données	Description	Utilisation principale	Caractéristiques clés
Tableau (Array)	Collection d'éléments de même type, indexée par position	Stockage et accès rapides par index	Taille fixe, accès en O(1), insertion/suppression coûteuses.
Liste chaînée	Suite de nœuds où chaque nœud contient un élément et un pointeur vers le suivant	Gestion dynamique des données	Taille dynamique, insertion/suppression en O(1), accès séquentiel.
Pile (Stack)	Structure LIFO (Last In, First Out)	Gestion des appels récursifs, annulation	Opérations `push` et `pop` en O(1).
File (Queue)	Structure FIFO (First In, First Out)	Gestion des files d'attente, traitement par lots	Opérations `enqueue` et `dequeue` en O(1).
Dictionnaire (HashMap)	Stockage clé-valeur avec accès rapide	Recherche, mappage rapide des données	Temps d'accès en O(1) dans le cas idéal, dépend du hachage.
Arbre (Tree)	Structure hiérarchique avec un nœud racine et des sous-nœuds	Organisation des données, recherche hiérarchique	Binaire, équilibré, ou n-aire pour différentes utilisations.
Graphes (Graphs)	Ensemble de nœuds (sommets) connectés par des liens (arêtes)	Modélisation de réseaux sociaux, chemins	Orienté ou non, pondéré ou non.
Tas (Heap)	Arbre partiellement ordonné utilisé pour accéder à l'élément minimum ou maximum	Algorithmes de tri, file de priorité	Structure efficace pour le tri en O(log n).
Table de hachage	Implémente un mappage entre clés et valeurs	Recherche rapide, dictionnaires	Résout les collisions avec le chaînage ou le probing.
Matrice (Matrix)	Tableau bidimensionnel utilisé pour représenter des données tabulaires	Calcul matriciel, représentation de graphes	Taille fixe, manipulation efficace pour des algorithmes spécifiques.

Un **système Linux** est un **système d'exploitation** basé sur le noyau Linux, conçu pour être **ouvert**, **libre** et **multitâche**. Il est composé du noyau (qui gère le matériel) et de logiciels essentiels (utilitaires, interfaces graphiques, etc.). Linux est souvent utilisé dans des **distributions** (ex. : Ubuntu, Fedora) et est réputé pour sa **sécurité**, sa **stabilité** et sa **flexibilité**. Il est adapté à divers usages, allant des ordinateurs personnels aux serveurs.

Commande	Description
`ls`	Affiche la liste des fichiers et répertoires dans un répertoire donné.
`cd`	Change de répertoire.
`pwd`	Affiche le répertoire de travail actuel (chemin complet).
`mkdir`	Crée un nouveau répertoire.
`rmdir`	Supprime un répertoire vide.
`rm`	Supprime un fichier ou un répertoire (avec `-r` pour répertoire, `-f` pour forcer).
`cp`	Copie un fichier ou un répertoire.
`mv`	Déplace ou renomme un fichier ou un répertoire.
`touch`	Crée un fichier vide ou met à jour l'heure de modification d'un fichier.
`cat`	Affiche le contenu d'un fichier.
`more`	Affiche le contenu d'un fichier avec pagination.
`less`	Affiche le contenu d'un fichier avec navigation en avant et en arrière.
`echo`	Affiche un message à l'écran.
`man`	Affiche le manuel d'une commande.
`chmod`	Modifie les permissions d'un fichier ou répertoire.
`chown`	Change le propriétaire et le groupe d'un fichier ou répertoire.
`ps`	Affiche les processus en cours.
`top`	Affiche les processus en temps réel et les ressources système utilisées.
`kill`	Envoie un signal à un processus, souvent pour le terminer.
`df`	Affiche l'espace disque disponible sur les systèmes de fichiers.
`du`	Affiche la taille des fichiers et répertoires.
`free`	Affiche la mémoire libre et utilisée sur le système.
`grep`	Recherche un motif dans un fichier ou une sortie.
`find`	Recherche des fichiers et répertoires selon des critères spécifiés.
`wget`	Télécharge des fichiers depuis Internet.
`curl`	Effectue des requêtes réseau (HTTP, FTP, etc.).
`tar`	Archive ou extrait des fichiers dans un fichier tar.
`zip`	Compresse des fichiers dans une archive ZIP.
`unzip`	Décompresse une archive ZIP.
`sudo`	Exécute une commande avec des privilèges d'administrateur.
`apt-get` / `apt`	Gère les paquets sous les distributions basées sur Debian (installation, mise à jour).
`yum`	Gère les paquets sous les distributions basées sur Red Hat (installation, mise à jour).
`systemctl`	Gère les services et démons système sous `systemd`.

