Formulario Completo - Programmazione Web

Indice

Introduzione

Questo documento raccoglie i formulari completi per HTML, CSS e JavaScript, utili per la preparazione all'esame di Programmazione Web.

1 Formulario HTML

Introduzione

Questi appunti raccolgono i principali comandi HTML studiati a lezione, organizzati in modo chiaro e accessibile per facilitare lo studio.

Tag HTML e Descrizione

Tag	Sezione (Dove va usato)	Descrizione
`<html>`	Radice del documento	Contiene tutto il codice HTML della pagina.
`<head>`	Dentro `<html>`	Contiene metadati, titolo, link a CSS e script.
`<title>`	Dentro `<head>`	Definisce il titolo della scheda del browser.
`<meta>`	Dentro `<head>`	Fornisce informazioni sulla pagina (charset, descrizione, autore, ecc.).
`<link>`	Dentro `<head>`	Collega file CSS esterni o altre risorse.
`<script>`	Dentro `<head>` o `<body>`	Inserisce codice JavaScript.
`<style>`	Dentro `<head>`	Definisce regole CSS direttamente nel documento.
`<body>`	Dentro `<html>`	Contiene tutto il contenuto visibile della pagina.
`<header>`	Dentro `<body>`	Intestazione della pagina o di una sezione, spesso contiene logo e menu.
`<nav>`	Dentro `<header>` o `<body>`	Contiene i link di navigazione del sito.
`<main>`	Dentro `<body>`	Contiene il contenuto principale della pagina.
`<section>`	Dentro `<main>` o `<body>`	Raggruppa contenuti correlati.
`<article>`	Dentro `<section>` o `<main>`	Contiene contenuti indipendenti (articoli, blog, news).
`<aside>`	Dentro `<body>` o `<main>`	Contiene contenuti secondari come barre laterali o widget.
`<footer>`	Dentro `<body>`	Piè di pagina della pagina, spesso con contatti o copyright.
`<h1>{<h6>`	Dentro `<body>`	Titoli della pagina, da più grande (`<h1>`) a più piccolo (`<h6>`).
`<p>`	Dentro `<body>`	Definisce un paragrafo di testo.
`<a>`	Dentro `<body>`	Crea un link ad altre pagine o sezioni.
`<img>`	Dentro `<body>`	Inserisce un'immagine.
` `	Dentro `<body>` o `<p>`	Va a capo (interruzione di linea).

`<hr>`	Dentro `<body>`	Inserisce una linea orizzontale per separare sezioni.
`<ul>`	Dentro `<body>`	Crea una lista non ordinata (punti elenco).
`<ol>`	Dentro `<body>`	Crea una lista ordinata (numerata).
`<li>`	Dentro `<ul>` o `<ol>`	Definisce un elemento della lista.
`<table>`	Dentro `<body>`	Crea una tabella.
`<tr>`	Dentro `<table>`	Definisce una riga della tabella.
`<td>`	Dentro `<tr>`	Definisce una cella della tabella.
`<th>`	Dentro `<tr>`	Definisce una cella di intestazione della tabella.
`<form>`	Dentro `<body>`	Crea un modulo per l'invio di dati.
`<input>`	Dentro `<form>`	Crea un campo di input.
`<button>`	Dentro `<form>` o `<body>`	Crea un pulsante cliccabile.
`<iframe>`	Dentro `<body>`	Incorpora una pagina web dentro un'altra.

Esempi Utili

Inserire un'immagine

```
<img src="miafoto.jpg" alt="Descrizione dell'immagine">
```

Inserire un link

```
<a href="https://www.google.com">Visita Google</a>
```

Aprire un link in una nuova scheda

```
<a href="https://www.google.com" target="_blank">Apri Google in una nuova scheda</a>
```

2 Formulario CSS

2.1 Inserire CSS

- **inline**: attributo `style="..."` direttamente nel tag.
- **interno**: dentro `<style>` nel `<head>`.
- **esterno**: file `.css` collegato con `<link rel="stylesheet" href="style.css">`.

Specificità: inline > interno > esterno. `!important` aumenta la priorità.

2.2 Selettori

2.2.1 Universale e base

Selettore	Descrizione
`*`	Seleziona tutti gli elementi
`tag`	Seleziona tutti gli elementi di tipo `tag`
`.classe`	Seleziona elementi con classe
`#id`	Seleziona elemento con ID

2.2.2 Combinatori

`A B`	Discendenti: tutti gli `B` dentro `A`
`A > B`	Figli diretti: `B` figlio di `A`
`A + B`	Fratello immediato: `B` subito dopo `A`
`A ~ B`	Fratelli generici: tutti i `B` dopo `A`

2.2.3 Pseudo-classi

`:link`, `:visited`	Stati dei link
`:hover`, `:active`, `:focus`	Interazioni utente
`:first-child`, `:first-letter`, `:first-line`	Selettori di posizione e testo
`:lang(...)`	Lingua specifica di un elemento

2.2.4 Pseudo-elementi

`::before`, `::after`	Contenuto virtuale prima/dopo
`::first-letter`, `::first-line`	Stilizza parte di testo

2.2.5 Attributi

```
[A]    /* attributo presente */
[A=V]  /* valore esatto */
[A^=V] /* inizia con V */
[A$=V] /* finisce con V */
[A*=V] /* contiene V */
```

2.3 Specificità e Cascade

Specificità = (inline, #id, .classe/pseudo-classe/attributo, tag/pseudo-elemento). La cascade risolve conflitti: ultima regola con specificità maggiore vince. `!important` fornisce massima priorità.

