JavaScript

JavaScript Errors: -

Throw, and Try...Catch...Finally

The try statement defines a code block to run (to try).

The catch statement defines a code block to handle any error.

The finally statement defines a code block to run regardless of the result.

The throw statement defines a custom error.

```
try {
  adddlert("Welcome guest!");
}
catch(err) {
  document.getElementById("demo").innerHTML = err.message;
}

finally {

  Block of code to be executed regardless of the try / catch result

}
```

instead of alert, we typed adddlert. It will output adddlert is not defined.

JavaScript will actually create an Error object with two properties: name and message.

```
throw "Too big";    // throw a text
throw 500;          // throw a number
```

The Error Object: -

name - Sets or returns an error name

message - Sets or returns an error message (a string)

Eval Error: -

An EvalError indicates an error in the eval() function. Newer versions of JavaScript do not throw EvalError. Use SyntaxError instead.

Range Error: -

A RangeError is thrown if you use a number that is outside the range of legal values.

```
let num = 1;
try {
  num.toPrecision(500);   // A number cannot have 500 significant digits
}
catch(err) {
  document.getElementById("demo").innerHTML = err.name; //Range Error
}
```

Reference Error: -

A ReferenceError is thrown if you use (reference) a variable that has not been declared.

```
let x = 5;
try {
  x = y + 1;   // y cannot be used (referenced)
}

catch(err) {
 //err.name
}
```

Syntax Error: -

A SyntaxError is thrown if you try to evaluate code with a syntax error.

Type Error: -

A TypeError is thrown if an operand or argument is incompatible with the type expected by an operator or function.

Ex: - number 1 is there and applying .toUpperCase() method.

URI (Uniform Resource Identifier) Error: -

A URIError is thrown if you use illegal characters in a URI function.

```
try {
  decodeURI("%%%");   // You cannot URI decode percent signs
}
catch(err) {
  document.getElementById("demo").innerHTML = err.name;
}
```

There are some other non-standard errors defined by mozilla and Microsoft. These will not work in all browsers.

An example: -

```
let x = document.getElementById("demo").value;
  try {
    if(x.trim() == "") throw "is empty";
    if(isNaN(x)) throw "is not a number";
    x = Number(x);
    if(x > 10) throw "is too high";
    if(x < 5) throw "is too low";
  }
  catch(err) {
    message.innerHTML = "Error: " + err + ".";
  }
  finally {
   document.getElementById("demo").value = "";
  }
```

JavaScript Math Object: -

The JavaScript Math object allows you to perform mathematical tasks on numbers.

Math.PI; // 3.141592653589793

Unlike other objects, the Math object has no constructor. The Math object is static. All methods and properties can be used without creating a Math object first.

JavaScript provides 8 mathematical constants that can be accessed as Math properties.

Math.property

Math.E: //2.718281828459045 – e value

Math.PI: // 3.141592653589793 – pi value

Math.SQRT2: //1.4142135623730951 – square root of 2

Math.SQRT1_2: // 0.7071067811865476 –square root of 1/2

Math.LN2: // 0.6931471805599453 - ln(2)

Math.LN10: // 2.302585092994046- ln(10)

Math.LOG2E: // 1.4426950408889634 – $\log_2 e$

Math.Log10E: // 0.4342944819032518 – $\log_{10} e$

Math Methods: -

Math.method(number)

Number to Integer: -

Math.round(): -

Math.round(x) returns the nearest integer.

