JAVASCRIPT EXCEPTION HANDLING

- 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.

Types of Errors:

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.

Exception Handling Statements:

1.trycatch statements

2.throw statements

3.trycatch and finally statements.

1.trycatch statements :

- 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

For example:

```
<!DOCTYPE html>
<html>
<body>
<h1>Javascript trycatch statements</h2>
<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>
```

2.Throw statements:

- 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.

  For example:

```
<!DOCTYPE html>
<html>
<body>
<h2>Throw statements</h2>
<script>
```

```
try {                    //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.
  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>
```

3.trycatch and 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 also.

  For example:

```
<!DOCTYPE html>
<html>
<body>
<h1>try catch and finally statements</h2>
<script>
try{
var a=2;     //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.
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>
```

COLLECTIONS CONCEPTS

1. FirstIndex:

   For example:

```
<!DOCTYPE html>
<html>
<body>
<script>
const fruits = ['banana', 'orange', 'mango', 'lemon']
let FirstIndex = fruits.length - 4
FirstFruit = fruits[FirstIndex]
document.write(FirstFruit);
</script>
</body>
</html>
```

2. LastIndex:

   For example:

   <!DOCTYPE html>

   <html>

   <body>

   <script>

```
const fruits = ['banana', 'orange', 'mango', 'lemon']
```

   let lastIndex = fruits.length - 1

   lastFruit = fruits[lastIndex]

```
document.write(lastFruit);
```

   </script>

   </body>

   </html>

3. push(),pop() and reversing():

   For example:

   <!DOCTYPE html>

   <html>

   <body>

   <script>

```
function reverse(str) {
   let stack = [];
                        // push letter into stack
                       //var declarations are globally scoped or function scoped while let and const are block scoped.
                      //var variables can be updated and re-declared within its scope.
                    //let variables can be updated but not re-declared.
                    //const variables can neither be updated nor re-declared.
   for (let i = 0; i < str.length; i++) {        //push,pop and reverse operation
      stack.push(str[i]);
   }
   // pop letter from the stack
   let reverseStr = '';
   while (stack.length > 0) {
      reverseStr += stack.pop();
   }
   return reverseStr;
}
document.write(reverse('Sandeep'));
```

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

JAVASCRIPT POLYMORPHISM:

For Example:

```
<!DOCTYPE html>
<html>
<body>
<script>
class A
  {
    display()
   {
      document.write("<br>");
     document.write("A is invoked");
   }
  }
class B extends A
  {
  }
```

```
var b=new B();
b.display();


class C extends A{
}
var c=new C();
c.display();
class D extends A {
}
var d=new D();
d.display();
</script>
</body>
</html>
```

Hashmap:

Example:

```
<!DOCTYPE html>
<html>
<body>
<script>
```

```
var map = new Map();
map
  .set(1, "sandeep")        //It stores the data in (Key, Value) pairs.
  .set(2, "anand")
  .set(3, "sachin")
  .forEach(function(value, key) {
document.write("<br>");
     document.write(key + " : " + value);
  });
</script>
</body>
</html>
```

JavaScript WeakMap Object:

Differences between Map and WeakMap:

- A WeakMap accepts only objects as keys whereas a Map,in addition to objects, accepts primitive datatype such as strings, numbers etc.

JavaScript WeakMap has(),delete() methods:

- The JavaScript WeakMap delete() method is used to remove the specified element from a WeakMap object.

  For example1:

```
<!DOCTYPE html>
<html>
<body>
<script>
var wm = new WeakMap();
var obj1={};
var obj2={};
wm.add(obj1);
wm.add(obj2);
document.write(wm.has(obj1));
document.write("<br>");
document.write(wm.has(obj2));
</script>
</body>
</html>
```

For example 2:

```
<!DOCTYPE html>
<html>
<body>
<script>
var wm = new WeakMap();//The JavaScript WeakMap delete() method is used to remove the specified object from WeakMap object.
var obj={};
wm.add(obj);
document.write(wm.has(obj));
wm.delete(obj);
document.write("<br>");
document.write(wm.has(obj));
</script>
</body>
</html>
```

JavaScript WeakMap set(),get() methods:

- The JavaScript WeakMap get() method returns the value of specified key of an element from a WeakMap object.
- The JavaScript WeakMap set() method is used to add or update an element to WeakMap object with the particular key-value pair. Each value must have a unique key

For example:

```
<!DOCTYPE html>
<html>
<body>
<script>
var wm = new WeakMap();
var obj1 = {};
var obj2 = {};
var obj3= {};
wm.set(obj1, "javascript");
wm.set(obj2, "HTML");
wm.set(obj3,"Bootstrap");
document.writeln(wm.get(obj1)+"<br>");
document.writeln(wm.get(obj2)+"<br>");
```

```
document.writeln(wm.get(obj3));
</script>
</body>
</html>
```

Other Methods :

Javascript concatenation:

```
<!DOCTYPE html>
<html>
<body>
<script>
const student = {
  name: "Mr.Sandeep"
}
const teacher = {          //var declarations are globally scoped or function scoped while let and const are block scoped.
```

```
                    //var variables can be updated and re-declared within its
scope.
                    //let variables can be updated but not re-declared.
                    //const variables can neither be updated nor re-declared.
  name: "Mr.Anand",
  room: "202"
}
let message =
  student.name +
  " please see " +
  teacher.name +
  " in " +
  teacher.room +
  " to pick up your report card."
document.write(message);
</script>
```

```
</body>

</html>
```