Processing Cheat-Sheet

Grundstruktur: Interaktiver Sketch

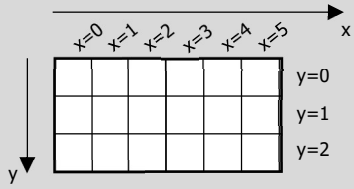
```
void setup() {
  size(400, 600);
}

void draw() {
  // …
}
```

Der `setup`-Block wird einmal zu Beginn ausgeführt. Dann wird wiederholt der `draw`-Block ausgeführt. Die `size`-Anweisung legt die Größe des Sketches fest.

Koordinatensystem in Processing

Der Ursprung befindet sich oben links. Alle Angaben sind in Pixeln.



Hintergrundfarbe

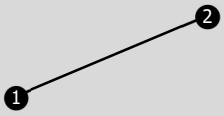
```
background(graustufe);
```
oder
```
background(r, g, b);
```

Übermalt den gesamten Zeichenbereich mit einer Farbe. *Siehe auch: Farbsystem.*

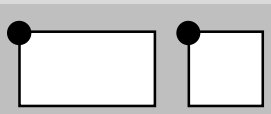
Typische Verwendung am Anfang des `draw`-Blocks:

```
void draw() {
  background(255);
  // …
}
```

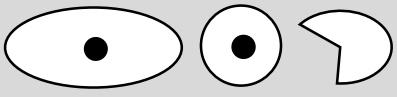
Einfache Zeichenanweisungen




```
line(x1, y1, x2, y2);
```



```
rect(x, y, breite, hoehe);
square(x, y, seitenlaenge);
```
Das Verhalten dieser Anweisungen kann auch verändert werden. *Siehe:* `rectMode`



```
ellipse(x, y, breite, hoehe);
circle(x, y, durchmesser);
arc( Parameter siehe Processing Referenz online );
```
Das Verhalten dieser Anweisungen kann auch verändert werden. *Siehe:* `ellipseMode`


```
triangle(x1, y1, x2, y2, x3, y3);
quad(x1, y1, x2, y2, x3, y3, x4, y4);
```

```
point(x, y);
```

Zeicheneinstellungen

Diese Einstellungen gelten für alle weiteren Zeichenanweisungen, bis die Einstellungen erneut geändert werden.

Standardeinstellung: mit Füllung und Rahmen

`noFill();`
Füllung von Formen ausschalten

`noStroke();`
Zeichnen des Rahmens ausschalten

```
strokeWeight(dicke);
```
Stellt die Dicke des Rahmens ein.

```
strokeCap(ROUND);
strokeCap(SQUARE);
strokeCap(PROJECT);
```

Stellt den Abschluss von Rahmenlinien ein. (z.B. für die `line`- und `point`-Anweisung)

```
strokeJoin(MITER);
strokeJoin(BEVEL);
strokeJoin(ROUND);
```

Stellt die Verbindung von Rahmenlinien ein. (z.B. bei Eckpunkten der `rect`-Anweisung).

RGB-Farbsystem

Die Farbeinstellungen gelten für alle weiteren Zeichenanweisungen, bis die Einstellungen erneut geändert werden.

Graustufen werden als Zahl von 0 (schwarz) bis 255 (weiß) angegeben.

Farben werden im RGB-System (Rot-Wert, Grün-Wert und Blau-Wert) angegeben. Jeder Wert ist eine Zahl zwischen 0 und 255.

Der **Alpha-Wert** legt die Deckkraft fest und kann verwendet werden, wenn mehrere Formen übereinander gezeichnet werden. Der Wert ist eine Zahl zwischen

0 (völlig transparent) und 255 (komplett deckend)

Farbeinstellungen

```
stroke(grauwert);
stroke(grauwert, alpha);
stroke(r, g, b);
stroke(r, g, b, alpha);
```

Legt die **Rahmenfarbe** von Formen fest.

```
fill(grauwert);
fill(grauwert, alpha);
fill(r, g, b);
fill(r, g, b, alpha);
```

Legt die **Füllfarbe** von Formen fest.

Processing Online Referenz

Online:
processing.org/reference

Kommentare und Einrückungen

Kommentare sind Hinweise und Erklärungen im Programmcode, die von Processing selbst ignoriert werden.

```
// einzeiliger Kommentar
/*
   mehrzeiliger
   Kommentar
*/
```

In **Blöcken** werden mehrere Anweisungen mit **geschweiften Klammern** { und } zusammengefasst.