Math.round(4.6); // 5

Math.round(4.5); // 5

Math.round(4.4); // 4

Math.ceil(): -

Math.ceil(x) returns the value of x rounded up to its next integer.

```
Math.ceil(4.9);
Math.ceil(4.7);
Math.ceil(4.4);
Math.ceil(4.2); // these four will give 5
Math.ceil(-4.2); // -4
```

Math.floor(): -

Math.floor(x) returns the value of x rounded down to its nearest integer.

```
Math.floor(4.9);
Math.floor(4.7);
Math.floor(4.4);
Math.floor(4.2); // these four will give 4
Math.floor(-4.2);  // it will give -5
```

Math.trunc(): -

Math.trunc(x) returns the integer part of x.

```
Math.trunc(4.9);
Math.trunc(4.7);
Math.trunc(4.4);
Math.trunc(4.2); // 4
Math.trunc(-4.2);  // -4
```

Math.sign(): -

Math.sign(x) returns if x is negative, null or positive.

```
Math.sign(-4); // -1
Math.sign(0);  // 0
Math.sign(4); //1
```

Math.pow(): -

Math.pow(x, y) returns the value of x to the power of y.

```
Math.pow(8, 2); // 64
```

Math.sqrt(): -

Math.sqrt(x) returns the square root of x.

```
Math.sqrt(64); // 8
```

Math.abs(): -

Math.abs(x) returns the absolute (positive) value of x.

```
Math.abs(-4.7); // 4.7
```

Math.sin(): -

Math.sin(x) returns the sine (a value between -1 and 1) of the angle x (given in radians)

Angle in radians = Angle in degrees x PI / 180

```
Math.sin(90 * Math.PI / 180);     // returns 1 (the sine of 90 degrees)
```

Math.cos(): -

```
Math.cos(0 * Math.PI / 180);     // returns 1 (the cos of 0 degrees)
```

Math.min() and Math.max(): -

Math.min() and Math.max() can be used to find the lowest or highest value in a list of arguments.

Math.min(0, 150, 30, 20, -8, -200); // -200

Math.max(0, 150, 30, 20, -8, -200); // 150

Math.random(): -

Math.random() returns a random number between 0 (inclusive), and 1 (exclusive).

Math.random(); // 0.6754474418212251

The Math.log() Method: -

Math.log(x) returns the natural logarithm of x.

The natural logarithm returns the time needed to reach a certain level of growth.

Math.log(1); //0

Math.log(2); // 0.6931471805599453

The Math.log2() Method: -

Math.log2(x) returns the base 2 logarithm of x.

Math.log2(8); // 3

The Math.log10() Method: -

Math.log10(x) returns the base 10 logarithm of x.

Math.log10(1000); // 3

JavaScript Math Methods: -

Method	**Description**
Math.abs(x)	Returns the absolute value of x.
Math.acos(x)	Returns the arccosine of x (in radians).
Math.acosh(x)	Returns the hyperbolic arccosine of x.
Math.asin(x)	Returns the arcsine of x (in radians).
Math.asinh(x)	Returns the hyperbolic arcsine of x.
Math.atan(x)	Returns the arctangent of x (range: -PI/2 to PI/2 radians).
Math.atan2(y, x)	Returns the arctangent of y/x, considering the quadrant.
Math.atanh(x)	Returns the hyperbolic arctangent of x.

Math.cbrt(x)	Returns the cube root of x.
Math.ceil(x)	Rounds x upwards to the nearest integer.
Math.cos(x)	Returns the cosine of x (where x is in radians).
Math.cosh(x)	Returns the hyperbolic cosine of x.
Math.exp(x)	Returns e^x (Euler’s number raised to x).
Math.floor(x)	Rounds x downwards to the nearest integer.
Math.log(x)	Returns the natural logarithm (base e) of x.
Math.max(x, y, ..., n)	Returns the highest number in the given list.
Math.min(x, y, ..., n)	Returns the lowest number in the given list.
Math.pow(x, y)	Returns x raised to the power of y.
Math.random()	Returns a random number between 0 and 1.
Math.round(x)	Rounds x to the nearest integer.
Math.sign(x)	Returns -1 for negative, 0 for zero, and 1 for positive.
Math.sin(x)	Returns the sine of x (where x is in radians).
Math.sinh(x)	Returns the hyperbolic sine of x.
Math.sqrt(x)	Returns the square root of x.
Math.tan(x)	Returns the tangent of x (where x is in radians).
Math.tanh(x)	Returns the hyperbolic tangent of x.
Math.trunc(x)	Returns the integer part of x by removing the decimal.

