RUNTIME ERROR

A runtime error is an error that occurs during the running of the program.

e.g.

```
<script type="text/javascript">
        window.printme();    //Uncaught    TypeError:    window.printme    is    not    a    function
  console.log("continue")
</script>
```

EXCEPTION HANDLING

Exception handling is a technique used in programming to handle and manage errors or exceptional conditions that may occur during the execution of a program. It allows you to gracefully handle errors, prevent program crashes, and provide meaningful feedback to users. In JavaScript, exception handling is done using try-catch blocks.

- **Try Block:** The code that might throw an exception is placed inside a try block. If an exception occurs within the try block, it will immediately exit the normal flow of execution and jump to the catch block.
- **Catch Block:** The catch block follows the try block and is used to handle the caught exception. It specifies the type of exception to catch (optional) and a block of code to execute if that specific exception occurs. The caught exception is assigned to a variable that you can use within the catch block.
  e.g.

```
<script type="text/javascript">
  try {
    // Code that might throw an exception
    var      result      =      someFunction();
    console.log("Result:", result);
  } catch (error) {
    // Handle the exception
    console.log("An exception occurred:", error.message);
  }
  console.log("continue")
</script>
```

- **Finally Block (optional):** You can also include a finally block after the catch block. This block is executed regardless of whether an exception was thrown or caught. It is commonly used to perform cleanup tasks, such as releasing resource

e.g.

```
try {
  // Code that might throw an exception
  var      result      =      someFunction();
  console.log("Result:", result);
} catch (error) {
  // Handle the exception
  console.log("An exception occurred:", error.message);
} finally {
  // Cleanup or finalization tasks (optional)
```

throw keyword

The throw keyword in JavaScript is used to manually trigger (or "throw") an error. When throw is used, JavaScript stops executing the current function and jumps to the nearest catch block, where you can handle the error. The throw keyword can be used to create custom errors or handle conditions that might otherwise not produce an error naturally.

Example 1: Using throw for Custom Error Messages
Suppose you have a function that validates a user’s age. You want to throw a specific error if the age is less than 18.

```
function validateAge(age) {
  try {
    if (age < 18) {
      throw new Error("User must be at least 18 years old.");
    }
    return "Age is valid.";
  } catch (error) {
    return error.message;  // Handling the error and returning the error message
  }
}

// Testing the function
console.log(validateAge(20));  // Output: "Age is valid."
console.log(validateAge(15));  // Output: "User must be at least 18 years old."
```

Explanation

- **throw new Error(...):** Here, we use the throw keyword to create a new Error object with a custom message. If age is less than 18, the function will throw an error, which will be caught by the catch block.
- **catch (error):** The catch block catches the error thrown in the try block, allowing us to handle it in a user-friendly way, such as returning an error message.

Example 2: Throwing Different Types of Values
You can throw any JavaScript value, not just Error objects. For instance, you might throw a string, number, or custom object to signal different types of issues.

```
function calculateSquareRoot(number) {
  try {
    if (number < 0) {
      throw "Cannot calculate square root of a negative number.";
    }
    return Math.sqrt(number);
  } catch (error) {
    return error;  // Returning the error message
  }
}

// Testing the function
console.log(calculateSquareRoot(16));   // Output: 4
console.log(calculateSquareRoot(-4));   // Output: "Cannot calculate square root of a negative number."
```

Explanation

- **throw "Cannot calculate square root of a negative number.":** Here, we throw a string instead of an Error object. This string acts as an error message, indicating why the operation cannot be performed.

Example 3: Custom Error Types Using Throw

You can also create custom error types by extending the Error class in JavaScript. This allows you to throw specific error types, making it easier to handle different error types separately.

```
class NegativeNumberError extends Error {
  constructor(message) {
    super(message);
    this.name = "NegativeNumberError";
  }
}

function calculateSquareRoot(number) {
  try {
    if (number < 0) {
      throw new NegativeNumberError("Negative numbers are not allowed.");
    }
    return Math.sqrt(number);
  } catch (error) {
    if (error instanceof NegativeNumberError) {
      return error.message; // Handle NegativeNumberError specifically
    } else {
      throw error; // Re-throw other unexpected errors
    }
  }
}

// Testing the function
console.log(calculateSquareRoot(16));   // Output: 4
console.log(calculateSquareRoot(-4));   // Output: "Negative numbers are not allowed."
```

Explanation

- **class NegativeNumberError:** We define a custom error class that extends Error. This allows us to throw a specific type of error (NegativeNumberError) instead of a general error.
- **throw new NegativeNumberError(...):** This creates an instance of NegativeNumberError and throws it, which allows the catch block to handle it specifically if desired.

Using throw helps you create meaningful error handling by controlling the flow of your program when certain conditions aren't met, and it enables better debugging and user feedback.

Akhilesh Kumar Gupta
(Technical Training specialist)
TechCarrel LLP

```
  console.log("Finally block executed");
}
```

e.g.

```
<!DOCTYPE html>
<html>
<body>
<h2>JavaScript Error Handling</h2>
<p>How to use <b>catch</b> to display an error.</p>
<p id="demo"></p>
<script>
try {
  adddlert("Welcome guest!");
}
catch(err) {
  document.getElementById("demo").innerHTML = err.message;
}
function adddlert(){

}
</script>
</body>
</html>
```