ChatGPT

```
// 1. Manipolazione del DOM
// Funzioni per semplificare le operazioni sul DOM (Document Object Model).

// Restituisce un elemento DOM dato il suo ID.
function getById(id) {
    return document.getElementById(id);
}

// Restituisce il primo elemento che corrisponde al selettore CSS specificato
(opzionalmente all'interno di un elemento padre).
function qs(selector, element = document) {
    return element.querySelector(selector);
}

// Restituisce tutti gli elementi che corrispondono al selettore CSS
specificato sotto forma di array.
function qsa(selector, element = document) {
    return Array.from(element.querySelectorAll(selector));
}

// Crea un nuovo elemento DOM con il tag e gli attributi specificati.
function createElement(tag, attributes = {}) {
    const el = document.createElement(tag);
    for (let attr in attributes) {
        if (attr === 'innerText' || attr === 'textContent') {
            el.textContent = attributes[attr];
        } else if (attr === 'innerHTML') {
            el.innerHTML = attributes[attr];
        } else {
            el.setAttribute(attr, attributes[attr]);
        }
    }
    return el;
}

// Aggiunge una classe CSS a un elemento.
function addClass(element, className) {
    if (element.classList) {
        element.classList.add(className);
    } else {
        element.className += ' ' + className;
    }
}

// Rimuove una classe CSS da un elemento.
function removeClass(element, className) {
    if (element.classList) {
```

```
        element.classList.remove(className);
    } else {
        element.className = element.className.replace(new RegExp('(^|\\s)' +
className + '(\\s|$)'), ' ').trim();
    }
}

// Alterna (aggiunge o rimuove) una classe CSS su un elemento.
function toggleClass(element, className) {
    if (element.classList) {
        element.classList.toggle(className);
    } else {
        if (new RegExp('(^|\\s)' + className + '(\\s|
$)').test(element.className)) {
            element.className = element.className.replace(new RegExp('(^|\
\s)' + className + '(\\s|$)'), ' ').trim();
        } else {
            element.className += ' ' + className;
        }
    }
}

// Verifica se un elemento possiede una determinata classe CSS.
function hasClass(element, className) {
    if (element.classList) {
        return element.classList.contains(className);
    } else {
        return new RegExp('(^|\\s)' + className + '(\\s|
$)').test(element.className);
    }
}

// 2. Richieste HTTP/API (AJAX/fetch)
// Funzioni per effettuare chiamate HTTP asincrone al server.

// Effettua una richiesta GET HTTP a un URL e chiama il callback con la
risposta (formato testo).
function httpGet(url, callback) {
    const xhr = new XMLHttpRequest();
    xhr.open('GET', url);
    xhr.onload = function() {
        if (xhr.status === 200) {
            callback(xhr.responseText);
        }
    };
    xhr.onerror = function() {
        console.error('Errore nella richiesta GET', url);
    };
    xhr.send();
}
```

```
// Effettua una richiesta POST HTTP inviando dati JSON e chiama il callback
con la risposta.
function httpPost(url, data, callback) {
    const xhr = new XMLHttpRequest();
    xhr.open('POST', url);
    xhr.setRequestHeader('Content-Type', 'application/json');
    xhr.onload = function() {
        if (xhr.status === 200) {
            callback(xhr.responseText);
        }
    };
    xhr.onerror = function() {
        console.error('Errore nella richiesta POST', url);
    };
    xhr.send(JSON.stringify(data));
}

// Recupera dati JSON da un URL usando fetch e restituisce una Promise.
function fetchJSON(url) {
    return fetch(url).then(response => response.json());
}

// Invia dati JSON a un URL via POST usando fetch e restituisce una Promise
con la risposta JSON.
function postJSON(url, data) {
    return fetch(url, {
        method: 'POST',
        headers: {'Content-Type': 'application/json'},
        body: JSON.stringify(data)
    }).then(response => response.json());
}

// Effettua una richiesta fetch con timeout specificato (in millisecondi). Se
il tempo scade, la Promise viene respinta.