2.4 Box Model

Ogni elemento ha: margin, border, padding, content. `box-sizing: content-box|border-box;`

2.4.1 Margini

- Collasso: margini verticali degli elementi adiacenti si uniscono.
- Non si può colorare il `margin`.

2.5 Tipografia

```
font-family: Arial, sans-serif;
font-size: 16px; line-height: 1.5;
font: italic small-caps bold 16px/1.5 Tahoma;
color: #123456 | rgb(...) | hsl(...) | rgba(..., 0.5);
text-decoration: none | underline | overline | line-through;
@font-face {
  font-family: 'Custom';
  src: url('Custom.woff2') format('woff2'),
       url('Custom.woff') format('woff');
  font-weight: 400;
  font-style: normal;
}
```

2.6 Background

```
background-color: lightgray;
background-image: url('bg.jpg') | linear-gradient(...);
background-repeat: repeat | no-repeat | repeat-x | repeat-y;
background-position: center center;
background-size: auto | cover | contain | 100px 200px;
background-attachment: scroll | fixed | local;
```

2.7 Display & Position

```
display: inline|block|none|flex|grid;
position: static|relative|absolute|fixed|sticky;
top: 10px; right: 0; bottom: 5px; left: 20px;
float: left | right | none;
clear: none | left | right | both;
```

2.8 Flexbox

```
display: flex;
flex-direction: row|column;
flex-wrap: nowrap|wrap;
justify-content: flex-start|center|space-between|...;
align-items: stretch|center|flex-start|...;
align-content: stretch|center|...;
align-self: auto|center|flex-end|...;
gap: 10px;
flex-grow: 1;
flex-shrink: 1;
flex-basis: auto | 100px;
```

2.9 Grid Layout

```
display: grid;
grid-template-columns: repeat(3, 1fr) | 100px auto;
grid-template-rows: auto 100px;
grid-template-areas:
  "hdr hdr"
  "main side";
grid-area: main;
grid-row: 1 / 3;
grid-column: 2 / 4;
jus tify-items: center;
align-items: start;
align-content: space-between;
gap: 20px;
```

2.10 Responsive Design

Meta viewport:

```
<meta name="viewport" content="width=device-width, initial-scale=1.0">
```

2.10.1 Media Queries

```
@media (max-width: 768px) {
  .container { display: block; }
}
@media (min-width: 1024px) {
```

```
  body { font-size: 18px; }
}
```

Strategie: Mobile First (min-width) vs Desktop First (max-width).

2.11 CSS Variables

```
:root { --main-color: #3498db; }
.element { color: var(--main-color, black); }
```

2.12 Bootstrap Grid

Sistema a 12 colonne: `.container`, `.row`, `.col-6`, `.col-md-4`.

3 Formulario JavaScript

3.1 Promise & Asincronia

3.1.1 Creazione di una Promise con delay

```
console.log("ciao");
const promiseDelay = new Promise((resolve, reject) => {
  setTimeout(() => {
    // resolve("Hai vinto!!"); // successo
    reject("Hai perso :( ");      // errore dopo 5s
  }, 5000);
});

promiseDelay
  .then(result => console.log(result))
  .catch(console.error);
```

3.1.2 Promise da evento click

```
const bottone = document.querySelector(".myButton");
const result = document.querySelector(".result");

function buttonExecutor(resolve, reject) {
  bottone.addEventListener('click', resolve);
  setTimeout(reject, 5000);
}

new Promise(buttonExecutor)
  .then(() => result.innerHTML = "Hai cliccato in tempo")
  .catch(() => result.innerHTML = "Non hai fatto in tempo");
```

3.2 Eventi & Interazione DOM

3.2.1 Selezione Elementi

```
// CSS selector
const elem = document.querySelector(".classe");
const list = document.querySelectorAll("p.warning");

// By id, tag, class
const byId = document.getElementById("id");
const byTag = document.getElementsByTagName("div");
const byClass = document.getElementsByClassName("item");
```

3.2.2 Gestione Eventi

```
// Via addEventListener
elem.addEventListener("click", function(evt) {
  console.log(evt.target);
});
```

3.2.3 Creazione e Manipolazione Nodo

```
// Creare elemento
const div = document.createElement("div");
const text = document.createTextNode("Ciao");
div.appendChild(text);
parent.appendChild(div);

// Attributi
div.setAttribute("id", "mioDiv");
const id = div.getAttribute("id");

// Rimuovere Nodo
parent.removeChild(div);
```