JavaScript Random: -

```
Math.random(); // always returns a number lower than 1.
```

```
// Returns a random integer from 0 to 9:
Math.floor(Math.random() * 10);
```

```
// Returns a random integer from 0 to 10:
Math.floor(Math.random() * 11);
```

```
// Returns a random integer from 1 to 10:
Math.floor(Math.random() * 10) + 1;
```

A Proper Random Function: -

```
function getRndInteger(min, max) {
  return Math.floor(Math.random() * (max - min) ) + min;
} // max excluded
```

```
function getRndInteger(min, max) {
  return Math.floor(Math.random() * (max - min + 1) ) + min;
} //max included
```

JavaScript Break and Continue: -

The break statement "jumps out" of a loop.

The continue statement "jumps over" one iteration in the loop.

```
for (let i = 0; i < 10; i++) {
  if (i === 3) { break; }
  text += "The number is " + i + "<br>";
}
```

In the example above, the break statement ends the loop ("breaks" the loop) when the loop counter (i) is 3

```
// The number is 0 The number is 1 The number is 2
```

```
for (let i = 0; i < 10; i++) {
  if (i === 3) { continue; }
  text += "The number is " + i + "<br>";
}
```

```
The number is 0 The number is 1 The number is 2 The number is 4 The number is 5 The
number is 6 The number is 7 The number is 8 The number is 9
```

JavaScript Labels: -

To label JavaScript statements you precede the statements with a label name and a colon.

The break and the continue statements are the only JavaScript statements that can "jump out of" a code block.

```
break labelname;
```

```
continue labelname;
```

The continue statement (with or without a label reference) can only be used to skip one loop iteration.

The break statement, without a label reference, can only be used to jump out of a loop or a switch.

With a label reference, the break statement can be used to jump out of any code block.

A code block is a block of code between { and }

```
const cars = ["BMW", "Volvo", "Saab", "Ford"];
list: {
  text += cars[0] + "<br>";
  text += cars[1] + "<br>";
  break list;
  text += cars[2] + "<br>";
  text += cars[3] + "<br>";
}
```

the text will only contain BMW and volvo.

JavaScript Type Conversion: -

JavaScript variables can be converted to a new variable and another data type.

By the use of a JavaScript function or Automatically by JavaScript itself.

Converting Strings to Numbers: -

The global method Number() converts a variable (or a value) into a number.

A numeric string (like "3.14") converts to a number (like 3.14).

An empty string (like "") converts to 0.

A non numeric string (like "John") converts to NaN (Not a Number).

```
Number("3.14")
Number(Math.PI)
Number(" ")
Number("")

Number("99 88")
Number("John")
```

```
3.14, 3.141592653589793, 0, 0, NaN, NaN
```

Number Methods: -

Number() - Returns a number, converted from its argument

parseFloat() - Parses a string and returns a floating point number

parseInt() - Parses a string and returns an integer.

Converting Numbers to Strings: -

The global method String() can convert numbers to strings.

It can be used on any type of numbers, literals, variables, or expressions.

```
String(x)         // returns a string from a number variable x
String(123)       // returns a string from a number literal 123
String(100 + 23)  // returns a string from a number from an expression
```

The Number method toString() does the same.

Converting Dates to Numbers: -

```
d = new Date();
Number(d)          // returns 1740036556418

d.getTime()   // also does same
```

Converting Dates to Strings: -

String(Date()) // Thu Feb 20 2025 07:30:38 GMT+0000 (Coordinated Universal Time)

Date().toString()

The Date method toString() does the same.

