# Cheat sheet p5.js

Corso di Computer graphics a.s. 2023/24
Prof. Ceccotti

## 1 Per iniziare

- `setup(){*contenuti*};`

  Funzione richiamata una volta all'avvio del programma, usata per inizializzare variabili, impostare la dimensione del canvas, ecc.

- `createCanvas(X,Y)`

  Crea la nostra "tela" virtuale rettangolare, con lato orizzontale di dimensioni "X" e lato verticale di dimensioni "Y". Viene solitamente messo all'interno delle parentesi graffe della funzione setup()

- `draw(){*contenuti*};`

  Funzione richiamata in continuazione dopo setup(), usata per disegnare gli elementi sul canvas.

## 2 Variabili

Una variabile è un'entità di programmazione utilizzata per **memorizzare** e **manipolare dati** all'interno di un programma. Essa è caratterizzata da un **nome univoco** e può contenere valori di diversi tipi.

### 2.1 Dichiarazione

. Tramite il comando

```
let x;
```

dichiariamo l'esistenza nel programma di una variabile **x**

### 2.2 Assegnazione

. Scrivendo il nome di una variabile già dichiarata seguita dal simbolo "=" e successivamente un valore è possibile assegnare un valore numerico a quella variabile che verrà salvato in memoria.

```
x=10.3;
```

è possibile anche assegnare un valore durante la dichiarazione:

```
let x=2;
```

### 2.3 Tipi di dati

- **Interi**: Numeri senza parte decimale.

  ```
  let numeroInteri = 42;
  ```

- **Decimali**: Numeri con una parte decimale.

  ```
  let numeroDecimali = 3.14;
  ```

- **Stringhe**: Sequenze di caratteri alfanumerici.

  ```
  let testo = "Una stringa";
  ```

- **Booleani**: Valori che possono essere vero o falso, usati per vari scopi.

  ```
  let vero = true;
  let falso = false;
  ```

- **Colori**: Possono essere definiti nello spazio dei colori dello spazio RGB (red, green, blue) con una terna di valori compresi tra 0 e 255.

  ```
  let verde = color(0, 255, 0);
  let rosso = color(255, 0, 0);
  let blu = color(0, 0, 255);
  ```

  In alternativa possono essere dichiarati con un valore esadecimale:

  ```
  let giallo = color('#FFFF00');
  let viola = color('#8f00ff');
  ```

  Se invece vogliamo lavorare con colori dal bianco al nero (in scale di grigi) è sufficiente usare un singolo numero compreso tra 0 (nero) e 255 (bianco).

  ```
  let nero = color(0);
  let grigio=color(122);
  let bianco = color(255);
  ```

### 2.4 Variabili speciali di sistema

Oltre alle variabili che possiamo dichiarare abbiamo un serie di nomi che sono già assegnati di default, che possono essere molto utili in certi contesti:

- ```
  width;
  height;
  ```

  Restituiscono i valori X e Y della tela.

- ```
  frameCount();
  millis();
  ```

  Variabili utili per evoluzione temporale delle animazioni, frameCount() restituisce il numero del fotogramma attuale (solitamente ci sono 30 fotogrammi al secondo), millis() il valore di tempo in millisecondi.

- ```
  mouseX;
  mouseY;
  pmouseX;
  pmouseY;
  ```

  mouseX e mouseY restituiscono le coordinate attuali del cursore, mentre pmouseX e pmouseY le coordinate del mouse nel fotogramma precedente.
  ```

# 3 Operatori

## 3.1 Operatori aritmetici

### 3.1.1 Operazioni fondamentali

(+, -, *, /) sono i simboli che rappresentano le quattro operazioni fondamentali: **somma, sottrazione, moltiplicazione, divisione**.

```
let risultato = x + y;
let k = x - 3;
let raggio = distanza * 5;
let vel = spazio / frameCount();
```

### 3.1.2 Modulo

(%) è l'operazione **modulo**. Questa restituisce il resto della divisione tra due numeri. In pratica, viene calcolato il resto quando si divide il primo numero per il secondo. Ad esempio, 17 modulo 4 produce 1 come risultato, poiché 17 diviso per 4 è 4 con un resto di 1.

In JavaScript è indicata:

```
let resto = 17 % 4;
```

Questa operazione è utile per verificare se un numero è pari o dispari (il resto di una divisione per 2 sarà 0 per i numeri pari e 1 per i numeri dispari) o per fare un ciclo attraverso un insieme finito di valori.

## 3.2 Operatori logici

Gli operatori logici sono utilizzati per eseguire operazioni logiche su valori booleani (vero o falso).

### 3.2.1 AND (&&)

L'operatore logico AND (&&) restituisce vero se entrambe le espressioni valutate sono vere; altrimenti, restituisce falso. Ad esempio:

```
let a = true;
let b = false;
let risultatoAND = a && b;
```

### 3.2.2 OR (||)

L'operatore logico OR (||) restituisce vero se almeno una delle due espressioni valutate è vera; altrimenti, restituisce falso. Ad esempio:

```
let c = true;
let d = false;
let risultatoOR = c || d;
```

### 3.2.3 NOT (!)

L'operatore logico NOT (!) inverte il valore di verità dell'espressione. Se l'espressione è vera, NOT la rende falsa, e viceversa. Ad esempio:

```
let e = true;
let risultatoNOT = !e;
```

## 3.3 Operatori Relazionali

(>, <, >=, <=, ==, !=) questi sono in ordine da sinistra: maggiore, minore, maggiore e uguale, minore e uguale, uguale e diverso. Sono usati verificare condizioni, producono un output **booleano**.

# 4 Strutture di Controllo

## 4.1 Condizionali (if, else if, else)

Le strutture di controllo condizionali consentono di eseguire istruzioni diverse in base al verificarsi di determinate condizioni.

- **if**: L'istruzione `if` viene utilizzata per eseguire un blocco di codice se una condizione specificata è vera. Ad esempio:

```
let x = 10;
if (x > 5) {
    print("x maggiore di 5");
}
```

- **else if**: L'istruzione `else if` viene utilizzata per eseguire un blocco di codice se la prima condizione specificata è falsa e la condizione successiva è vera. Può essere utilizzata dopo un'istruzione `if` o un'altra istruzione `else if`. Ad esempio:

```
let y = 3;
if (y > 5) {
    print("y maggiore di 5");
} else if (y < 5) {
     print("y minore di 5");
}
```

- **else**: L'istruzione `else` viene utilizzata per eseguire un blocco di codice se tutte le condizioni precedenti sono false. Ad esempio:

```
let z = 5;
if (z > 5) {
     print("z maggiore di 5");
} else if (z < 5) {
     print("z minore di 5");
} else {
     print("z uguale a 5");
}
```

## 4.2 Cicli (for, while, do-while)

I cicli sono utilizzati per eseguire ripetutamente un blocco di codice fino a quando una condizione specificata non è più vera.

- **for**: Il ciclo `for` viene utilizzato per eseguire un blocco di codice un numero specificato di volte. Ad esempio:

```
for (let i = 0; i < 5; i++) {
     print(i);
}
```

- **while**: Il ciclo `while` viene utilizzato per eseguire un blocco di codice fintanto che una condizione specificata è vera. Ad esempio:

