I Canvas - TypeScript

Docente: Prof. Mario Perna **Email:** prof.perna.mario@darzo.net **Revisione:** 04.02.2025

I Canvas in TypeScript

La rappresentazione grafica con Canvas si basa sull'uso dell'elemento HTML **<canvas>** e dell'interfaccia **CanvasRenderingContext2D**.

Questo approccio permette di creare e manipolare grafiche bidimensionali in un contesto dinamico e interattivo, come **grafici, animazioni e giochi**.

Struttura di base

TypeScript, essendo un **superset** di **JavaScript**, offre vantaggi significativi come la tipizzazione statica e il supporto per i modelli OOP (Object-Oriented Programming), rendendo il codice più leggibile e manutenibile.

Struttura di base

1. Creazione del canvas nel DOM: tramite l'elemento **<canvas>** nel file HTML.

```
<body>
    <canvas id="myCanvas" width="500" height="500"></canvas>
    <script src="script.js"></script>
</body>
```

Struttura di base

2. Configurazione del canvas in TypeScript:
   - Recupera il contesto di rendering 2D per disegnare. E' grazie al **getContext** se possiamo accedere ai metodi necessari per il disegno.
   - Usa tipi TypeScript per ottenere suggerimenti e verifiche durante la scrittura del codice.

```
const canvas = document.getElementById("myCanvas") as HTMLCanvasElement;
const ctx = canvas.getContext("2d");

if (!ctx) {
    throw new Error("Il contesto 2D non è disponibile!");
}
```

Funzionalità di base dei Canvas

Una volta ottenuto il contesto, puoi utilizzare metodi come **fillRect, beginPath, moveTo, lineTo** e altri per disegnare forme, immagini e testo.

1. Raffigurazione di un RETTANGOLO

```
ctx!.beginPath();
ctx!.fillStyle = "blue"; // Colore del riempimento
// Posizione (x, y) e dimensioni (larghezza, altezza)
ctx!.fillRect(50, 50, 200, 100);
ctx!.closePath();
```

Funzionalità di base dei Canvas

1. Raffigurazione di un RETTANGOLO

Funzionalità di base dei Canvas

2. Raffigurazione di un CERCHIO

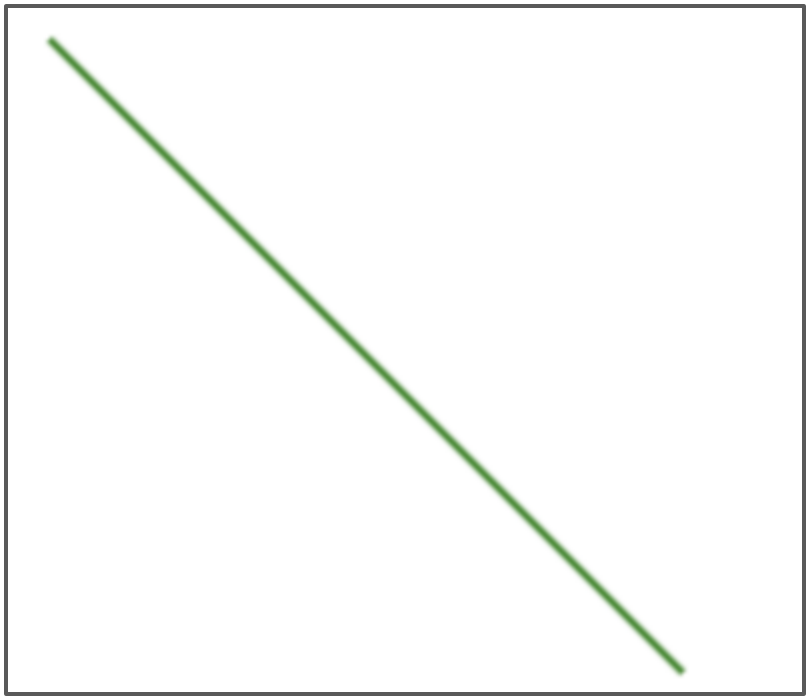
Funzionalità di base dei Canvas

3. Raffigurazione del cerchio

```
ctx.arc(100, 100, 50, 0, Math.PI * 2);
ctx.fillStyle = "red";
ctx.fill(); // Riempie il cerchio del colore definito in fillStyle
ctx.strokeStyle = "black";
ctx.lineWidth = 2; // Spessore del bordo
ctx.stroke();
```

Funzionalità di base dei Canvas

3. Raffigurazione di LINEE



Funzionalità di base dei Canvas

3. Raffigurazione di LINEE

```
ctx!.beginPath();
ctx!.moveTo(10, 10); // Punto iniziale
ctx!.lineTo(200, 200); // Punto finale
ctx!.strokeStyle = "green"; // Colore della linea
ctx!.lineWidth = 2;
ctx!.stroke();
ctx!.closePath();
```

Uso di TS per migliorare il codice

1. Tipizzazione di funzioni La tipizzazione migliora la leggibilità e aiuta a evitare errori.

```
function drawRectangle(ctx: CanvasRenderingContext2D,
                       x: number, y: number, width: number,
                       height: number, color: string): void {
    ctx.fillStyle = color;
    ctx.fillRect(x, y, width, height);
}

drawRectangle(ctx!, 20, 30, 150, 100, "purple");
```

