Einführung in p5.js

p5.js ist eine JavaScript-Bibliothek für *Creative Coding*. Mit vergleichsweise wenig Code lassen sich direkt im Browser Grafiken, Animationen, generative Systeme und interaktive Anwendungen entwickeln.

Im Kontext von *Creative Technologies* verwenden wir p5.js vor allem als **gestalterisches Werkzeug**. Code wird dabei ähnlich wie Form, Farbe, Typografie, Bewegung oder Klang als Gestaltungsmaterial eingesetzt.

Mit p5.js können beispielsweise:

- Formen und grafische Kompositionen erzeugt,
- Farben und visuelle Systeme untersucht,
- Muster und generative Grafiken entwickelt,
- Bewegung und zeitliche Abläufe gestaltet,
- Maus und Tastatur als Eingabe verwendet,
- interaktive Systeme prototypisch umgesetzt,
- Bild, Video und Sound eingebunden werden.

Keine Angst vor Code.

Für die ersten Experimente sind keine Programmierkenntnisse notwendig. Ziel ist zunächst nicht, JavaScript vollständig zu verstehen oder Code auswendig schreiben zu können. Entscheidend ist, eine gestalterische Idee zu entwickeln und zu verstehen, welche Teile des Codes das sichtbare und interaktive Ergebnis beeinflussen.

Der p5.js Web Editor

Für die ersten Experimente verwenden wir den p5.js Web Editor.

Damit können p5.js-Projekte direkt im Browser geschrieben, verändert und ausgeführt werden. Eine zusätzliche Entwicklungsumgebung muss zunächst nicht installiert werden.

Nach dem Öffnen des Editors kann direkt ein neuer Sketch angelegt werden. Der eigentliche p5.js-Code befindet sich in der Datei:

```
sketch.js
```

Unser erster Sketch

Wir beginnen mit einem sehr einfachen Beispiel: einem Kreis auf einer Zeichenfläche.

Code

```
function setup() {
  createCanvas(400, 400);
}

function draw() {
  background(240);
  circle(200, 200, 100);
```

```
}
```

Sketch im p5.js Editor öffnen

Ergebnis

Was passiert hier?

Das Beispiel besteht zunächst aus zwei wichtigen Funktionen:

```
function setup() {
}
```

`setup()` wird einmal beim Start des Sketches ausgeführt.

```
function draw() {
}
```

`draw()` wird anschließend kontinuierlich wiederholt.

Mit

```
createCanvas(400, 400);
```

wird eine Zeichenfläche mit einer Größe von 400 × 400 Pixeln erzeugt.

Der Befehl

```
circle(200, 200, 100);
```

zeichnet einen Kreis.

Das Koordinatensystem

p5.js verwendet ein zweidimensionales Koordinatensystem. Der Ursprung `0,0` befindet sich oben links.

- x bestimmt die Position von links nach rechts.
- y bestimmt die Position von oben nach unten.

Bei

```
circle(200, 200, 100);
```

liegt der Mittelpunkt des Kreises bei:

- `x = 200`
- `y = 200`

Der Durchmesser beträgt `100` Pixel.

Ändert die Werte und beobachtet, was passiert:

```
circle(100, 300, 50);
```

Grundprinzip des Creative Coding

Code verändern → Ergebnis beobachten → vergleichen → weiter verändern

Form gestalten

p5.js stellt verschiedene grundlegende geometrische Formen zur Verfügung.

```
circle(200, 200, 100);

rect(50, 50, 100, 100);

line(50, 350, 350, 50);
```

Die Formen können miteinander kombiniert werden.

Weitere grundlegende Zeichenfunktionen findet ihr in der p5.js Reference.

Farbe gestalten

Mit `background()`, `fill()` und `stroke()` können Farben verändert werden.

```
function setup() {
  createCanvas(400, 400);
}

function draw() {
  background(240);

  fill(255, 100, 80);
  stroke(0);

  circle(200, 200, 100);
}
```

`background()` definiert die Hintergrundfarbe.

`fill()` bestimmt die Füllfarbe.

`stroke()` bestimmt die Konturfarbe.

Eine Kontur kann mit

```
noStroke();
```

ausgeschaltet werden.

Parameter verändern

Viele Eigenschaften eines Sketches werden durch Zahlen definiert.

Bei

```
circle(200, 200, 100);
```

sind beispielsweise Position und Größe Parameter.

Genau hier beginnt die gestalterische Arbeit mit Code.

Experiment

Verändert jeweils nur einen Parameter und beobachtet die Wirkung.

