2.3 EXCEPTION HANDLING

An exception signifies the presence of an abnormal condition which requires special operable techniques. In programming terms, an exception is the anomalous code that breaks the normal flow of the code. Such exceptions require specialized programming constructs for its execution.

What is Exception Handling

In programming, exception handling is a process or method used for handling the abnormal statements in the code and executing them. It also enables to handle the flow control of the code/program. For handling the code, various handlers are used that process the exception and execute the code. For example, the Division of a non-zero value with zero will result into infinity always, and it is an exception. Thus, with the help of exception handling, it can be executed and handled.

In exception handling:

- A throw statement is used to raise an exception. It means when an abnormal condition occurs, an exception is thrown using throw.
- The thrown exception is handled by wrapping the code into the try…catch block. If an error is present, the catch block will execute, else only the try block statements will get executed.
- Thus, in a programming language, there can be different types of errors which may disturb the proper execution of the program.

Types of Errors

While coding, there can be three types of errors in the code:

1. **Syntax Error**: When a user makes a mistake in the pre-defined syntax of a programming language, a syntax error may appear.
2. **Runtime Error**: When an error occurs during the execution of the program, such an error is known as Runtime error. The codes which create runtime errors are known as Exceptions. Thus, exception handlers are used for handling runtime errors.

3. **Logical Error**: An error which occurs when there is any logical mistake in the program that may not produce the desired output, and may terminate abnormally. Such an error is known as Logical error.

Error Object

When a runtime error occurs, it creates and throws an Error object. Such an object can be used as a base for the user-defined exceptions too. An error object has two properties:

1. name: This is an object property that sets or returns an error name.
2. message: This property returns an error message in the string form.

Although Error is a generic constructor, there are following standard built-in error types or error constructors beside it:

1. EvalError: It creates an instance for the error that occurred in the eval(), which is a global function used for evaluating the js string code.
2. InternalError: It creates an instance when the js engine throws an internal error.
3. RangeError: It creates an instance for the error that occurs when a numeric variable or parameter is out of its valid range.
4. ReferenceError: It creates an instance for the error that occurs when an invalid reference is de-referenced.
5. SyntaxError: An instance is created for the syntax error that may occur while parsing the eval().
6. TypeError: When a variable is not a valid type, an instance is created for such an error.
7. URIError: An instance is created for the error that occurs when invalid parameters are passed in encodeURI() or decodeURI().

Exception Handling Statements

There are following statements that handle if any exception occurs:

- throw statements

- try…catch statements
- try…catch…finally statements.

JavaScript try…catch

A try…catch is a commonly used statement in various programming languages. Basically, it is used to handle the error-prone part of the code. It initially tests the code for all possible errors it may contain, then it implements actions to tackle those errors (if occur). A good programming approach is to keep the complex code within the try…catch statements.

Let's discuss each block of statement individually:

try{} statement: Here, the code which needs possible error testing is kept within the try block. In case any error occur, it passes to the catch{} block for taking suitable actions and handle the error. Otherwise, it executes the code written within.

catch{} statement: This block handles the error of the code by executing the set of statements written within the block. This block contains either the user-defined exception handler or the built-in handler. This block executes only when any error-prone code needs to be handled in the try block. Otherwise, the catch block is skipped.

Syntax:

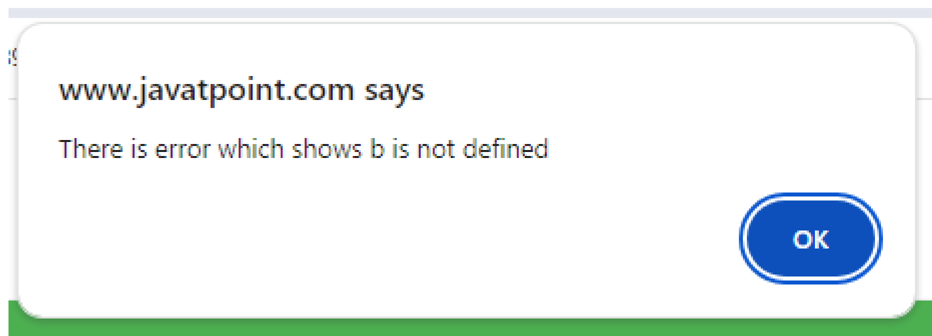
```
try{
expression; } //code to be written.
catch(error){
expression; } // code for handling the error.
```

try…catch example

```
<html>
<head> Exception Handling</br></head>
<body>
<script>
try{
var a= ["34","32","5","31","24","44","67"]; //a is an array
```

```
document.write(a);    // displays elements of a
document.write(b); //b is undefined but still trying to fetch its value. Thus catch block will be
 invoked
}catch(e){
alert("There is error which shows "+e.message); //Handling error
}
</script>
</body>
</html>
```

Output:



Throw Statement

Throw statements are used for throwing user-defined errors. User can define and throw their own custom errors. When throw statement is executed, the statements present after it will not execute. The control will directly pass to the catch block.

Syntax:

```
throw exception;
```

try…catch…throw syntax

```
try{
throw exception; // user can define their own exception
}
catch(error){
```

```
expression; }  // code for handling exception.
```

The exception can be a string, number, object, or boolean value.

throw example with try…catch

```
<html>
<head>Exception Handling</head>
<body>
<script>
try {
  throw new Error('This is the throw keyword'); //user-defined throw statement.
}
catch (e) {
 document.write(e.message); // This will generate an error message
}
</script>
</body>
</html>
```

Output:

Exception Handling This is the throw keyword

try…catch…finally statements

Finally is an optional block of statements which is executed after the execution of try and catch statements. Finally block does not hold for the exception to be thrown. Any exception is thrown or not, finally block code, if present, will definitely execute. It does not care for the output too.

Syntax:

```
try{
expression;
```

```
}
catch(error){
expression;
}
finally{
expression; } //Executable code
```

try…catch…finally example

```
<html>
<head>Exception Handling</head>
<body>
<script>
try{
var a=2;
if(a==2)
document.write("ok");
}
catch(Error){
document.write("Error found"+e.message);
}
finally{
document.write("Value of a is 2 ");
}
</script>
</body>
</html>
```

Output:

Exception Handling okValue of a is 2