Uso di TS per migliorare il codice

2. Classi per gestire le entità Con TypeScript puoi creare classi per modellare oggetti grafici.

```
class Circle {
    constructor(public x: number, public y: number,
                public radius: number, public color: string) {}

    draw(ctx: CanvasRenderingContext2D): void {
        ctx.beginPath();
        ctx.arc(this.x, this.y, this.radius, 0, Math.PI * 2);
        ctx.fillStyle = this.color;
        ctx.fill();
        ctx.closePath();
    }
}

const circle = new Circle(100, 100, 40, "orange");
circle.draw(ctx!);
```

Uso di TS per migliorare il codice

3. Gestione delle animazioni TypeScript consente di creare animazioni con un loop di rendering.

```
let x = 0;

function animate(): void {
    // Pulisci il canvas
    ctx!.clearRect(0, 0, canvas.width, canvas.height);
    ctx!.fillStyle = "cyan";
    ctx!.fillRect(x, 100, 50, 50); // Disegna un rettangolo
    x += 2; // Incrementa la posizione
    requestAnimationFrame(animate); // Richiama l'animazione
}

animate();
```

Uso di requestAnimationFrame()

Il metodo **requestAnimationFrame()** è una funzione nativa del browser che consente di eseguire animazioni in modo fluido e sincronizzato con il refresh rate dello schermo.

È una soluzione moderna e più efficiente rispetto all'uso di **setInterval()** o **setTimeout()**, poiché ottimizza il consumo di risorse e garantisce migliori prestazioni.

Uso di requestAnimationFrame()

- Quando si richiama **requestAnimationFrame()**, indica al browser di eseguire una funzione di callback al prossimo frame di animazione.
- La funzione verrà eseguita prima del prossimo repaint dello schermo, che di solito avviene a 60 fps (frame al secondo), ovvero ogni ~16.67 millisecondi.
- Se il browser è occupato (ad esempio, la scheda è in background), l'esecuzione viene sospesa, riducendo il carico della CPU.

Uso di requestAnimationFrame()

Movimento di una forma geometrica.

```
const canvas = document.getElementById("myCanvas") as HTMLCanvasElement;
const ctx = canvas.getContext("2d")!;

let x = 0;

function animate() {
  ctx.clearRect(0, 0, canvas.width, canvas.height); // Pulisce il canvas
  ctx.fillStyle = "red";
  ctx.fillRect(x, 50, 50, 50); // Disegna il quadrato

  x += 2; // Sposta il quadrato verso destra

  requestAnimationFrame(animate); // Chiama di nuovo la funzione
}

animate(); // Avvia l'animazione
```

Rappresentazione di un grafico a torta

1. Dividere i dati in percentuali.
2. Calcolare gli angoli di ciascun settore (usando i gradi o i radianti).
3. Disegnare gli spicchi uno alla volta.
4. Aggiungere opzioni di stile come colori o etichette.

Rappresentazione di un grafico a torta

1. File HTML

```
<body>
    <canvas id="grafico" width="500" height="500"></canvas>
    <script src="script.js"></script>
</body>
```

Rappresentazione di un grafico a torta

2. Definizione delle variabili necessari e controllo degli errori.

```
const canvas = document.getElementById("grafico") as HTMLCanvasElement;
const ctx = canvas.getContext("2d");

if (!ctx) {
    throw new Error("Il contesto 2D non è disponibile!");
}
```

Rappresentazione di un grafico a torta

3. Gestione dei dati per il grafico e calcolo dei centri e raggi.

```
// Dati per il grafico
const data = [40, 30, 30]; // Percentuali
const colors = ["#FF5733", "#33FF57", "#3357FF"]; // Colori per gli spicchi

// Centri e raggio del grafico
const centerX = canvas.width / 2;
const centerY = canvas.height / 2;
const radius = 150;
```

Rappresentazione di un grafico a torta

4. Gestione delle singole sezioni del grafico.

```
data.forEach((percentage, index) => {
    // Calcola l'angolo finale
    const sliceAngle = (percentage / 100) * Math.PI * 2;

    // Disegna l'arco (spicchio)
    ctx.beginPath();
    ctx.moveTo(centerX, centerY); // Centro del cerchio
    // Disegna l'arco
    ctx.arc(centerX, centerY, radius, startAngle, startAngle + sliceAngle);
    ctx.closePath();
```

Rappresentazione di un grafico a torta

5. Gestione delle singole sezioni del grafico.

```
    // Riempie lo spicchio con il colore corrispondente
    ctx.fillStyle = colors[index];
    ctx.fill();

    // Avanza l'angolo di partenza
    startAngle += sliceAngle;
});
```

Rappresentazione di un grafico a torta

5. Gestione delle singole sezioni del grafico.

```
    // Riempie lo spicchio con il colore corrispondente
    ctx.fillStyle = colors[index];
    ctx.fill();

    // Avanza l'angolo di partenza
    startAngle += sliceAngle;
});
```

Rappresentazione di un grafico a torta

Risultato Finale