function fetchWithTimeout(url, timeout) {
    const controller = new AbortController();
    const timer = setTimeout(() => controller.abort(), timeout);
    return fetch(url, { signal: controller.signal })
        .then(response => {
            clearTimeout(timer);
            return response;
        });
}

// Carica dinamicamente uno script esterno e invoca un callback al termine
del caricamento.
function loadScript(src, callback) {
    const script = document.createElement('script');
    script.src = src;
    script.onload = callback;
    script.onerror = () => console.error('Impossibile caricare lo script',
```

```
src);
    document.head.appendChild(script);
}

// Esegue richieste HTTP periodiche (polling) all'URL specificato ogni
intervallo di millisecondi.
function poll(url, interval, callback) {
    function request() {
        fetch(url)
            .then(response => response.text())
            .then(data => { callback(data); setTimeout(request, interval); })
            .catch(err => { console.error('Polling error:', err);
setTimeout(request, interval); });
    }
    request();
}

// Costruisce una query string (parametri URL) a partire da un oggetto di
parametri.
function objectToQueryString(params) {
    return '?' + Object.keys(params).map(key => encodeURIComponent(key) +
'=' + encodeURIComponent(params[key])).join('&');
}

// 3. Manipolazione e formattazione delle date
// Funzioni per gestire e formattare oggetti Date.

// Restituisce una stringa formattata secondo il formato specificato (es.
'DD/MM/YYYY').
function formatDate(date, format) {
    const pad = (n) => n.toString().padStart(2, '0');
    return format
        .replace('YYYY', date.getFullYear())
        .replace('MM', pad(date.getMonth() + 1))
        .replace('DD', pad(date.getDate()))
        .replace('HH', pad(date.getHours()))
        .replace('mm', pad(date.getMinutes()))
        .replace('ss', pad(date.getSeconds()));
}

// Aggiunge un certo numero di giorni alla data fornita e restituisce una
nuova data.
function addDays(date, days) {
    const result = new Date(date);
    result.setDate(result.getDate() + days);
    return result;
}

// Aggiunge un certo numero di ore alla data fornita e restituisce una nuova
data.
function addHours(date, hours) {
```

```
    const result = new Date(date);
    result.setHours(result.getHours() + hours);
    return result;
}

// Calcola la differenza in giorni tra due date.
function differenceInDays(date1, date2) {
    const diffTime = Math.abs(date2 - date1);
    return Math.floor(diffTime / (1000 * 60 * 60 * 24));
}

// Verifica se la data fornita cade di sabato o domenica.
function isWeekend(date) {
    const day = date.getDay(); // 0 = Domenica, 6 = Sabato
    return day === 0 || day === 6;
}

// Restituisce il numero di giorni nel mese specificato (anno e mese).
function daysInMonth(year, month) {
    return new Date(year, month + 1, 0).getDate();
}

// Verifica se l'anno specificato è bisestile.
function isLeapYear(year) {
    return (year % 4 === 0 && year % 100 !== 0) || (year % 400 === 0);
}

// Converte un timestamp (millisecondi) in un oggetto Date.
function timestampToDate(timestamp) {
    return new Date(timestamp);
}

// 4. Validazione dei dati
// Funzioni per validare stringhe (email, password, numeri, ecc.).

// Verifica se una stringa è un indirizzo email valido.
function isEmail(str) {
    const pattern = /^[^\s@]+@[^\s@]+\.[^\s@]+$/;
    return pattern.test(str);
}

// Verifica la robustezza di una password (minimo 8 caratteri, maiuscola,
minuscola, numero).
function isStrongPassword(str) {
    const lengthOk = str.length >= 8;
    const hasUpper = /[A-Z]/.test(str);
    const hasLower = /[a-z]/.test(str);
    const hasNumber = /[0-9]/.test(str);
    return lengthOk && hasUpper && hasLower && hasNumber;
}
```

```
// Verifica se una stringa è un URL valido.
function isURL(str) {
    try {
        new URL(str);
        return true;
    } catch (e) {
        return false;
    }
}

// Verifica se una stringa è un numero di telefono valido (contiene solo
cifre e opzionalmente il prefisso +).
