:

*if *some condition is true*, then *a certain action will happen*, else *another action will happen**

For example, *if* it is raining then, I will take my umbrella, *else* I will leave my umbrella at home. It is not that much different in code:

```
let rain = true;

if(rain){
  console.log("** Taking my umbrella when I need to go outside **");
} else {
  console.log("** I can leave my umbrella at home **");
}
```

In this case, the value of `rain` is `true`. And therefore, it will log to the console:

```
** Taking my umbrella when I need to go outside **
```

But let's first take a step back and look at the syntax. We start with the word "if." After this, we get something within parentheses. Whatever is between these parantheses will be translated to a Boolean. If the value of this Boolean is `true`, it will execute the block of code associated with `if`. You can recognize this block by the curly braces.

The next block is optional; it is an `else` block. It starts with the word "else" and is only executed in case of the Boolean having the value `false`. If there is no `else` block and the condition evaluates to false, the program will just skip ahead to the code underneath the `if`.

Only one of these two blocks will be executed; the `if` block when the expression is true, and the `else` block when the expression is false:

```
if(expression) {
  // code associated with the if block
  // will only be executed if the expression is true
} else {
  // code associated with the else block
  // we don't need an else block, it is optional
  // this code will only be executed if the expression is false
}
```

Here is another example. If the age is below 18, log to the console that access is denied, otherwise log to the console that the person is allowed to come in:

```
if(age < 18) {
  console.log("We're very sorry, but you can't get in under 18");
} else {
  console.log("Welcome!");
}
```

There is a common coding mistake related to `if` statements. I have made it in the following code snippet. Can you see what this code does?

```
let hobby = "dancing";

if(hobby == "coding"){
  console.log("** I love coding too! **");
} else {
  console.log("** Can you teach me that? **");
}
```

It will log the following:

```
** I love coding too! **
```

That might surprise you. The problem here is the single equal sign in the `if` statement. Instead of evaluating the condition, it is assigning `coding` to `hobby`. And then it is converting `coding` to a Boolean, and since it is not an empty string, it will become true, so the if block will be executed. So, always remember to use the double equal sign in this case.

Let's test our knowledge with a practice exercise.

# Practice exercise 4.1

1. Create a variable with a Boolean value.
2. Output the value of the variable to the console.
3. Check whether the variable is true and if so, output a message to the console, using the following syntax:

```
if(myVariable){
//action
}
```

4. Add another if statement with an `!` in front of the variable to check whether the condition is *not* true, and create a message that will be printed to the console in that instance. You should have two if statements, one with an `!` and the other without. You could also use an if and an else statement instead—experiment!
5. Change the variable to the opposite to see how the result changes.

# else if statements

A variation of the `if` statement is an `if` statement with multiple `else if` blocks. This can be more efficient in certain situations because you are always only going to execute one or zero blocks. If you have many `if else` statements underneath one another, they are going to be evaluated and possibly executed even though one of the ones above already had a condition evaluate to true and proceeded to execute the associated code block.

Here is the written template:

*If *a value falls into a certain category*, then *a certain action will happen*, else if *the value falls into a different category than the previous statement*, then *a certain action will happen*, else if *the value falls into a different category than either of the previous brackets*, then *a certain action will happen**

For example, take this statement, to determine what the ticket price should be. If a person is younger than 3, then access is free, else if a person is older than 3 and younger than 12, then access is 5 dollars, else if a person is older than 12 and younger than 65, then access is 10 dollars, else if a person is 65 or older, then access is 7 dollars:

```
let age = 10;
let cost = 0;
let message;
if (age < 3) {
    cost = 0;
    message = "Access is free under three.";
} else if (age >= 3 && age < 12) {
    cost = 5;
    message ="With the child discount, the fee is 5 dollars";
} else if (age >= 12 && age < 65) {
    cost = 10;
    message ="A regular ticket costs 10 dollars.";
```

```
} else {
    cost = 7;
    message ="A ticket is 7 dollars.";
}

console.log(message);
console.log("Your Total cost "+cost);
```

Chances are that you will think the code is easier to read than the written template. In that case, nicely done! You are really starting to think like a JavaScript developer already.