```
let n = 0;
while (n < 5) {
     print(n);
    n++;
}
```

# 5 Elementi base del disegno

- **point(x, y)**: Disegna un punto sul canvas alle coordinate specificate.
  - `x`: La coordinata x del punto.
  - `y`: La coordinata y del punto.
- **line(x1, y1, x2, y2)**: Disegna una linea tra due punti sul canvas.
  - `x1`: La coordinata x del primo punto.
  - `y1`: La coordinata y del primo punto.
  - `x2`: La coordinata x del secondo punto.
  - `y2`: La coordinata y del secondo punto.
- **rect(x, y, larghezza, altezza)**: Disegna un rettangolo sul canvas.
  - `x`: La coordinata x dell'angolo in alto a sinistra del rettangolo.
  - `y`: La coordinata y dell'angolo in alto a sinistra del rettangolo.
  - `larghezza`: La larghezza del rettangolo.
  - `altezza`: L'altezza del rettangolo.
- **circle(x, y, diametro)**: Disegna un cerchio sul canvas.
  - `x`: La coordinata x del centro del cerchio.
  - `y`: La coordinata y del centro del cerchio.
  - `diametro`: Il diametro del cerchio.
- **ellipse(x, y, larghezza, altezza)**: Disegna un'ellisse sul canvas.
  - `x`: La coordinata x del centro dell'ellisse.
  - `y`: La coordinata y del centro dell'ellisse.
  - `larghezza`: La larghezza dell'ellisse.
  - `altezza`: L'altezza dell'ellisse.
- **arc(x, y, larghezza, altezza, inizio, fine)**: Disegna un arco sul canvas.
  - `x`: La coordinata x del centro dell'arco.
  - `y`: La coordinata y del centro dell'arco.
  - `larghezza`: La larghezza dell'arco.
  - `altezza`: L'altezza dell'arco.
  - `inizio`: L'angolo iniziale dell'arco in radianti.
  - `fine`: L'angolo finale dell'arco in radianti.
- **triangle(x1, y1, x2, y2, x3, y3)**: Disegna un triangolo sul canvas.
  - `x1`, `y1`: Le coordinate del primo vertice del triangolo.
  - `x2`, `y2`: Le coordinate del secondo vertice del triangolo.
  - `x3`, `y3`: Le coordinate del terzo vertice del triangolo.
- **quad(x1, y1, x2, y2, x3, y3, x4, y4)**: Disegna un quadrilatero sul canvas.
  - `x1`, `y1`: Le coordinate del primo vertice del quadrilatero.
  - `x2`, `y2`: Le coordinate del secondo vertice del quadrilatero.
  - `x3`, `y3`: Le coordinate del terzo vertice del quadrilatero.
  - `x4`, `y4`: Le coordinate del quarto vertice del quadrilatero.

# 6 Colore

- **fill(r, g, b)**: Imposta il colore di riempimento utilizzando i valori RGB.
  - `r`: La quantità di rosso nel colore (da 0 a 255).
  - `g`: La quantità di verde nel colore (da 0 a 255).
  - `b`: La quantità di blu nel colore (da 0 a 255).
- **noFill()**: Disabilita il riempimento. Gli oggetti disegnati successivamente non saranno riempiti di colore.
- **stroke(r, g, b)**: Imposta il colore del tratto utilizzando i valori RGB (vedi sopra).
- **noStroke()**: Disabilita il tratto. Gli oggetti disegnati successivamente non avranno un bordo.
- **background(r, g, b)**: Imposta il colore di sfondo del canvas utilizzando i valori RGB (vedi sopra).

# 7 Testo

- **text(testo, x, y)**: Disegna il testo specificato sul canvas alle coordinate `x` e `y`.
  - `testo`: Il testo da disegnare.
  - `x`: La coordinata x del punto di origine del testo.
  - `y`: La coordinata y del punto di origine del testo.
- **textSize(dimensione)**: Imposta la dimensione del testo utilizzando il parametro `dimensione`.
  - `dimensione`: La dimensione del testo da impostare.
- **textAlign(allineamento)**: Imposta l'allineamento del testo sul canvas.
  - `allineamento`: L'allineamento del testo, che può essere impostato su "LEFT", "CENTER" o "RIGHT".

# 8 Immagini

- **loadImage(percorso)**: Carica un'immagine dal percorso specificato.
  - `percorso`: Il percorso dell'immagine da caricare.
- **image(img, x, y)**: Mostra un'immagine sul canvas alle coordinate specificate.
  - `img`: L'oggetto immagine da mostrare.
  - `x`: La coordinata x dell'angolo in alto a sinistra dell'immagine.
  - `y`: La coordinata y dell'angolo in alto a sinistra dell'immagine.

# 9 Roba a caso

- **random(i, f)**: Genera un numero casuale tra due numeri "i" e "f".
- **noise(x)**: Restituisce un valore di rumore casuale basato sulla variazione continua nel tempo e nello spazio. Se viene specificato un argomento opzionale 'x', il valore di rumore sarà coerente nello spazio e nel tempo al variare del valore di 'x'.

# 10 Interazione con mouse e tastiera

- **mousePressed()**: Questa funzione contiene i comandi che verranno eseguiti quando il mouse viene premuto.

```
mousePressed() {
    // Inserisci qui i comandi da eseguire quando il mouse viene premuto
}
```

- **mouseReleased()**: Questa funzione contiene i comandi che verranno eseguiti quando il mouse viene rilasciato.

```
mouseReleased() {
    // Inserisci qui i comandi da eseguire quando il mouse viene rilasciato
}
```

- **mouseClicked()**: Questa funzione contiene i comandi che verranno eseguiti quando viene fatto clic sul mouse.

```
mouseClicked() {
    // Inserisci qui i comandi da eseguire quando viene fatto clic sul mouse
}
```

- **keyPressed()**: Questa funzione viene chiamata quando una tastiera viene premuta. È possibile utilizzare la logica degli if per determinare quale tasto è stato premuto utilizzando la variabile keyCode. Ad esempio:

```
keyPressed() {
    if (keyCode == UP_ARROW) {
        // Inserisci qui i comandi da eseguire quando viene premuto il tasto freccia su
    } else if (keyCode == DOWN_ARROW) {
        // Inserisci qui i comandi da eseguire quando viene premuto il tasto freccia gi
    }
    // Aggiungi altre condizioni if necessarie per gestire altri tasti
}
```

# 11 Commenti

- **Singola riga**: I commenti su singola riga vengono utilizzati per inserire brevi spiegazioni o annotazioni nel codice. Possono essere aggiunti inserendo due barre "//" seguite dal testo del commento.
- **Multilinea**: I commenti multilinea vengono utilizzati per inserire spiegazioni più lunghe o blocchi di commento nel codice. Inizia con "/*" e termina con "*/". Possono coprire più righe di codice.