Step 9 - Enums

Enums (short for enumerations) in TypeScript are a feature that allows you to define a set of named constants.

The concept behind an enumeration is to create a human-readable way to represent a set of constant values, which might otherwise be represented as numbers or strings.

Example 1 - Game

Let's say you have a game where you have to perform an action based on weather the user has pressed the `up` arrow key, `down` arrow key, `left` arrow key or `right` arrow key.

```
function doSomething(keyPressed) {
  // do something.
}
```

What should the `type` of keyPressed be?

Should it be a string? (`UP` , `DOWN` , `LEFT`, `RIGHT`) ?

Should it be numbers? (`1`, `2`, `3`, `4`) ?

The best thing to use in such a case is an `enum`.

```
enum Direction {
  Up,
  Down,
  Left,
  Right
}

function doSomething(keyPressed: Direction) {
  [illegible]
}
```

```
doSomething(Direction.Up)
```

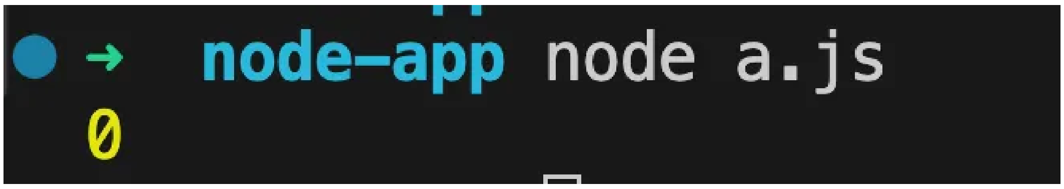
This makes code slightly `cleaner` to read out.

💡 The final value stored at `runtime` is still a number (0, 1, 2, 3).

2. What values do you see at runtime for `Direction.UP` ?

Try logging `Direction.Up` on screen

▶ Code



```
● → node-app node a.js
  0
```

This tells you that by default, `enums` get values as `0` , `1`, `2` ...

3. How to change values?

```
enum Direction {
    Up = 1,
    Down, // becomes 2 by default
    Left, // becomes 3
    Right // becomes 4
}

function doSomething(keyPressed: Direction) {
    // do something.
}

doSomething(Direction.Down)
```

▶ Solution

4. Can also be strings

```
enum Direction {
   Up = "UP",
   Down = "Down",
   Left = "Left",
   Right = 'Right'
}

function doSomething(keyPressed: Direction) {
   // do something.
}

doSomething(Direction.Down)
```

5. Common usecase in express

```
enum ResponseStatus {
   Success = 200,
   NotFound = 404,
   Error = 500
}

app.get("/', (req, res) => {
   if (!req.query.userId) {
         res.status(ResponseStatus.Error).json({})
   }
   // and so on...
      res.status(ResponseStatus.Success).json({});
})
```