function isPhoneNumber(str) {
    return /^\+?\d{7,15}$/.test(str);
}

// Verifica se una stringa rappresenta un colore esadecimale valido (es. #FFF
o #FFFFFF).
function isHexColor(str) {
    return /^#([0-9A-Fa-f]{3}|[0-9A-Fa-f]{6})$/.test(str);
}

// Verifica se il valore è numerico (numero o stringa numerica).
function isNumeric(value) {
    return !isNaN(parseFloat(value)) && isFinite(value);
}

// Verifica se una stringa è vuota o composta solo da spazi bianchi.
function isEmpty(str) {
    return str.trim().length === 0;
}

// Verifica se una stringa può essere convertita in una data valida.
function isDateString(str) {
    const date = new Date(str);
    return !isNaN(date.getTime());
}

// 5. Sicurezza
// Funzioni utili per sanitizzare input ed evitare vulnerabilità XSS.

// Converte i caratteri speciali in HTML (es. <, >, &) nelle relative entità
di escape.
function escapeHTML(str) {
    return str.replace(/[&<>"']/g, function(match) {
        switch (match) {
            case '&': return '&amp;';
            case '<': return '&lt;';
            case '>': return '&gt;';
            case '"': return '&quot;';
            case "'": return '&#39;';
```

```
        }
    });
}

// Converte una stringa con entità HTML di escape nei corrispondenti
caratteri originali.
function unescapeHTML(str) {
    return str.replace(/&amp;|&lt;|&gt;|&quot;|&#39;/g, function(match) {
        switch (match) {
            case '&amp;': return '&';
            case '&lt;': return '<';
            case '&gt;': return '>';
            case '&quot;': return '"';
            case '&#39;': return "'";
        }
    });
}

// Rimuove tutti i tag HTML da una stringa, restituendo solo il testo.
function stripTags(html) {
    return html.replace(/<[^>]*>/g, '');
}

// Rimuove eventuali script HTML (tag <script>) da una stringa di codice
HTML.
function stripScripts(html) {
    return html.replace(/<script[\s\S]*?>[\s\S]*?<\/script>/gi, '');
}

// Verifica se un URL è potenzialmente sicuro (non utilizza protocolli
pericolosi come javascript:).
function isSafeURL(url) {
    return !/^javascript:/i.test(url);
}

// Escape dei caratteri speciali in una stringa da usare in espressioni
regolari.
function escapeRegExp(str) {
    return str.replace(/[.*+?^${}()|[\]\\]/g, '\\$&');
}

// Imposta in modo sicuro il contenuto testuale di un elemento DOM (evitando
interpretazione come HTML).
function setTextSafe(element, text) {
    if (element) {
        element.textContent = text;
    }
}

// Genera un token casuale alfanumerico di lunghezza specificata.
function generateRandomToken(length) {
```

```
    const chars =
'ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789';
    let token = '';
    const randomValues = new Uint8Array(length);
    window.crypto.getRandomValues(randomValues);
    for (let i = 0; i < length; i++) {
        token += chars.charAt(randomValues[i] % chars.length);
    }
    return token;
}

// 6. Utility per array e oggetti
// Funzioni per la manipolazione di array e oggetti JavaScript.

// Restituisce un nuovo array con soli valori unici (rimuove duplicati).
function uniqueArray(arr) {
    return [...new Set(arr)];
}

// Appiattisce un array di array in un unico array (ricorsivamente per
annidamenti profondi).
function flattenArray(arr) {
    const result = [];
    (function flat(a) {
        a.forEach(el => {
            if (Array.isArray(el)) flat(el);
            else result.push(el);
        });
    })(arr);
    return result;
}

// Suddivide un array in sotto-array di lunghezza specificata.
function chunkArray(arr, size) {
    const chunks = [];
    for (let i = 0; i < arr.length; i += size) {
        chunks.push(arr.slice(i, i + size));
    }
    return chunks;
}

// Raggruppa gli oggetti di un array in base a una chiave comune.
function groupBy(arr, key) {
    return arr.reduce((group, obj) => {
        const value = obj[key];
        (group[value] = group[value] || []).push(obj);
        return group;
    }, {});
}

// Verifica se un oggetto non ha proprietà proprie.