Converting Booleans to Numbers: -

```
Number(false)     // returns 0
Number(true)      // returns 1
```

Converting Booleans to Strings: -

```
String(false)      // returns "false"
String(true)       // returns "true"
```

Automatic Type Conversion: -

When JavaScript tries to operate on a "wrong" data type, it will try to convert the value to a "right" type.

```
5 + null    // returns 5         because null is converted to 0
"5" + null  // returns "5null"   because null is converted to "null"
"5" + 2     // returns "52"      because 2 is converted to "2"
"5" - 2     // returns 3         because "5" is converted to 5
"5" * "2"   // returns 10        because "5" and "2" are converted to 5 and 2
```

Automatic String Conversion

JavaScript automatically calls the variable's toString() function when you try to "output" an object or a variable.

JSON: -

JSON stands for JavaScript Object Notation

JSON is a text format for storing and transporting data

JSON is "self-describing" and easy to understand

example of a JSON string.

'{"name":"John", "age":30, "car":null}'

It defines an object with 3 properties:

name, age, car

Each property has a value.

If you parse the JSON string with a JavaScript program, you can access the data as an object.

```
let personName = obj.name;
let personAge = obj.age;
```

JSON is language independent.

The JSON format is syntactically similar to the code for creating JavaScript objects. Because of this, a JavaScript program can easily convert JSON data into JavaScript objects.

Since the format is text only, JSON data can easily be sent between computers, and used by any programming language.

JavaScript has a built in function for converting JSON strings into JavaScript objects:

JSON.parse()

JavaScript also has a built-in function for converting an object into a JSON string:

JSON.stringify()

JSON Syntax: -

Data is in name/value pairs

Data is separated by commas

Curly braces hold objects

Square brackets hold arrays

A name/value pair consists of a field name (in double quotes), followed by a colon, followed by a value.

"name":"John"

JSON names require double quotes. keys must be strings.

values must be one of the following data types: string, number, object, array, boolean, null

The file type for JSON files is ".json"

The MIME type for JSON text is "application/json"

Numbers in JSON must be an integer or a floating point. {"age":30}

Objects as values in JSON must follow the JSON syntax.

```
{
"employee":{"name":"John", "age":30, "city":"New York"}
}
```

JSON.parse(): -

When receiving data from a web server, the data is always a string.

Ex: - '{"name":"John", "age":30, "city":"New York"}'

const obj = JSON.parse('{"name":"John", "age":30, "city":"New York"}');

then we can use that object.

When using the JSON.parse() on a JSON derived from an array, the method will return a JavaScript array, instead of a JavaScript object.

```
const text = '["Ford", "BMW", "Audi", "Fiat"]';
const myArr = JSON.parse(text);
```

Parsing Dates: -

Date objects are not allowed in JSON.

If you need to include a date, write it as a string.

You can convert it back into a date object later.

```
const text = '{"name":"John", "birth":"1986-12-14", "city":"New York"}';
const obj = JSON.parse(text);
obj.birth = new Date(obj.birth);
```

you can use the second parameter, of the JSON.parse() function, called reviver.

It is a function that checks each property, before returning the value.

```
const text = '{"name":"John", "birth":"1986-12-14", "city":"New York"}';
const obj = JSON.parse(text, function (key, value) {
  if (key == "birth") {
    return new Date(value);
  } else {
    return value;
  }
});
```

Parsing Functions: -

Functions are not allowed in JSON. If you need to include a function, write it as a string. You can convert it back into a function later.

```
const text = '{"name":"John", "age":"function () {return 30;}", "city":"New York"}';
const obj = JSON.parse(text);
obj.age = eval("(" + obj.age + ")");
```

avoid using functions in JSON.

JSON.stringify(): -

Stringify a JavaScript Object

```
const obj = {name: "John", age: 30, city: "New York"};
```

```
const myJSON = JSON.stringify(obj);
```

The result will be a string following the JSON notation.

Also, can be done for a array.

Storing Data: -

JSON makes it possible to store JavaScript objects as text.