La récursivité est une technique en programmation où une fonction s'appelle elle-même directement ou indirectement jusqu'à atteindre une condition d'arrêt.

function factorial(n)
{ if (n === 0)// Condition d'arrêt
{ return 1; }
return n * factorial(n - 1); // Appel récursif
console.log(factorial(5)); // Output: 120

- ☐ F(0)=0, F(1)=1F(1) = 1F(1)=1
- ☐ F(n)=F(n−1)+F(n−2) pour n≥2n \geq 2n≥2

function fibonacci(n) {
if (n === 0) return 0; // Base case
if (n === 1) return 1; // Base case
return fibonacci(n - 1) + fibonacci(n - 2); // Recursive case }
console.log(fibonacci(6)); // Output: 8

---

1. Variables
```
let x = 10; // Block scope variable
const y = 20; // Constant variable
var z = 30; // Function scope variable
```
2. Data Types
```
let str = "Hello"; // String
let num = 123; // Number
let bool = true; // Boolean
let arr = [1, 2, 3]; // Array
let obj = { key: "value" }; // Object
let und; // Undefined
let nul = null; // Null
```
3. Functions
```
function greet(name) {
    return `Hello, ${name}`;
}
let greetArrow = (name) => `Hello, ${name}`; // Arrow Function
```
4. Conditionals
```
if (x > 5) {
    console.log("x is greater than 5");
} else if (x === 5) {
    console.log("x is 5");
} else {
    console.log("x is less than 5");
}
```
5. Loops
```
for (let i = 0; i < 5; i++) {
    console.log(i); // For Loop
}
let i = 0;
while (i < 5) {
    console.log(i); // While Loop
    i++;
}
do {
    console.log(i); // Do-While Loop
    i--;
} while (i > 0);
```
6. Arrays
```
let fruits = ["Apple", "Banana", "Cherry"];
fruits.push("Date"); // Add to end
fruits.pop(); // Remove from end
fruits.shift(); // Remove from start
fruits.unshift("Apricot"); // Add to start
console.log(fruits);
```
7. Objects
```
let person = {
    firstName: "John",
    lastName: "Doe",
    age: 25,
    greet() {
        return `Hello, ${this.firstName}`;
    }
};
console.log(person.greet());
```
8. DOM Manipulation
```
let elem = document.getElementById("myElement"); // Get element by ID
elem.textContent = "New Content"; // Change text content
```
9. Event Handling
```
let button = document.querySelector("button");
```

```
button.addEventListener("click", () => {
    alert("Button clicked!");
});
```

10. Promises & Async/Await

```
let fetchData = () => {
    return new Promise((resolve, reject) => {
        setTimeout(() => resolve("Data
fetched!"), 1000);
    });
};
fetchData().then((data) =>
console.log(data));
async function fetchDataAsync() {
    let data = await fetchData();
    console.log(data);
}
fetchDataAsync();
```

11. Error Handling

```
try {
    throw new Error("Something went
wrong");
} catch (error) {
    console.error(error.message);
}
```

Node.js

Node.js est une plateforme puissante pour créer des applications côté serveur. En utilisant **Express.js**, un framework léger pour Node.js, vous pouvez rapidement développer des API et des applications web. Voici les concepts essentiels pour débuter :

---

1. Concepts de base de Node.js

- **Introduction :**
    - Node.js permet d'exécuter JavaScript côté serveur.
    - Basé sur le moteur V8 de Google Chrome.
    - Orienté événement et non-bloquant (utilisé le modèle d'E/S non-bloquant).
- **Points essentiels :**
- Écosystème riche grâce à **npm** (Node Package Manager).
- Support des **modules** (CommonJS ou ES Modules).
- Manipulation des fichiers avec le module `fs`.
- Gestion des événements avec `EventEmitter`.
- **Commandes clés :**
- `node fichier.js` pour exécuter un fichier.
- `npm init` pour initialiser un projet.
- `npm install <module>` pour installer un package.