The code gets executed top to bottom, and only one of the blocks will be executed. As soon as a true expression is encountered, the other ones will be ignored. This is why we can also write our sample like this:

```
if(age < 3){
  console.log("Access is free under three.");
} else if(age < 12) {
  console.log("With the child discount, the fee is 5 dollars");
} else if(age < 65) {
  console.log("A regular ticket costs 10 dollars.");
} else if(age >= 65) {
  console.log("A ticket is 7 dollars.");
}
```

# Practice exercise 4.2

1. Create a prompt to ask the user's age
2. Convert the response from the prompt to a number
3. Declare a message variable that you will use to hold the console message for the user
4. If the input age is equal to or greater than 21, set the `message` variable to confirm entry to a venue and the ability to purchase alcohol
5. If the input age is equal to or greater than 19, set the `message` variable to confirm entry to the venue but deny the purchase of alcohol
6. Provide a default else statement to set the `message` variable to deny entry if none are true
7. Output the response `message` variable to the console

# Conditional ternary operators

We did not actually discuss this very important operator in our section on operators in *Chapter 2, JavaScript Essentials*. This is because it helps to understand the if else statement first. Remember that we had a unary operator that was called a unary operator because it only had one operand? This is why our ternary operator has its name; it has three operands. Here is its template:

```
operand1 ? operand2 : operand3;
```

`operand1` is the expression that is to be evaluated. If the value of the expression is `true`, `operand2` gets executed. If the value of the expression is `false`, `operand3` gets executed. You can read the question mark as "then" and the colon as "else" here:

```
expression ? statement for true : statement associated with false;
```

The template for saying it in your head should be:

*if *operand1*, then *operand2*, else *operand3**

Let's have a look at a few examples:

```
let access = age < 18 ? "denied" : "allowed";
```

This little code snippet will assign a value to access. *If* age is lower than 18, *then* it will assign the value `denied`, *else* it will assign the value `allowed`. You can also specify an action in a ternary statement, like this:

```
age < 18 ? console.log("denied") : console.log("allowed");
```

This syntax can be confusing at first. The template of what to say in your head while reading it can really come to the rescue here. You can only use these ternary operators for very short actions, so it's best practice to use the ternary operator in these instances as it makes code easier to read. However, if the logic contains multiple comparison arguments, you'll have to use the regular if-else.

# Practice exercise 4.3

1. Create a Boolean value for an ID variable
2. Using a ternary operator, create a message variable that will check whether their ID is valid and either allow a person into a venue or not
3. Output the response to the console

# switch statements

If else statements are great for evaluating Boolean conditions. There are many things you can do with them, but in some cases, it is better to replace them with a switch statement. This is especially the case when evaluating more than four or five values.

We are going to see how switch statements can help us and what they look like. First, have a look at this if else statement:

```
if(activity === "Get up") {
  console.log("It is 6:30AM");
} else if(activity === "Breakfast") {
  console.log("It is 7:00AM");
} else if(activity === "Drive to work") {
  console.log("It is 8:00AM");
} else if(activity === "Lunch") {
  console.log("It is 12.00PM");
} else if(activity === "Drive home") {
  console.log("It is 5:00PM")
} else if(activity === "Dinner") {
  console.log("It is 6:30PM");
}
```

It is determining what the time is based on what we are doing. It would be better to implement this using a switch statement. The syntax of a switch statement looks like this:

```
switch(expression) {
  case value1:
    // code to be executed
    break;
  case value2:
    // code to be executed
    break;
  case value-n:
    // code to be executed
    break;
}
```

You can read it in your head as follows: If the expression equals `value1`, do whatever code is specified for that case. If the expression equals `value2`, do whatever code is specified for that case, and so on.

Here is how we can rewrite our long if else statement in a much cleaner manner using a switch statement:

```
switch(activity) {
  case "Get up":
    console.log("It is 6:30AM");
    break;
  case "Breakfast":
    console.log("It is 7:00AM");
    break;
  case "Drive to work":
    console.log("It is 8:00AM");
    break;
  case "Lunch":
    console.log("It is 12:00PM");
    break;
  case "Drive home":
    console.log("It is 5:00PM");
    break;
  case "Dinner":
    console.log("It is 6:30PM");
    break;
}
```

If our activity has the value `Lunch` it will output the following to the console:

```
It is 12:00PM
```

What's up with all these breaks, you may be wondering? If you do not use the command `break` at the end of a case, it will execute the next case as well. This will be done from the case where it has a match, until the end of the switch statement or until we encounter a `break` statement. This is what the output of our switch statement would be without breaks for the `Lunch` activity:

```
It is 12:00PM
It is 5:00PM
It is 6:30PM
```

One last side note. `switch` uses strict type checking (the triple equals strategy) to determine equality, which checks for both a value and a data type.