3.3 Fetch API

```
fetch("http://127.0.0.1:5500/data.txt")
  .then(resp => {
    if (resp.ok) return resp.text();
    throw new Error("Errore HTTP" + resp.status);
  })
  .then(data => document.querySelector(".container").innerHTML =
    ↪ data)
  .catch(console.error);
```

3.4 Async/Await

```
async function fetchData() {
  try {
    const resp = await fetch("https://api.example.com/data");
    if (!resp.ok) throw new Error(resp.statusText);
    const json = await resp.json();
    console.log(json);
  } catch (err) {
    console.error("Errore nella richiesta:", err);
  } finally {
    console.log("Operazione completata");
  }
}

fetchData();
```

3.5 Node.js - Server HTTP Base

```
// file: server.js
const http = require("http");
const fs = require("fs");
const path = require("path");
const mylog = require("./log");

const server = http.createServer((req, res) => {
  console.log("Richiesta:", req.url);

  if (req.url === "/data.txt") {
    const filePath = path.join(__dirname, "data.txt");
    fs.readFile(filePath, "utf8", (err, data) => {
      if (err) {
        res.writeHead(404, {"Content-Type": "text/plain"});
        res.end("File non trovato");
      } else {
        res.writeHead(200, {"Content-Type": "text/plain"});
        res.end(data);
      }
    });
  } else {
    res.writeHead(404, {"Content-Type": "text/plain"});
    res.end("Risorsa non trovata");
  }
});

const PORT = 8080;
server.listen(PORT, () => mylog("Server in ascolto sulla porta: "
  ↪  + PORT));
```

3.6 File System (fs)

3.6.1 Lettura Sincrona

```
const fs = require("fs");
const data = fs.readFileSync("data.txt", "utf-8");
console.log(data);
```

3.6.2 Lettura Asincrona

```
fs.readFile("data.txt", "utf-8", (err, data) => {
  if (err) console.error(err.message);
  else console.log(data);
});
```

3.7 Moduli CommonJS

```
// log.js
const mylog = txt => console.log("mylog " + txt);
const error = err => console.error(new Date().toISOString() + "
   ↪ ERRORE " + err);

exports.mylog = mylog;
exports.error = error;

// oppure
module.exports = mylog;
```

3.8 Moduli di Terze Parti (NPM)

```
npm init
npm install express moment chalk dotenv betterlog
npm install --save-dev nodemon
```

3.9 Express - Routing e API JSON

```
const express = require("express");
const fs = require("fs");
const app = express();
const PORT = 8080;

// Body parsing
app.use(express.json());

// Rotte base
app.get("/", (req, res) => res.send("Ciao yuki"));
app.get("/about", (req, res) => res.send("Pagina About"));

// Parametri URL
app.get("/users/:id", (req, res) => {
  const id = req.params.id;
  res.send(`User ID: ${id}`);
});

// POST
app.post("/contact", (req, res) => res.status(201).send("Contatto
   ↪  creato"));

// JSON API
const data = JSON.parse(fs.readFileSync("prof2.json"));
app.get("/api/v1/profs", (req, res) => res.json(data));
app.get("/api/v1/profs/:id", (req, res) => {
  const prof = data.find(p => p.id === +req.params.id);
  prof
```

```
    ? res.json(prof)
    : res.status(404).json({message: "Data not found ;)"});
});

app.listen(PORT, () => console.log(`Server in ascolto su porta ${
   ↪ PORT}`));
```

3.10 HTTP & REST

- **Metodi HTTP**: GET, POST, PUT, DELETE, PATCH, OPTIONS
- **Codici di stato**: 2xx (successo), 4xx (errore client), 5xx (errore server)
- **Header CORS**: `Access-Control-Allow-Origin: *`, `Access-Control-Allow-Methods: GET, POST, OPTIONS`, `Access-Control-Allow-Headers: Content-Type`
- **Principi REST**:
  1. Identificazione risorse via URI
  2. Interfaccia uniforme (CRUD + HTTP)
  3. Comunicazione stateless
  4. Rappresentazioni (JSON)
  5. HATEOAS (link tra risorse)

3.11 Snippet Utili di Base

3.11.1 Console & Debug

```
console.log("Messaggio");
console.error("Errore");
console.warn("Attenzione");
console.table([{a:1, b:2}]);
```

3.11.2 Interazione Utente

```
alert("Ciao!");
let nome = prompt("Come ti chiami?");
let ok = confirm("Sei sicuro?");
```

3.11.3 Coercizione e parsing

```
let n = parseInt("42");
let f = parseFloat("3.14");
let s = (123).toString();
```

3.11.4 Creazione Rapida Server Node.js

```
# one-liner con npx
npx http-server . -p 8080
```

3.11.5 IIFE (Immediately Invoked Function Expression)

```
(function(){
  console.log("Scoped!");
})();
```

Buono studio e in bocca al lupo per l'esame!