---

2. Concepts de base d'Express.js

- **Qu'est-ce que Express ?**
    - Framework minimaliste pour créer des applications web.
    - Simplifie la gestion des routes et des middlewares.

Installation :

```
npm install express
```

Structure de base d'une application :

```
const express = require('express');

const app = express();

const port = 3000;

// Définir une route

app.get('/', (req, res) => {

    res.send('Hello, world!');

});

// Démarrer le serveur
```

```
app.listen(port, () => {

   console.log(`Serveur démarré sur http://localhost:${port}`);

});
```

---

3. Concepts clés pour Express

1. **Routes** :
    - Permettent de définir comment traiter des requêtes pour des chemins spécifiques.

Exemple :
```
app.get('/users', (req, res) => {

   res.send('Liste des utilisateurs');

});
```

2. **Middleware** :

Fonction exécutée entre la réception d'une requête et la réponse.

Utilisation pour : journalisation, authentification, parsing des données.

Exemple :
```
app.use((req, res, next) => {

   console.log(`${req.method} ${req.path}`);

   next();

});
```

3. **Requêtes et réponses** :

Récupérer des données :

- **Query String** : `req.query`
- **Paramètres dynamiques** : `req.params`
- **Body** : `req.body` (nécessite `body-parser` ou `express.json()`).

4. **Routing avancé** :

Gérer plusieurs verbes HTTP sur une route unique :
```
app.route('/profile')

   .get((req, res) => res.send('Get Profile'))

   .post((req, res) => res.send('Create Profile'));
```

5. **Static Files** :

Servir des fichiers statiques comme CSS, JS, images.

Exemple :
```
app.use(express.static('public'));
```

---

4. Apprendre les concepts de Node.js nécessaires pour Express

1. **Modules :**
    - `require` pour importer des fichiers/modules.

Exemple :
```
const fs = require('fs'); // Module natif

const myModule = require('./myModule'); // Fichier local
```

2. **Asynchronicité :**
   Utilisez des callbacks, des Promcatchises (`then/`) ou `async/await`.

Exemple :
```
const fs = require('fs/promises');

async function readFile() {

   const data = await fs.readFile('file.txt', 'utf8');

   console.log(data);

} readFile();
```

3. **Événements :**
    - Gérer des événements avec `EventEmitter`.

Exemple :
```
const EventEmitter = require('events');

const emitter = new EventEmitter();

emitter.on('event', () => console.log('Event déclenché'));

emitter.emit('event');
```

4. **Gestion des erreurs :**
    - Utilisez `try/catch` ou le middleware `error-handling` d'Express.

---

5. Fonctionnalités avancées à explorer

Routing dynamique : Traitez des chemins dynamiques avec `req.params`.

```
app.get('/user/:id', (req, res) => {
  res.send(`Utilisateur ${req.params.id}`);
});
```

1. **APIs RESTful** :

Apprenez à structurer une API REST avec Express.

Exemple d'endpoints :

`GET /articles` - Liste d'articles.

`POST /articles` - Ajouter un article.

`PUT /articles/:id` - Modifier un article.

`DELETE /articles/:id` - Supprimer un article.

2. **Base de données :**
    - Apprenez à intégrer des bases comme **MongoDB** (via Mongoose), **PostgreSQL**, ou **MySQL**.