# The default case

There is one part of switch that we have not worked with yet, and that is a special case label, namely, `default`. This works a lot like the else part of an if else statement. If it does not find a match with any of the cases and a default case is present, then it will execute the code associated with the default case. Here is the template of a switch statement with a default case:

```
switch(expression) {
  case value1:
    // code to be executed
    break;
  case value2:
    // code to be executed
    break;
  case value-n:
    // code to be executed
    break;
  default:
    // code to be executed when no cases match
    break;
}
```

The convention is to have the default case as the last case in the switch statement, but the code will work just fine when it is in the middle or the first case. However, we recommend you stick to the conventions and have it as a last case, since that is what other developers (and probably your future self) will expect when dealing with your code later.

Let's say our long `if` statement has an `else` statement associated with it that looks like this:

```
if(…) {
  // omitted to avoid making this unnecessarily long
} else {
  console.log("I cannot determine the current time.");
}
```

The switch statement would then look like this:

```
switch(activity) {
  case "Get up":
    console.log("It is 6:30AM");
```

```
    break;
  case "Breakfast":
    console.log("It is 7:00AM");
    break;
  case "Drive to work":
    console.log("It is 8:00AM");
    break;
  case "Lunch":
    console.log("It is 12:00PM");
    break;
  case "Drive home":
    console.log("It is 5:00PM");
    break;
  case "Dinner":
    console.log("It is 6:30PM");
    break;
  default:
    console.log("I cannot determine the current time.");
    break;
}
```

If the value of the activity was to be something that is not specified as a case, for example, "Watch Netflix," it would log the following to the console:

```
I cannot determine the current time.
```

# Practice exercise 4.4

As discussed in *Chapter 1, Getting Started with JavaScript*, the JavaScript function `Math.random()` will return a random number in the range of 0 to less than 1, including 0 but not 1. You can then scale it to the desired range by multiplying the result and using `Math.floor()` to round it down to the nearest whole number; for example, to generate a random number between 0 and 9:

```
// random number between 0 and 1
let randomNumber = Math.random();
// multiply by 10 to obtain a number between 0 and 10
randomNumber = randomNumber * 10;
// removes digits past decimal place to provide a whole number
RandomNumber = Math.floor(randomNumber);
```

In this exercise, we'll create a Magic 8-Ball random answer generator:

1. Start by setting a variable that gets a random value assigned to it. The value is assigned by generating a random number 0-5, for 6 possible results. You can increase this number as you add more results.
2. Create a prompt that can get a string value input from a user that you can repeat back in the final output.
3. Create 6 responses using the switch statement, each assigned to a different value from the random number generator.
4. Create a variable to hold the end response, which should be a sentence printed for the user. You can assign different string values for each case, assigning new values depending on the results from the random value.
5. Output the user's original question, plus the randomly selected case response, to the console after the user enters their question.

# Combining cases

Sometimes, you would want to do the exact same thing for multiple cases. In an if statement, you would have to specify all the different *or* (||) clauses. In a switch statement, you can simply combine them by putting them on top of each other like this:

```
switch(grade){
  case "F":
  case "D":
    console.log("You've failed!");
    break;
  case "C":
  case "B":
    console.log("You've passed!");
    break;
  case "A":
    console.log("Nice!");
    break;
  default:
    console.log("I don't know this grade.");
}
```

For the values `F` and `D`, the same thing is happening. This is also true for `C` and `B`. When the value of `grade` is either `C` or `B`, it will log the following to the console:

```
You've passed!
```

This is more readable than the alternative if-else statement:

```
if(grade === "F" || grade === "D") {
  console.log("You've failed!");
} else if(grade === "C" || grade === "B") {
  console.log("You've passed!");
} else if(grade === "A") {
  console.log("Nice!");
} else {
  console.log("I don't know this grade.");
}
```

## Practice exercise 4.5

1. Create a variable called `prize` and use a prompt to ask the user to set the value by selecting a number between 0 and 10
2. Convert the prompt response to a number data type
3. Create a variable to use for the output message containing the value "`My Selection: `"
4. Using the switch statement (and creativity), provide a response back regarding a prize that is awarded depending on what number is selected
5. Use the switch break to add combined results for prizes
6. Output the message back to the user by concatenating your prize variable strings and the output message string

# Chapter projects

## Evaluating a number game

Ask the user to enter a number and check whether it's greater than, equal to, or less than a dynamic number value in your code. Output the result to the user.