Storing data in local storage.

```
// Storing data:
const myObj = {name: "John", age: 31, city: "New York"};
const myJSON = JSON.stringify(myObj);
localStorage.setItem("testJSON", myJSON);

// Retrieving data:
let text = localStorage.getItem("testJSON");
let obj = JSON.parse(text);
document.getElementById("demo").innerHTML = obj.name;
```

JSON.stringify() can not only convert objects and arrays into JSON strings, it can convert any JavaScript value into a string.

```
const num = 123e-5;
const myJSON = JSON.stringify(num); // 0.00123
```

```
let bool = new Boolean(1);
const myJSON = JSON.stringify(bool); //true
```

```
const obj = {name: "John", today: new Date(), city : "New York"};
const myJSON = JSON.stringify(obj);
```

```
// {"name":"John","today":"2025-02-20T11:27:30.237Z","city":"New York"}
```

The JSON.stringify() function will remove any functions from a JavaScript object, both the key and the value.

```
const obj = {name: "John", age: function () {return 30;}, city: "New York"};
const myJSON = JSON.stringify(obj);
```

```
// {"name":"John","city":"New York"}
```

This can be omitted if you convert your functions into strings before running the JSON.stringify() function. // obj.age = obj.age.toString();

Train-ticket console app: -

Main Menu

The user is presented with three options: Book Ticket, Cancel Ticket, and Exit.

1.Book Ticket Flow

Show all places: Display all available places (source and destination).

Enter Source: User enters the source station.

Validate Source: Check if the source is valid.

Enter Destination: User enters the destination station.

Validate Destination: Check if the destination is valid.

Enter No. of Passengers: User enters the number of passengers (max 10).

Validate No. of Passengers: Check if the number of passengers is valid.

Display Available Trains & Seat Details: Show available trains and seat details.

Select Train?: User selects a train.

Enter Passenger Details: User enters passenger details (name, age, class).

Validate Passenger Details: Check if passenger details are valid.

Check if seats are available for the selected class between the source and destination.

If seats are available, proceed to generate booking and passenger IDs.

If seats are not available, go back to displaying available trains and seat details.

Generate Booking ID & Passenger IDs: Generate unique IDs for the booking and passengers.

Update Seats in JSON: Update seat availability in trains.json.

Save Booking Details in Text File: Save booking details in bookings.txt.

2.Cancel Ticket Flow

Enter Booking ID: User enters the booking ID.

Validate Booking ID: Check if the booking ID is valid.

Display Passengers: Show passengers under the booking.

Select Passengers to Cancel?: User selects passengers to cancel.

Validate the selected passengers.

If the selection is invalid, return to the "Select Passengers to Cancel" step.

If the selection is valid, ask for confirmation (e.g., "Are you sure you want to cancel?").

If the user confirms (Yes), proceed to update the JSON file and modify the booking file.

If the user declines (No), return to the Main Menu

3.Exit

Save all data (updated JSON and booking details) and exit.

Validation Rules: -

Source and Destination:

Must exist in the list of places.

Source and destination cannot be the same.

Number of Passengers:

Must be a number between 1 and 10.

Passenger Details:

Name must be a non-empty string.

Age must be a positive number.

Class must be one of GEN, SL, or AC.

Booking ID:

Must exist in the bookings.txt file.

Error Handling

Invalid Input:

Display an error message and prompt the user to re-enter the input.

No Trains Available:

Display a message and return to the main menu.