Exemple avec MongoDB :
```
const mongoose = require('mongoose');

mongoose.connect('mongodb://localhost:27017/mydb', { useNewUrlParser: true });
```

3. **Authentification :**
    - Utilisez des bibliothèques comme **Passport.js** ou **JWT (Json Web Token)** pour sécuriser vos applications.

---

6. Outils recommandés

- **Postman** ou **Insomnia** : Tester vos API.

Nodemon : Relancer automatiquement le serveur à chaque modification.
```
npm install -g nodemon

nodemon app.js
```

-
- **Debugger** : Utilisez les outils intégrés de Node.js ou `console.log`.

---

7. Concepts supplémentaires

- **CORS** :
    - Activez les Cross-Origin Resource Sharing si votre application frontend appelle une API sur un autre domaine.

Exemple :
```
const cors = require('cors');

app.use(cors());
```

- **Sécurité** :
  Utilisez des modules comme **helmet** pour protéger votre application.

```
const helmet = require('helmet');

app.use(helmet());
```

JS CheatSheet

Loops ↶

For Loop

```
for (var i = 0; i < 10; i++) {
    document.write(i + ": " + i*3 + "<br />");
}
var sum = 0;
for (var i = 0; i < a.length; i++) {
    sum + = a[i];
}               // parsing an array
html = "";
for (var i of custOrder) {
    html += "<li>" + i + "</li>";
}
```

While Loop

```
var i = 1;                      // initialize
while (i < 100) {               // enters the cycle
    i *= 2;                     // increment to avo
    document.write(i + ", ");   // output
}
```

Do While Loop

```
var i = 1;                      // initialize
do {                            // enters cycle at
    i *= 2;                     // increment to avo
    document.write(i + ", ");   // output
} while (i < 100)               // repeats cycle if
```

Break

```
for (var i = 0; i < 10; i++) {
    if (i == 5) { break; }          // stops and ex
    document.write(i + ", ");       // last output
}
```

Continue

```
for (var i = 0; i < 10; i++) {
    if (i == 5) { continue; }       // skips the re
    document.write(i + ", ");       // skips 5
}
```

Variables x

```
var a;                          // variable
var b = "init";                 // string
var c = "Hi" + " " + "Joe";     // = "Hi Joe"
var d = 1 + 2 + "3";            // = "33"
var e = [2,3,5,8];              // array
var f = false;                  // boolean
var g = /()/;                   // RegEx
var h = function(){};           // function object
const PI = 3.14;                // constant
var a = 1, b = 2, c = a + b;    // one line
let z = 'zzz';                  // block scope loca
```

Strict mode

```
"use strict";   // Use strict mode to write secure
x = 1;          // Throws an error because variable
```

Basics ➤

On page script

```
<script type="text/javascript">  ...
</script>
```

Include external JS file

```
<script src="filename.js"></script>
```

Delay - 1 second timeout

```
setTimeout(function () {

}, 1000);
```

Functions

```
function addNumbers(a, b) {
    return a + b; ;
}
x = addNumbers(1, 2);
```

Edit DOM element

```
document.getElementById("elementID").innerHTML =
```

Output

```
console.log(a);             // write to the browse
document.write(a);          // write to the HTML
alert(a);                   // output in an alert
confirm("Really?");         // yes/no dialog, retu
prompt("Your age?","0");    // input dialog. Secon
```

Comments

```
/* Multi line
   comment */
// One line
```

If - Else ⇵

```
if ((age >= 14) && (age < 19)) {        // logica
    status = "Eligible.";               // execute
} else {                                // else bl
    status = "Not eligible.";           // execute
}
```

Switch Statement

```
switch (new Date().getDay()) {      // input is cu
    case 6:                         // if (day ==
        text = "Saturday";
        break;
    case 0:                         // if (day ==
        text = "Sunday";
        break;
    default:                        // else...
        text = "Whatever";
}
```

Data Types ℜ

```
var age = 18;                       // number
var name = "Jane";                  // string
```

Values

```
false, true                    // boolean
18, 3.14, 0b10011, 0xF6, NaN   // number
"flower", 'John'               // string
undefined, null , Infinity     // special
```