```

```
function isEmptyObject(obj) {
    return Object.keys(obj).length === 0;
}

// Crea una copia profonda di un oggetto (JSON serializzabile).
function deepClone(obj) {
    return JSON.parse(JSON.stringify(obj));
}

// Congela un oggetto e i suoi sotto-oggetti per renderli immutabili.
function deepFreeze(obj) {
    Object.keys(obj).forEach(prop => {
        if (typeof obj[prop] === 'object' && obj[prop] !== null) {
            deepFreeze(obj[prop]);
        }
    });
    return Object.freeze(obj);
}

// Ottiene in sicurezza un valore annidato all'interno di un oggetto dato un
percorso (es. "a.b.c").
function getNested(obj, path, defaultValue) {
    return path.split('.').reduce((o, key) => (o && o[key] !== undefined ?
o[key] : undefined), obj) ?? defaultValue;
}

// 7. Funzioni per prestazioni e ottimizzazione
// Funzioni per limitare l'esecuzione frequente di operazioni costose.

// Rimanda l'esecuzione di una funzione finché non trascorre un certo
intervallo di tempo dall'ultimo invoco.
function debounce(func, wait) {
    let timeout;
    return function(...args) {
        const context = this;
        clearTimeout(timeout);
        timeout = setTimeout(() => {
            func.apply(context, args);
        }, wait);
    };
}

// Limita il numero di volte in cui una funzione può essere eseguita in un
intervallo di tempo.
function throttle(func, wait) {
    let lastTime = 0;
    return function(...args) {
        const now = Date.now();
        if (now - lastTime >= wait) {
            lastTime = now;
            func.apply(this, args);
```

```
        }
    };
}

// Ritorna una versione di una funzione che può essere eseguita solo una
volta.
function once(func) {
    let called = false;
    let result;
    return function(...args) {
        if (!called) {
            called = true;
            result = func.apply(this, args);
        }
        return result;
    };
}

// Memorizza i risultati di una funzione costosa in base ai suoi argomenti
per velocizzare le chiamate successive.
function memoize(func) {
    const cache = {};
    return function(...args) {
        const key = JSON.stringify(args);
        if (cache[key] === undefined) {
            cache[key] = func.apply(this, args);
        }
        return cache[key];
    };
}

// Restituisce una Promise che si risolve dopo un certo numero di
millisecondi (utilità per delay).
function sleep(ms) {
    return new Promise(resolve => setTimeout(resolve, ms));
}

// Esegue una funzione in modo asincrono, dopo aver liberato il thread
principale (equivalente a process.nextTick in Node).
function defer(func, ...args) {
    setTimeout(() => func.apply(this, args), 0);
}

// Alias per requestAnimationFrame con un fallback su setTimeout (per
animazioni fluide).
function rAF(callback) {
    return window.requestAnimationFrame ?
window.requestAnimationFrame(callback) : setTimeout(callback, 16);
}

// Misura il tempo di esecuzione di una funzione in millisecondi.
```

```
function measureExecutionTime(func) {
    const start = performance.now();
    func();
    const end = performance.now();
    return end - start;
}

// 8. Funzioni matematiche e randomiche
// Funzioni di utilità per operazioni matematiche e generazione di valori
casuali.

// Restituisce un intero casuale tra min e max (inclusi).
function randomInt(min, max) {
    return Math.floor(Math.random() * (max - min + 1)) + min;
}

// Restituisce un numero in virgola mobile casuale tra min (incluso) e max
(escluso).
function randomFloat(min, max) {
    return Math.random() * (max - min) + min;
}

// Genera un colore casuale in formato esadecimale (es. "#A1B2C3").
function randomColor() {
    const hex = Math.floor(Math.random() *
0xFFFFFF).toString(16).padStart(6, '0');
    return '#' + hex;
}

// Limita un numero all'interno di un intervallo (se minore del minimo
restituisce il minimo, se maggiore del massimo restituisce il massimo).
function clamp(num, min, max) {
    return Math.min(Math.max(num, min), max);
}

// Arrotonda un numero al numero di decimali specificato.