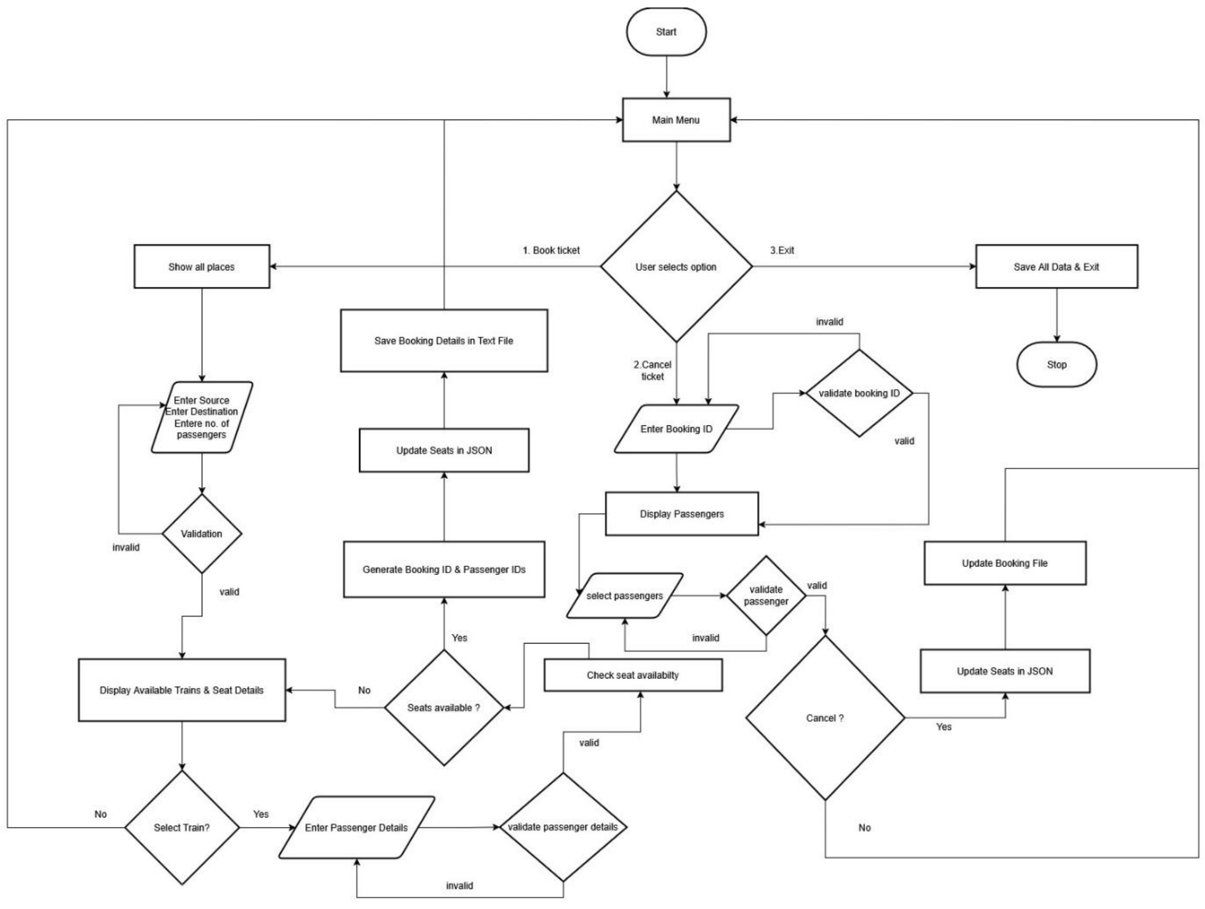
No Seats Available:

Display a message and prompt the user to select a different class or train and move to displaying available trains and seat details.

Booking Not Found:

Display a message and return to the main menu.

Flow diagram: -


Start
Main Menu
User selects option
1. Book ticket
3.Exit
2.Cancel ticket
Show all places
Save All Data & Exit
Stop
Save Booking Details in Text File
Update Seats in JSON
Generate Booking ID & Passenger IDs
Enter Source
Enter Destination
Entere no. of passengers
Validation
invalid
valid
Display Available Trains & Seat Details
Seats available ?
Yes
No
Check seat availabilty
Select Train?
Yes
No
Enter Passenger Details
validate passenger details
invalid
valid
Enter Booking ID
validate booking ID
invalid
valid
Display Passengers
select passengers
validate passenger
invalid
valid
Cancel ?
Yes
No
Update Seats in JSON
Update Booking File