# Friend checker game

Ask the user to enter a name, using the switch statement to return a confirmation that the user is a friend if the name selected is known in the case statements. You can add a default response that you don't know the person if it's an unknown name. Output the result into the console.

# Rock Paper Scissors game

This is a game between a player and the computer, where both will make a random selection of either Rock, Paper, or Scissors (alternatively, you could create a version using real player input!). Rock will beat out Scissors, Paper will beat out Rock, and Scissors will beat out Paper. You can use JavaScript to create your own version of this game, applying the logic with an if statement. Since this project is a little more difficult, here are some suggested steps:

1. Create an array that contains the variables Rock, Paper, and Scissors.
2. Set up a variable that generates a random number 0-2 for the player and then do the same for the computer's selection. The number represents the index values in the array of the 3 items.
3. Create a variable to hold a response message to the user. This can show the random results for the player and then also the result for the computer of the matching item from the array.
4. Create a condition to handle the player and computer selections. If both are the same, this results in a tie.
5. Use conditions to apply the game logic and return the correct results. There are several ways to do this with the condition statements, but you could check which player's index value is bigger and assign the victory accordingly, with the exception of Rock beating Scissors.
6. Add a new output message that shows the player selection versus the computer selection and the result of the game.

# Self-check quiz

1. What will be outputted to the console in this instance?

```
const q = '1';
switch (q) {
    case '1':
        answer = "one";
```

```
        break;
    case 1:
        answer = 1;
        break;
    case 2:
        answer = "this is the one";
        break;
    default:
        answer = "not working";
}
console.log(answer);
```

2. What will be outputted to the console in this instance?

```
const q = 1;

switch (q) {
    case '1':
        answer = "one";
    case 1:
        answer = 1;
    case 2:
        answer = "this is the one";
        break;
    default:
        answer = "not working";
}
console.log(answer);
```

3. What will be outputted to the console in this instance?

```
let login = false;
let outputHolder = "";
let userOkay = login ? outputHolder = "logout" : outputHolder =
"login";
console.log(userOkay);
```

4. What will be outputted to the console in this instance?

```
const userNames = ["Mike", "John", "Larry"];
const userInput = "John";
let htmlOutput = "";
if (userNames.indexOf(userInput) > -1) {
    htmlOutput = "Welcome, that is a user";
```

```
} else {
    htmlOutput = "Denied, not a user ";
}
console.log(htmlOutput + ": " + userInput);
```

5. What will be outputted to the console in this instance?

```
let myTime = 9;
let output;
if (myTime >= 8 && myTime < 12) {
    output = "Wake up, it's morning";
} else if (myTime >= 12 && myTime < 13) {
    output = "Go to lunch";
} else if (myTime >= 13 && myTime <= 16) {
    output = "Go to work";
} else if (myTime > 16 && myTime < 20) {
    output = "Dinner time";
} else if (myTime >= 22) {
    output = "Time to go to sleep";
} else {
    output = "You should be sleeping";
}
console.log(output);
```

6. What will be outputted to the console in this instance?

```
let a = 5;
let b = 10;
let c = 20;
let d = 30;
console.log(a > b || b > a);
console.log(a > b && b > a);
console.log(d > b || b > a);
console.log(d > b && b > a);
```

7. What will be outputted to the console in this instance?

```
let val = 100;
let message = (val > 100) ? `${val} was greater than 100` :
`${val} was LESS or Equal to 100`;
console.log(message);
let check = (val % 2) ? `Odd` : `Even`;
check = `${val} is ${check}`;
console.log(check);
```

# Summary

Now, let's wrap things up. In this chapter, we have covered conditional statements. We started with if else statements. Whenever the condition associated with the if is true, the if block gets executed. If the condition is false and there is an else block present, that will be executed. We have also seen ternary operators and the funky syntax they bring to the table. It is a short way of writing an if-else statement if you only need one statement per block.

And lastly, we have seen switch statements and how they can be used to optimize our conditional code. With the switch statement, we can compare one condition with many different cases. When they are equal (value and type), the code associated with the case gets executed.

In the next chapter, we are going to add loops to the mix! This is going to help us write more efficient code and algorithms.