The switch Keyword

This lesson is about how the switch conditional expression handles different conditions.

WE'LL COVER THE FOLLOWING

What is the `switch` Keyword?

The `switch` keyword allows us to specify the different actions the program can perform based on the value of a particular expression.

In an `if-else` expression, the condition always returns a boolean. This means that it can only cater for two cases: `true` and `false`.

With a `switch` keyword, the conditional expression is not restricted to booleans. Hence, we can define the behavior of the program for several cases. These cases are separated by the pipe operator, `|`.

The Structure

Here's the template for a typical `switch` conditional expression.

```
switch (value) {
  | (case1) => (execute expression1) // Case 1
  | (case2) => (execute expression2) // Case 2
  ... // Define the rest of the cases
};
```

In this template, `value` refers to the actual data which must be checked. Then we *switch* between different cases. Each case contains a possible value.

If the `value` variable matches the value in a particular case, the expression after the `=>` operator for that case is executed.

For example, if `value` is equal to `case1`, `expression1` will be executed. Otherwise, `value` will be further compared with `case2` and so on.

All of this will become clear when we see the `switch` expression in action.

Creating a `switch` Expression

Let's create a `switch` which checks whether the traffic light is `green`, `yellow`, or `red`. These three values will become our **cases**.

```
let light = "red";

let decision: string = switch (light) {
  | "red" => "Stop"
  | "yellow" => "Caution"
  | "green" => "Go"
  | _ => ""
};

Js.log(decision);
```

In the code above, we are comparing the value of `light` with all our cases in the `switch` expression. Since `light` is currently `red`, the `switch` expression returns `Stop`.

The `_` operator is used to encapsulate all other possible cases which are not useful to use. It is also known as the *default* case.

So, if `light` contains something which is not `red`, `yellow` or `green`, the switch will simply return a blank string instead of causing an error or a warning.

Generally, the output of a `switch` expression is stored in a `let` binding.

`switch` with Tuples

The `switch` expression works with tuples as well!

However, we must ensure that the order and types of all the tuple cases in our `switch` expression are the same as the input tuple.

```
let t = ("Batman", 40);

let name: string = switch (t) {
  | ("Batman", 40) => "Bruce Wayne"
  | ("Superman", 80) => "Clark Kent"
  | ("Wonder Woman", 2000) => "Diana Prince"
  | (_, _) => ""
};

Js.log(name);
```

If we add another tuple, `(25, "Spiderman")`, as a case in the `switch` expression, we would get a compilation error since the order of this tuple is incorrect.

Once again, the `_` operator is used to handle all tuples which are not useful for us.

The last conditional on our list is the **ternary operator**. We'll cover this in the next lesson.