function roundTo(num, decimals) {
    const factor = 10 ** decimals;
    return Math.round(num * factor) / factor;
}

// Converte gradi in radianti.
function degToRad(deg) {
    return (deg * Math.PI) / 180;
}

// Converte radianti in gradi.
function radToDeg(rad) {
    return (rad * 180) / Math.PI;
}
```

```javascript
// Calcola la distanza tra due punti in un piano cartesiano.
function distance(x1, y1, x2, y2) {
    const dx = x2 - x1;
    const dy = y2 - y1;
    return Math.sqrt(dx * dx + dy * dy);
}

// 9. Funzioni di logging e debug
// Funzioni utili per tracciare e formattare output di debug.

// Stampa un messaggio di log solo se la modalità di debug è attiva.
let DEBUG_MODE = false;
function debugLog(msg) {
    if (DEBUG_MODE) {
        console.log('DEBUG:', msg);
    }
}

// Stampa un oggetto in formato JSON leggibile (con indentazione).
function logJSON(obj) {
    console.log(JSON.stringify(obj, null, 2));
}

// Genera un errore se la condizione specificata non è soddisfatta (usato per
debug).
function assert(condition, message) {
    if (!condition) {
        throw new Error(message || 'Assert fallito');
    }
}

// Stampa un messaggio nel log con timestamp ISO corrente.
function logWithTimestamp(msg) {
    const timestamp = new Date().toISOString();
    console.log(timestamp + ' - ' + msg);
}

// Stampa un messaggio di errore nel log della console.
function logError(error) {
    console.error('Errore:', error);
}

// Visualizza un array o oggetto in formato tabellare (se supportato dalla
console).
function logTable(data) {
    console.table(data);
}

// 10. Utility di tipo
// Funzioni per controllare il tipo di variabili e valori.
```

```
// Controlla se un valore è di tipo stringa.
function isString(value) {
    return typeof value === 'string' || value instanceof String;
}

// Controlla se un valore è di tipo numero (escludendo NaN).
function isNumber(value) {
    return typeof value === 'number' && !isNaN(value);
}

// Controlla se un valore è un array.
function isArray(value) {
    return Array.isArray(value);
}

// Controlla se un valore è una funzione.
function isFunction(value) {
    return typeof value === 'function';
}

// Controlla se un valore è un oggetto semplice (non null e non array).
function isObject(value) {
    return value !== null && typeof value === 'object' && !
Array.isArray(value);
}

// Controlla se un valore è booleano.
function isBoolean(value) {
    return typeof value === 'boolean';
}

// Controlla se un valore è null o undefined.
function isNullOrUndefined(value) {
    return value === null || value === undefined;
}

// Restituisce il tipo del valore come stringa (gestisce null e array
correttamente).
function getType(value) {
    if (value === null) return 'Null';
    if (Array.isArray(value)) return 'Array';
    return typeof value;
}

// 11. Conversioni di formato
// Funzioni per convertire dati tra vari formati (JSON, Base64, query string,
ecc.).

// Converte un oggetto JavaScript in una stringa JSON.
function toJSON(obj) {
    return JSON.stringify(obj);
```

```
}

// Converte una stringa JSON in un oggetto JavaScript (ritorna null se
fallisce).
function fromJSON(jsonStr) {
    try {
        return JSON.parse(jsonStr);
    } catch (e) {
        return null;
    }
}

// Codifica una stringa in Base64.
function toBase64(str) {
    return btoa(str);
}

// Decodifica una stringa da Base64.
function fromBase64(str) {
    return atob(str);
}

// Converte una query string (es. "?a=1&b=2") in un oggetto con chiave/
valore.
function queryStringToObject(query) {
    const result = {};
    if (!query) return result;
    if (query[0] === '?') query = query.substring(1);
    query.split('&').forEach(param => {
        const [key, value = ''] = param.split('=');
        result[decodeURIComponent(key)] = decodeURIComponent(value);
    });
    return result;
}

// Converte un oggetto in una query string (es. {a:1,b:2} -> "?a=1&b=2").