Regel 1: Die öffnende Klammer steht am Ende der Zeile, die den Block einleitet.

Regel 2: Alle Anweisungen in einem Block werden mit zwei Leerzeichen eingerückt.

Regel 3: Die schließende Klammer steht allein in einer Zeile.

Tipp: Der Editor von Processing kann die Einrückungen automatisch korrigieren:

Menü: Bearbeiten → Autoformatierung
Tastenkombination: STRG + T

Rechenoperatoren

An allen Stellen, wo Zahlenwerte angegeben werden, können stattdessen auch Rechnungen verwendet werden.

Wie in Mathe gilt:
Klammer vor Punkt vor Strich

Beispiel:
```
circle(mouseX + 10, mouseY - 10, 5);
```

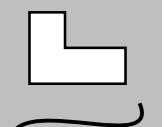
`a + b`	Addition
`a - b`	Subtraktion
`a * b`	Multiplikation
`a / b`	Division
`a % b`	Modulo (Rest bei Division)
`pow(a, b)`	Potenz a^b

```
random(a, b)
```
Zufallszahl zwischen a und b.

Erweiterte Zeichenanweisungen

siehe Online-Referenz:

```
beginShape, vertex
endShape
```



```
bezier, bezierVertex
```

```
push();   pop();
```
Soll die Zeichen- und Farbeinstellung nur für wenige Anweisungen geändert werden, kann man mit `push` die bisherigen Einstellungen speichern. Nach dem Ändern und Zeichnen können die gespeicherten Einstellungen mit `pop` wiederhergestellt werden.

```
frameRate(zahl);
```
Legt fest, wie oft der `draw`-Block pro Sekunde gestartet wird.

Verzweigungen

```
if ( <Bedingung> ) {
  <Dann-Block>
}
```
„Wenn…dann…"-Verzweigung

```
if ( <Bedingung> ) {
  <Dann-Block>
}
else {
  <Sonst-Block>
}
```
„Wenn…dann…sonst…"-Verzweigung

```
if ( <Bedingung A> ) {
  <Dann-Block A>
}
else if ( <Bedingung B> ) {
  <Dann-Block B>
}
else {
  <Sonst-Block>
}
```
Mehrfach-Verzweigung: Der Block mit der ersten erfüllten Bedingung wird ausgeführt.

Bedingungen

`x > y`	`x >= y`	größer / größergleich
`x < y`	`x <= y`	kleiner / kleinergleich
`x == y`		gleich
`x != y`		ungleich

Vergleichsoperatoren für Bedingungen

`A && B`	A und B
`A || B`	A oder B
`!A`	nicht A

Logische Operatoren für zusammengesetzte Bedingungen

Variablen

`int`	Ganzzahl, z.B. `123`, `-56`
`long`	Ganzzahl mit größerem Wertebereich
`float`	Gleitkommazahl z.B `3.1415`
`double`	genauere Gleitkommazahl
`boolean`	Wahrheitswert (`true`/`false`)
`char`	Zeichen z.B. `'t'`
`color`	Farben, z.B. `color(255, 0, 0)`

Elementare **Datentypen**

```
<Datentyp> <Bezeichner> = <Wert>;
```
Deklaration einer Variablen

Beispiel:
```
int position = 30;
boolean wachsend = true;
```

```
<Bezeichner> = <Wert>;
```
Zuweisung
Der Wert der Variablen wird überschrieben. Der Wert kann auch eine Rechnung oder die Variable selbst beinhalten.

Kurzschreibweisen für Veränderungen:
```
x += 5; x -= 5; x *= 5; x /= 5;
x++; x--;
```

Beachte: Variablen sind nur innerhalb des Blocks, in dem sie deklariert werden, sichtbar!

