Javascript Avanzato: Appunti

Peculiarità di Javascript

Valori Falsy e Truthy

- **Falsy:** Valori che, in un contesto booleano, vengono valutati come `false`.
  - `false`
  - `0`
  - `null`
  - `undefined`
  - `""` (stringa vuota)
  - `NaN`
- **Truthy:** Tutti gli altri valori, inclusi:
  - Stringhe non vuote (`"any non-empty string"`)
  - Numeri come `3.14`, `Infinity`
  - Oggetti vuoti `{}`
  - Array vuoti `[]`
  - Stringhe `"0"`, `"undefined"`, `"null"`

Implicazioni:

- Semplificazione delle condizioni: `if (value)` invece di `if (value != null && value.length > 0)`
- Inizializzazione di variabili: `misura = (param || '12') + 'px'` invece di `if (param== null || param==0) { misura = "12px" ; } else { misura = param + 'px'}`
- controllo esistenza di variabili, librerie o servizi, es: `if (window.XMLHttpRequest) { ... }`
- inizializzazione di parametri opzionali di default, es: `hostname = hostname || "localhost";`

Funzioni come Entità di Prima Classe

- Le funzioni sono oggetti:
  - Assegnabili a variabili: `let potenza = function(a,b) {return Math.pow(a,b);}`
  - Passabili come parametri: `setTimeout(function() { ... }, 1000)`
  - Restituibili da altre funzioni: `let expGenerator = function(e) { return function(b) { return Math.pow(b,e) } }`
  - Memorizzabili come proprietà di oggetti o array
  - Invocabili con `()`
- **Function statement vs. Function expression:**

```
function potenza(a,b) {return Math.pow(a,b);} // Function statement
let potenza = function(a,b) {return Math.pow(a,b);} // Function expression
```

- **`bind(obj, args)`:** Associa una funzione a un oggetto (`this`) e argomenti specifici.

Funzioni Filtro su Array

- Metodi che accettano funzioni come parametro per operazioni sugli array:
  - `sort(f)`: Ordinamento. `f` restituisce `1` (mantiene l'ordine) o `-1` (inverte l'ordine).

- `filter(f)` : Crea un nuovo array con elementi che soddisfano `f` (booleana).
- `some(f)` , `every(f)` : Verificano se almeno uno o tutti gli elementi soddisfano `f` .
- `find(f)` : Restituisce il primo elemento che soddisfa `f` .
- `forEach(f)` : Esegue `f` su ogni elemento, modificando l'array originale.
- `map(f)` : Crea un nuovo array applicando `f` a ogni elemento.
- `reduce(f)` : Applica `f` cumulativamente, utile per totali.

Funzioni Freccia (Arrow Functions)

- Sintassi compatta per definire funzioni:

```
    let square = (x) => x * x; // Senza graffe e return (una sola istruzione)
    let square = (x) => {return x * x; } // Con graffe e return.
    let c = (x => x * x)(5); // IIFE
```

- Utili per callback semplici (filtri array, eventi).

```
    let squares = arr.map(x => x * x); // map con arrow function
    showHelp.onclick = () => helpDiv.classList.toggle('d-none') // arrow function
per eventi
```

Object Orientedness in Javascript

Oggetti e Classi

- Javascript è object-oriented ma basato su **prototipi**, non su classi.
- Le funzioni possono essere contenute negli oggetti senza ricorrere alla classe.
- Istanziazione di oggetti:
  - Dichiarazione diretta del contenuto.
  - Costruttore (funzione che restituisce un oggetto, invocata con `new`).

Classi in Javascript

- Non sono entità di primo livello.
- Si usano oggetti provenienti dallo stesso costruttore.
- Costruttore: Funzione che restituisce un oggetto (usare `new`).

```
    function Persona(nome,altezza,nascita){  // Costruttore
        this.nome = nome
        this.altezza = altezza
        this.nascita = nascita
        this.saluta = function() { return 'ciao!' }
    }
    let mario = new Persona(“Mario”, 185, new Date(2002,3,14));
```

`this` in Javascript

- Riferimento a un oggetto:
  - Dentro una classe: Istanza dell'oggetto.
  - Callback di un evento: Oggetto che ha ricevuto l'evento.

- `bind()` esplicito: Oggetto specificato.
- Altrimenti: `window` (browser) o `module.exports` (server-side).

Prototype

- Ogni oggetto ha una proprietà `.prototype`.
- Aggiunta di membri al `prototype` per condividerli tra oggetti.
- Utile per estendere oggetti built-in o creati in precedenza.
- Le modifiche al `prototype` influenzano anche le istanze già create.
- Modificare il prototype di classi esistenti: controverso, namespace pollution. Ma utile per estendere funzionalità, attenzione alla collisione di nomi.

Scope delle Variabili, Closure e IIFE

Scope delle Variabili

- Quattro tipi di scope:
  - **Globale:** Variabile definita esternamente alle funzioni (anche senza `var`, `let`, `const`). Nel browser, sono membri di `window`.
  - **Modulo:** Variabile definita in un file modulo (con `export`).
  - **Funzione:** Variabile definita con `var` dentro una funzione.
  - **Blocco:** Variabile definita con `let` o `const` dentro un blocco `{}`.

Closure

- Quinto tipo di scope: Scope della funzione in cui è definita un'altra funzione.
- Permette di creare variabili "private" accessibili solo alla funzione interna.

```
Counter = function() {
    var state = 0; // Variabile privata
    return {
        incrementa: function() { return ++state },
        decrementa: function() { return --state }
    }
}
```

IIFE (Immediately Invoked Function Expression)

- Funzione anonima creata e invocata immediatamente.
- Utilizzata per creare singleton con stato interno privato (grazie alla closure).

```
var people = (function() {
    var persone = [];
    return {
        add:   function(p) { persone.push(p)},
        lista: function(){ return persone.join(', ') }
    }
})()
```

Altri Aspetti di Javascript

Definizioni di Classe (Zucchero Sintattico)

- Modo compatto per creare oggetti, simile a Java/C++.

```
// sintassi con classi, più compatta
    class Shape {
        constructor (id,x,y) {
            this.id= id
            this.x = x ;
            this.y= y;
        }
        move (x,y) {
            this.x= x ;
            this.y= y ;
        }
    }
```

Template Literal

- Stringhe multi-linea con interpolazione di variabili.
- Delimitatori: backtick (`\`).
- Interpolazione: `${varName}` .

```
    let x = `Hello ${firstName}!
    How are you
    today?`;
```

Optional Chaining (ES2020)

- Operatore `?.` per accedere a proprietà annidate senza errori se un elemento è `undefined` .

```
    console.log( persone[i]?.indirizzo?.via?.numero ) // Restituisce undefined
invece di un errore
```

Operatore Spread (`...`)

- "Spalma" elementi di array, oggetti, iterabili.
- Utile per:
  - Concatenare array: `let a3 = [...a1, ...a2]`
  - Unire/clonare oggetti: `let fv2 = { ...fv, professione: "docente" }`
  - Passare elementi come argomenti: `let myFullName = fullName(...fvAsArray)`