function objectToQuery(obj) {
    return '?' + Object.keys(obj).map(key => encodeURIComponent(key) + '=' +
encodeURIComponent(obj[key])).join('&');
}

// Converte un colore esadecimale (es. "#FF8800") in un oggetto con
componenti RGB.
function hexToRgb(hex) {
    hex = hex.replace('#', '');
    if (hex.length === 3) {
        hex = hex.split('').map(c => c + c).join('');
    }
    const num = parseInt(hex, 16);
    const r = (num >> 16) & 255;
    const g = (num >> 8) & 255;
```

```
    const b = num & 255;
    return { r: r, g: g, b: b };
}

// Converte valori RGB (componenti 0-255) in una stringa colore esadecimale
("#RRGGBB").
function rgbToHex(r, g, b) {
    const toHex = (val) => val.toString(16).padStart(2, '0');
    return '#' + toHex(r) + toHex(g) + toHex(b);
}

// 12. Funzioni di supporto per animazioni/CSS
// Funzioni per manipolare stili CSS e creare semplici animazioni.

// Mostra un elemento (display block).
function showElement(element) {
    element.style.display = 'block';
}

// Nasconde un elemento (display none).
function hideElement(element) {
    element.style.display = 'none';
}

// Alterna la visibilità di un elemento.
function toggleVisibility(element) {
    const style = window.getComputedStyle(element);
    element.style.display = (style.display === 'none') ? 'block' : 'none';
}

// Effettua una transizione graduale in entrata (fade in) di un elemento.
function fadeIn(element, duration) {
    element.style.opacity = 0;
    element.style.display = 'block';
    let opacity = 0;
    const interval = 50;
    const increment = 1 / (duration / interval);
    const fade = setInterval(() => {
        opacity += increment;
        if (opacity >= 1) {
            opacity = 1;
            clearInterval(fade);
        }
        element.style.opacity = opacity;
    }, interval);
}

// Effettua una transizione graduale in uscita (fade out) di un elemento.
function fadeOut(element, duration) {
    let opacity = parseFloat(window.getComputedStyle(element).opacity) || 1;
    const interval = 50;
```

```
    const decrement = opacity / (duration / interval);
    const fade = setInterval(() => {
        opacity -= decrement;
        if (opacity <= 0) {
            opacity = 0;
            clearInterval(fade);
            element.style.display = 'none';
        }
        element.style.opacity = opacity;
    }, interval);
}

// Scorre dolcemente la pagina fino all'inizio (scroll top).
function scrollToTop() {
    window.scrollTo({ top: 0, behavior: 'smooth' });
}

// 13. Funzioni per la manipolazione di stringhe
// Funzioni per modificare e formattare stringhe di testo.

// Rende maiuscola la prima lettera di una stringa.
function capitalize(str) {
    if (!str) return '';
    return str.charAt(0).toUpperCase() + str.slice(1);
}

// Rende maiuscola la prima lettera di ogni parola in una stringa.
function capitalizeWords(str) {
    return str.split(' ').map(word => word.charAt(0).toUpperCase() +
word.slice(1)).join(' ');
}

// Converte una stringa in formato camelCase.
function toCamelCase(str) {
    return str.toLowerCase().replace(/\s+(.)/g, (match, chr) =>
chr.toUpperCase());
}

// Converte una stringa in formato kebab-case.
function toKebabCase(str) {
    return str.replace(/\s+/g, '-').toLowerCase();
}

// Tronca una stringa oltre una certa lunghezza e aggiunge "...".
function truncate(str, maxLength) {
    if (str.length <= maxLength) return str;
    return str.slice(0, maxLength - 3) + '...';
}

// Inverte i caratteri di una stringa.
function reverseString(str) {
```

```
    return str.split('').reverse().join('');
}

// Conta quante volte una sottostringa appare all'interno di una stringa.
function countOccurrences(str, substring) {
    if (substring === '') return 0;
    return str.split(substring).length - 1;
}

// Rimuove gli spazi extra (multipli) in una stringa, lasciando singoli
spazi.
function removeExtraSpaces(str) {
    return str.trim().replace(/\s+/g, ' ');
}
```
