Truthy and Falsy Values

Introduction to Truthy and Falsy Values

We have discussed type conversion and coercion to numbers and to strings, but booleans have not been mentioned yet. The reason is that it is important to first understand the concept of truthy and falsy values.

Falsy Values in JavaScript

Falsy values are values that are not exactly false, but will become false when converted to a boolean. In JavaScript, there are only five falsy values:

- 0
- The empty string
- undefined
- null
- NaN

False itself is already false, so it does not need to be included in the list of falsy values. All of these five values will be converted to false when coerced to a boolean. They are not exactly false initially, but they become false when converted to a boolean.

Truthy Values in JavaScript

Everything else is considered a truthy value. Truthy values are values that will be converted to true. For example, any number that is not zero or any string that is not an empty string will be converted to true when coerced to a boolean.

Demonstrating Boolean Conversion

Just like with numbers and strings, we can use the Boolean function to convert values to booleans.

javascript Code Sample

```
Boolean(0)

Boolean(undefined)

Boolean('Jonas')

Boolean({})

Boolean('')
```

Converting zero to a boolean results in false, and the same for undefined. The string 'Jonas' is true, so any string that is not an empty string is a truthy value. The same applies to an empty object, which is also a truthy value. However, converting an empty string results in false, as it is a falsy value.

Implicit Boolean Conversion in Practice

In practice, the conversion to boolean is almost always implicit, not explicit. In other words, it is type coercion that JavaScript does automatically behind the scenes. JavaScript does type coercion to booleans in two scenarios:

1. When using logical operators.
2. In a logical context, such as the condition of an if-else statement.

Example: Using Truthy and Falsy in If-Else Statements

Let us consider a variable called money. Initially, we do not have any money, so we set it to zero. We can use what we have learned to test if the person currently has any money or not.

javascript Code Sample

```
let money = 0;

if (money) {

  console.log("Don't spend it all");

} else {

  console.log("You should get a job");

}
```

Running this code results in 'You should get a job', which is the else part. The reason is that money is a number, and this number is zero. In the logical context of an if-else statement condition, JavaScript will try to coerce any value into a boolean. If it is not a boolean, JavaScript will convert it to a boolean using the truthy and falsy value rules. Since money is zero, and zero is a falsy value, the else block is executed.

If we change money to another value, such as 100, then 100 is a truthy value, and the if block will be executed.

javascript Code Sample

```
money = 100;

if (money) {

  console.log("Don't spend it all");

} else {

  console.log("You should get a job");

}
```

Now the output is 'Don't spend it all'.

Checking if a Variable is Defined

Another use case for truthy and falsy values is to check if a variable is actually defined or not. For example, let us define a variable called height.

javascript Code Sample

```
let height;

if (height) {

  console.log("YAY! Height is defined");

} else {

  console.log("Height is undefined");

}
```

Since height is undefined at this moment, and undefined is a falsy value, the else block is executed, and the output is 'Height is undefined'. If we assign a value to height, such as 100, then the if block is executed.

javascript Code Sample

```
height = 100;

if (height) {
  console.log("YAY! Height is defined");
} else {
  console.log("Height is undefined");
}
```

Now the output is 'YAY! Height is defined'.

Potential Pitfall: Zero as a Falsy Value

However, there can be a problem with this approach. If height is zero, which is a perfectly valid number, running the code will result in 'Height is undefined'. This is because zero is also a falsy value, and will trigger the else block. In this case, that is not what we want, and it is a bug in the application. The if-else statement did not account for the scenario where height is zero.

javascript Code Sample

```
height = 0;

if (height) {
  console.log("YAY! Height is defined");
} else {
  console.log("Height is undefined");
}
```

This illustrates that there can be problems using this approach. However, this can be fixed using logical operators, which will be discussed later in the section.

Conclusion

In the next lecture, equality operators will be discussed.

Key Takeaways

- Falsy values in JavaScript include zero, empty string, undefined, null, and NaN, and are converted to false when coerced to boolean.
- Truthy values are all values that are not falsy, such as non-zero numbers and non-empty strings.
- Boolean conversion in JavaScript is usually implicit, especially in logical contexts like if-else statements.
- Using truthy and falsy checks can sometimes lead to bugs, such as when zero is a valid value but is treated as falsy.