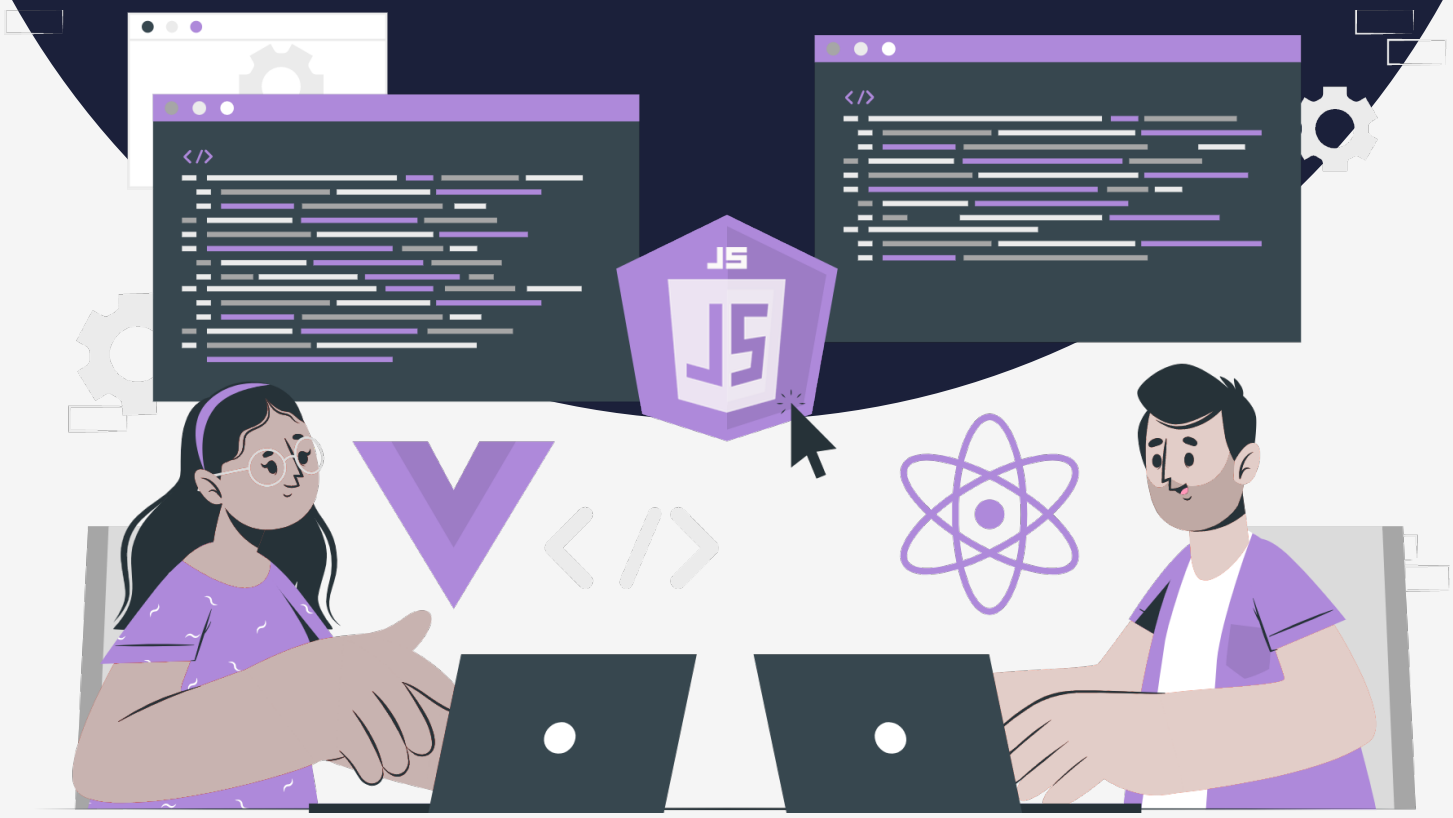
Lesson:

Use of try-catch in real-world Application



Topics Covered:

1. Uses of exception handling.
2. Implementation of try-catch.
3. Implementation of try-catch-finally.
4. Error object.
5. User-defined errors.

In the previous lecture, we looked into how exception handling is done, in javascript and its syntax. In this lecture let's understand why should one use exception handling and look at a real-world implementation of it.