Arrays

```
int[] <Bezeichner> = new int[n];
```
Array deklarieren
Erzeugt ein `int`-Array der Länge n mit den Indizes 0. 1. …. n-1.

```
<Bezeichner>[<i>]
```
Zugriff auf einen Wert im Array

```
<Bezeichner>.length
```
Länge des Arrays auslesen

Variablen von Processing

`mouseX` (`int`)	aktuelle Mausposition
`mouseY` (`int`)	
`pmouseX` (`int`)	Mausposition im vor-
`pmouseX` (`int`)	herigen Frame
`mousePressed` (`boolean`)	
`mouseButton`	Werte `LEFT`, `RIGHT` od. `CENTER`

Variablen zur **Maus**

`keyPressed` (`boolean`)	
`key` (`char`)	gedrückte Taste

Variablen zur **Tastatur**
Sondertasten: Siehe `keyCode`

`width`, `height`, `frameCount`, `frameRate`, `focused`
Weitere Variablen zum **Sketchfenster**

Datentyp String

```
String text = new String();
```
oder
```
String text = "";
```

Variablen mit dem Datentyp **String** können Zeichenketten (Texte) speichern. Strings in doppelten Anführungszeichen angegeben!

```
int wert = 15;
String s = "Betrag:" + wert + "€";
```
Mehrere Strings und Werte kann man **mit dem + Operator verbinden.**

Jede Stringvariable bringt eigene Unterprogramme („Methoden") mit:

`text.`**`length()`**
Anzahl der Zeichen im String

`text.`**`charAt(`**`position`**`)`**
Zeichen an den gegebenen Position

`text.`**`equals(`**`vergleichsText`**`)`**
Überprüft, ob zwei Strings gleich sind.
(bei Strings kann man nicht == verwenden!)

`text.`**`substring`**`(start, ende)`
Erstellt einen neuen Teil-String aus dem String von Position `start` bis `ende-1`.

```
int x = int("42");
float f = float("1.234");
```
`String`-Werte und -Variablen müssen für Berechnungen umgewandelt werden.

Schleifen

```
for ( <Start>; <Bedingung>; <Veränderung> ) {
  <Schleifen-Block>
}
```
for-Schleife (Zählschleife)

```
while ( <Bedingung> ) {
  <Schleifen-Block>
}
```
while-Schleife (Solange…-Schleife)

Steuerungsanweisungen im Schleifenblock:
`break;` Schleife komplett beenden
`continue;` Aktuellen Schleifenblock beenden

Unterprogramme *(Funktionen bzw. Methoden)*

```
void <Bezeichner Unterprogramm> (<Parameterliste>) {
  <Unterprogramm-Block>;
}
```
Deklaration eines Unterprogramms ohne Rückgabewert

```
<Rückgabetyp> <Bezeichner Unterprogramm> (<Parameterliste>) {
  <Unterprogramm-Block>;
  return <Rückgabewert>;
}
```
Deklaration eines Unterprogramms mit Rückgabewert

Mit der **`return`**-Anweisung wird ein Wert zurückgeben und das Unterprogramm beendet.

Unterprogramme *zur Nutzerinteraktion in Processing*

`void` **`mousePressed`**`() {…}`
Maustaste wird (herunter) gedrückt

`void` **`mouseReleased`**`() {…}`
Maustaste wird losgelassen

`void` **`mouseClicked`**`() {…}`
ein „Click" hat stattgefunden

`void` **`mouseMoved`**`() {…}`
Maus wird bewegt

`void` **`keyPressed`**`() {…}`
Taste wird (herunter) gedrückt

`void` **`keyReleased`**`() {…}`
Taste wird losgelassen

`void` **`keyTyped`**`() {…}`
Zeichen wird getippt

`void` **`mouseDragged`**`() {…}`
Maus wird mit gedrückter Taste bewegt

Textausgabe

`println``(text);`
`print``(text);`
Gibt einen Text oder Wert auf der Konsole aus. (`-ln`: mit Zeilenumbruch)

`printArray``(array);`
Gibt die Werte eines Arrays übersichtlich auf der Konsole aus.

`text(``text, x, y`**`);`**
Zeichnet den Text auf die Zeichenfläche

`textSize``(groesse);`
`textAlign``(ausrichtichtung);`
Legt die Schriftgröße und Ausrichtung (`LEFT`, `CENTER`, `RIGHT`) fest.

Mit dem Datentyp `PFont` und den Anweisungen `loadFont`, `textFont` kann die Schriftart geändert werden.

Dateien Ein-/Ausgabe

`saveStrings``(datei, stringArray);`
Speichert ein Array aus Strings (zeilenweise) in eine Datei im Sketch-Verzeichnis.

`String[] zeilen =` **`loadStrings`**`(datei);`
Lädt den Inhalt einer Datei zeilenweise in ein Array von Strings ein.