Operators

```
a = b + c - d;      // addition, substraction
a = b * (c / d);    // multiplication, division
x = 100 % 48;       // modulo. 100 / 48 remainder =
a++; b--;           // postfix increment and decrem
```

Bitwise operators

&	AND	5 & 1 (0101 & 0001)	1 (1)
|	OR	5 | 1 (0101 | 0001)	5 (101)
~	NOT	~ 5 (~0101)	10 (1010)
^	XOR	5 ^ 1 (0101 ^ 0001)	4 (100)
<<	left shift	5 << 1 (0101 << 1)	10 (1010)
>>	right shift	5 >> 1 (0101 >> 1)	2 (10)
>>>	zero fill right shift	5 >>> 1 (0101 >>> 1)	2 (10)

Arithmetic

```
a * (b + c)         // grouping
person.age          // member
person[age]         // member
!(a == b)           // logical not
a != b              // not equal
typeof a            // type (number, object, functi
x << 2  x >> 3      // minary shifting
a = b               // assignment
a == b              // equals
a != b              // unequal
a === b             // strict equal
a !== b             // strict unequal
a < b   a > b       // less and greater than
a <= b  a >= b      // less or equal, greater or eq
a += b              // a = a + b (works with - * %.
a && b              // logical and
a || b              // logical or
```

Numbers and Math ∑

```
var pi = 3.141;
pi.toFixed(0);           // returns 3
pi.toFixed(2);           // returns 3.14 - for worki
pi.toPrecision(2)        // returns 3.1
pi.valueOf();            // returns number
Number(true);            // converts to number
Number(new Date())       // number of milliseconds s
parseInt("3 months");    // returns the first number
parseFloat("3.5 days"); // returns 3.5
Number.MAX_VALUE         // largest possible JS numb
Number.MIN_VALUE         // smallest possible JS num
Number.NEGATIVE_INFINITY// -Infinity
Number.POSITIVE_INFINITY// Infinity
```

Math.

```
var pi = Math.PI;        // 3.141592653589793
Math.round(4.4);         // = 4 - rounded
Math.round(4.5);         // = 5
Math.pow(2,8);           // = 256 - 2 to the power o
Math.sqrt(49);           // = 7 - square root
Math.abs(-3.14);         // = 3.14 - absolute, posit
Math.ceil(3.14);         // = 4 - rounded up
Math.floor(3.99);        // = 3 - rounded down
Math.sin(0);             // = 0 - sine
```

```
var name = {first:"Jane", last:"Doe"};  // object
var truth = false;                      // boolear
var sheets = ["HTML","CSS","JS"];       // array
var a; typeof a;                        // undefi
var a = null;                           // value
```

Objects

```
var student = {                 // object name
    firstName:"Jane",           // list of propert
    lastName:"Doe",
    age:18,
    height:170,
    fullName : function() {     // object functio
       return this.firstName + " " + this.lastNam
    }
};
student.age = 19;            // setting value
student[age]++;              // incrementing
name = student.fullName();   // call object functi
```

Strings ⊗

```
var abc = "abcdefghijklmnopqrstuvwxyz";
var esc = 'I don\'t \n know';   // \n new line
var len = abc.length;           // string length
abc.indexOf("lmno");            // find substring
abc.lastIndexOf("lmno");        // last occurance
abc.slice(3, 6);                // cuts out "def"
abc.replace("abc","123");       // find and repla
abc.toUpperCase();              // convert to upp
abc.toLowerCase();              // convert to low
abc.concat(" ", str2);          // abc + " " + str
abc.charAt(2);                  // character at i
abc[2];                         // unsafe, abc[2]
abc.charCodeAt(2);              // character code
abc.split(",");                 // splitting a str
abc.split("");                  // splitting on c
128.toString(16);               // number to hex(
```

Events

```
<button onclick="myFunction();">
   Click here
</button>
```

Mouse

onclick, oncontextmenu, ondblclick, onmousedown, onmouseenter, onmouseleave, onmousemove, onmouseover, onmouseout, onmouseup