There are many uses of using exception handling in our programs. Some of them include:

- Using exception handling will help us to deal with the errors properly and avoids the code from crashing and keeps the code running smoothly.
- We have a clear understanding of what's happening inside the code, if any error is encountered we can get some error messages which give us some information on what's happening inside our code.
- Exception handling makes it easier to debug your code. By catching exceptions, we can know the exact location where the error occurred and take appropriate action.

Considering the above examples it is always advised to use exception handling in all of the complex code we write.

Now let's look at a basic example to understand the implementation of exception handling in javascript.

Imagine that you are printing multiple lines to the console and in between, there is a console log message with an undefined variable. This will throw us an error message and the execution stops.

```
console.log("Hello from PW Skills");
console.log("Congratulations !! You have enrolled to the course.");
console.log("You have completed HTML & CSS. Hurray !!");
console.log(webDevelopment);
console.log("You are now learning Javascript");

/*
OUTPUT:

Hello from PW Skills
Congratulations !! You have enrolled to the course.
You have completed HTML & CSS. Hurray !!

ReferenceError: webDevelopment is not defined
*/
```

In the above code the execution stops when the error is encountered, and the further lines of code are not executed. This is not acceptable. We need to write code that is capable of handling exceptions.

Yes, we need to rectify the errors but the erros must not stop the execution or crash the program. To do this we can use exception handling.

```
console.log("Hello from PW Skills");
console.log("Congratulations !! You have enrolled to the course.");
console.log("You have completed HTML & CSS. Hurray !!");
try {
  console.log(webDevelopment);
} catch (error) {
  console.log(error);
}
console.log("You are now learning Javascript");

/* OUTPUT:

Hello from PW Skills
Congratulations !! You have enrolled to the course.
You have completed HTML & CSS. Hurray !!
ReferenceError: webDevelopment is not defined
You are now learning Javascript
*/
```

Using try-catch blocks we can handle the exceptions effectively. Here on encountering an error, the catch block accepts the error and logs it to the console. The error now is just a log statement which is why the program execution didn't stop. By doing this we can prevent our code from crashing and get the information about the error too for debugging.

The Error which is passed as a parameter to the catch block has some useful properties like

- name: return the name of the error.
- message: return the error message.

```
/*
OUTPUT:

Hello from PW Skills
Congratulations !! You have enrolled to the course.
You have completed HTML & CSS. Hurray !!
ReferenceError
You are now learning Javascript
*/

console.log("Hello from PW Skills");
console.log("Congratulations !! You have enrolled to the course.");
console.log("You have completed HTML & CSS. Hurray !!");
try {
  console.log(webDevelopment);
} catch (error) {
  console.log(error.message);
}
console.log("You are now learning Javascript");

/*
OUTPUT:

Hello from PW Skills
Congratulations !! You have enrolled to the course.
You have completed HTML & CSS. Hurray !!
webDevelopment is not defined
You are now learning Javascript

*/
```

Another block can be added, that is the finally block which will be executed regardless of the try or catch block.

```
console.log("Hello from PW Skills");
console.log("Congratulations !! You have enrolled to the course.");
console.log("You have completed HTML & CSS. Hurray !!");
try {
  console.log(webDevelopment);
} catch (error) {
  console.log(error.message);
} finally {
  console.log("You can now create Static webpages");
}
console.log("You are now learning Javascript");

/*
OUTPUT:

Hello from PW Skills
Congratulations !! You have enrolled to the course.
You have completed HTML & CSS. Hurray !!
webDevelopment is not defined
You can now create Static webpages
You are now learning Javascript
*/
```

Although, the finally block is used for a reason such as no matter whether the resources are fetched from an API or not it is essential to close the connection. These operations are written inside the finally block. We will be looking at them in upcoming lectures.

We can throw a new error by using the "throw" keyword, followed by a new Error object. The Error object can contain an error message, which will be displayed when the error is caught.

In our example let's consider if the student has not completed the assignment we store a false value in the "assignmentCompletion" variable and throw a new error "You have not completed the Assignment".

```
console.log("Hello from PW Skills");
console.log("Congratulations !! You have enrolled to the course.");
console.log("You have completed HTML & CSS. Hurray !!");

let assignmentCompletion = false;
```

```
try {
  if (assignmentCompletion === false)
    throw new Error("You have not completed the Assignment");
} catch (error) {
  console.log(error.message);
} finally {
  console.log("You can now create Static webpages");
}
console.log("You are now learning Javascript");

/*
OUTPUT:

Hello from PW Skills
Congratulations !! You have enrolled to the course.
You have completed HTML & CSS. Hurray !!
You have not completed the Assignment
You can now create Static webpages
You are now learning Javascript

*/
```