Keyboard

onkeydown, onkeypress, onkeyup

Frame

onabort, onbeforeunload, onerror, onhashchange, onload onpageshow, onpagehide, onresize, onscroll, onunload

Form

onblur, onchange, onfocus, onfocusin, onfocusout, oninput, oninvalid, onreset, onsearch, onselect, onsubmit

Drag

ondrag, ondragend, ondragenter, ondragleave, ondragover, ondragstart, ondrop

Clipboard

oncopy, oncut, onpaste

```
Math.cos(Math.PI);       // OTHERS: tan,atan,asin,ac
Math.min(0, 3, -2, 2);   // = -2 - the lowest value
Math.max(0, 3, -2, 2);   // = 3 - the highest value
Math.log(1);             // = 0 natural logarithm
Math.exp(1);             // = 2.7182pow(E,x)
Math.random();           // random number between 0
Math.floor(Math.random() * 5) + 1;  // random integ
```

Constants like Math.PI:

E, PI, SQRT2, SQRT1_2, LN2, LN10, LOG2E, Log10E

Dates 31

Mon Feb 17 2020 13:42:03 GMT+0200 (Eastern European Standard Time)

```
var d = new Date();
```

1581939723047 miliseconds passed since 1970

```
Number(d)
```

```
Date("2017-06-23");                 // date declara
Date("2017");                       // is set to Ja
Date("2017-06-23T12:00:00-09:45");  // date - time
Date("June 23 2017");               // long date fo
Date("Jun 23 2017 07:45:00 GMT+0100 (Tokyo Time)");
```

Get Times

```
var d = new Date();
a = d.getDay();     // getting the weekday

getDate();          // day as a number (1-31)
getDay();           // weekday as a number (0-6)
getFullYear();      // four digit year (yyyy)
getHours();         // hour (0-23)
getMilliseconds();  // milliseconds (0-999)
getMinutes();       // minutes (0-59)
getMonth();         // month (0-11)
getSeconds();       // seconds (0-59)
getTime();          // milliseconds since 1970
```

Setting part of a date

```
var d = new Date();
d.setDate(d.getDate() + 7); // adds a week to a dat

setDate();          // day as a number (1-31)
setFullYear();      // year (optionally month and d
setHours();         // hour (0-23)
setMilliseconds();  // milliseconds (0-999)
setMinutes();       // minutes (0-59)
setMonth();         // month (0-11)
setSeconds();       // seconds (0-59)
setTime();          // milliseconds since 1970)
```

Global Functions ()

```
eval();                     // executes a string as
String(23);                 // return string from n
(23).toString();            // return string from n
Number("23");               // return number from s
decodeURI(enc);             // decode URI. Result:
encodeURI(uri);             // encode URI. Result:
decodeURIComponent(enc);    // decode a URI compone
encodeURIComponent(uri);    // encode a URI compone
isFinite();                 // is variable a finite
isNaN();                    // is variable an illeg
parseFloat();               // returns floating poi
parseInt();                 // parses a string and
```

Media

onabort, oncanplay, oncanplaythrough, ondurationchang onended, onerror, onloadeddata, onloadedmetadata, onloadstart, onpause, onplay, onplaying, onprogress, onratechange, onseeked, onseeking, onstalled, onsuspend, ontimeupdate, onvolumechange, onwaiting

Animation

animationend, animationiteration, animationstart

Miscellaneous

transitionend, onmessage, onmousewheel, ononline, onoffline, onpopstate, onshow, onstorage, ontoggle, onwheel, ontouchcancel, ontouchend, ontouchmove, ontouchstart

Arrays ≡

```
var dogs = ["Bulldog", "Beagle", "Labrador"];
var dogs = new Array("Bulldog", "Beagle", "Labrad

alert(dogs[1]);             // access value at in
dogs[0] = "Bull Terier";    // change the first i

for (var i = 0; i < dogs.length; i++) {     // pa
    console.log(dogs[i]);
}
```

Methods

```
dogs.toString();                        // conver
dogs.join(" * ");                       // join: '
dogs.pop();                             // remove
dogs.push("Chihuahua");                 // add ne
dogs[dogs.length] = "Chihuahua";        // the sa
dogs.shift();                           // remove
dogs.unshift("Chihuahua");              // add ne
delete dogs[0];                         // change
dogs.splice(2, 0, "Pug", "Boxer");      // add el
var animals = dogs.concat(cats,birds);  // join t
dogs.slice(1,4);                        // elemen
dogs.sort();                            // sort s
dogs.reverse();                         // sort s
x.sort(function(a, b){return a - b});   // numeri
x.sort(function(a, b){return b - a});   // numeri
highest = x[0];                         // first
x.sort(function(a, b){return 0.5 - Math.random()}
```

concat, copyWithin, every, fill, filter, find, findIndex, forEach, indexOf, isArray, join, lastIndexOf, map, pop, push, reduce, reduceRight, reverse, shift, slice, some, sort, splice, toString, unshift, valueOf

Regular Expressions \n

```
var a = str.search(/CheatSheet/i);
```

Modifiers

i	*perform case-insensitive matching*
g	*perform a global match*
m	*perform multiline matching*

Patterns

****	*Escape character*
\d	*find a digit*
\s	*find a whitespace character*
\b	*find match at beginning or end of a word*

Errors ⚠

```
try {                              // block of code to
    undefinedFunction();
}
catch(err) {                       // block to handle
    console.log(err.message);
}
```

Throw error

```
throw "My error message";    // throw a text
```

Input validation

```
var x = document.getElementById("mynum").value; //
try {
    if(x == "")  throw "empty";                  //
    if(isNaN(x)) throw "not a number";
    x = Number(x);
    if(x > 10)   throw "too high";
}
catch(err) {                                     //
    document.write("Input is " + err);           //
    console.error(err);                          //
}
finally {
    document.write("</br />Done");               //
}
```

Error name values

RangeError	*A number is "out of range"*
ReferenceError	*An illegal reference has occurred*
SyntaxError	*A syntax error has occurred*
TypeError	*A type error has occurred*
URIError	*An encodeURI() error has occurred*

Useful Links ↲

JS cleaner	**Obfuscator**
Can I use?	**Node.js**
jQuery	**RegEx tester**

n+	*contains at least one n*
n*	*contains zero or more occurrences of n*
n?	*contains zero or one occurrences of n*
^	*Start of string*

JSON j

```
var str = '{"names":[' +                    // cr
'{"first":"Hakuna","lastN":"Matata" },' +
'{"first":"Jane","lastN":"Doe" },' +
'{"first":"Air","last":"Jordan" }]}';
obj = JSON.parse(str);                      // pa
document.write(obj.names[1].first);         // ac
```

Send

```
var myObj = { "name":"Jane", "age":18, "city":"Ch
var myJSON = JSON.stringify(myObj);
window.location = "demo.php?x=" + myJSON;
```

Storing and retrieving

```
myObj = { "name":"Jane", "age":18, "city":"Chicag
myJSON = JSON.stringify(myObj);                  /
localStorage.setItem("testJSON", myJSON);
text = localStorage.getItem("testJSON");         /
obj = JSON.parse(text);
document.write(obj.name);
```

Promises Þ

```
function sum (a, b) {
   return Promise(function (resolve, reject) {
     setTimeout(function () {
       if (typeof a !== "number" || typeof b !== "
         return reject(new TypeError("Inputs must
       }
       resolve(a + b);
     }, 1000);
   });
}
var myPromise = sum(10, 5);
myPromsise.then(function (result) {
  document.write(" 10 + 5: ", result);
  return sum(null, "foo");                  // Invalid
}).then(function () {                       // Won't b
}).catch(function (err) {                   // The cat
  console.error(err);                       // => Plea
});
```

States

pending, fulfilled, rejected

Properties

Promise.length, Promise.prototype

Methods

Promise.all(iterable), Promise.race(iterable), Promise.reject(reason), Promise.resolve(value)