- Position
- Größe
- Farbe
- Anzahl
- Abstand
- Verhältnis der Formen zueinander

Erstellt mehrere Varianten statt direkt nach einer vermeintlich „richtigen“ Lösung zu suchen.

Die Maus als Input

Bisher haben wir feste Zahlen verwendet.

p5.js kann Werte aber auch aus einer Interaktion beziehen.

Dafür stehen beispielsweise

```
mouseX
mouseY
```

zur Verfügung.

Wir ersetzen die feste Position des Kreises:

```
circle(200, 200, 100);
```

durch:

```
circle(mouseX, mouseY, 100);
```

Der Kreis folgt jetzt der Maus.

```
function setup() {
  createCanvas(400, 400);
}
```

```
function draw() {
  background(240);

  circle(mouseX, mouseY, 100);
}
```

Damit haben wir aus einer statischen Grafik bereits ein sehr einfaches interaktives System entwickelt.

Input → Process → Output

Interaktive Systeme können zunächst sehr einfach gedacht werden:

INPUT → PROCESS → OUTPUT

Zum Beispiel:

Mausposition → Berechnung im Code → Position eines Kreises

oder:

Mausklick → Bedingung im Code → Farbe verändert sich

Dieses Prinzip begegnet uns in sehr unterschiedlichen interaktiven Systemen immer wieder.

Bedingungen

Mit einer Bedingung kann ein Sketch unterschiedlich auf eine Situation reagieren.

Ein einfaches Beispiel:

```
function setup() {
  createCanvas(400, 400);
}

function draw() {
  background(240);

  if (mouseIsPressed) {
    fill(255, 100, 80);
  } else {
    fill(50, 120, 255);
  }

  circle(200, 200, 120);
}
```

Solange die Maustaste gedrückt wird, verändert sich die Farbe des Kreises.

Die grundlegende Struktur lautet:

```
if (BEDINGUNG) {

  // etwas passiert

} else {

  // etwas anderes passiert

}
```

Bewegung und Zeit

Die Funktion `draw()` wird kontinuierlich wiederholt. Dadurch können Werte über die Zeit verändert werden.

```
let x = 0;

function setup() {
  createCanvas(400, 200);
}

function draw() {
  background(240);

  circle(x, 100, 40);

  x = x + 2;
}
```

Die Variable x verändert sich bei jedem Durchlauf.

Dadurch bewegt sich der Kreis.

Interessant sind dabei nicht nur technische, sondern vor allem gestalterische Fragen:

- Wie schnell bewegt sich etwas?
- In welche Richtung?
- Ist die Bewegung gleichmäßig?
- Wie wirkt eine schnelle oder langsame Bewegung?
- Wann beginnt oder endet eine Bewegung?
- Wie beeinflusst Bewegung die Aufmerksamkeit?

Wiederholung

Eine besondere Stärke von Code besteht darin, Regeln wiederholt ausführen zu können.

```
function setup() {
  createCanvas(400, 400);
}
```

```
function draw() {
  background(240);

  for (let x = 20; x < width; x += 40) {
    circle(x, 200, 20);
  }
}
```

Die Schleife wiederholt den Kreis automatisch.

Aus einer einfachen Regel kann dadurch ein visuelles System entstehen.

Generative Gestaltung

Interessant wird Creative Coding, wenn Regeln, Wiederholung und Variation miteinander verbunden werden.

Beispielsweise können Eigenschaften verändert werden wie:

- Position
- Größe
- Farbe
- Rotation
- Abstand
- Geschwindigkeit
- Anzahl
- Zufall
- Reaktion auf Eingaben

Dadurch wird nicht mehr jedes einzelne Element gestaltet.

Stattdessen gestalten wir die **Regeln, nach denen ein visuelles oder interaktives System entsteht**.

Arbeiten mit KI

KI-Tools dürfen verwendet werden.

Für die Arbeit mit p5.js dürfen KI-Tools eurer Wahl verwendet werden, um Code zu generieren, zu erklären, zu verändern oder Fehler zu finden.

Ziel ist nicht, möglichst viel Code selbst schreiben zu können. Ziel ist, **Code als Werkzeug für gestalterische Experimente einsetzen zu können**.

Ein möglicher Prompt könnte beispielsweise lauten:

```
Create a simple p5.js sketch.

Canvas size: 600 × 600 pixels.

Create 20 circles that react to the position of the mouse.

Use only black, white and one additional color.

Keep the code simple.
```

```
Explain which parameters I can change to modify the visual appearance.
```

Das Ergebnis der KI ist dabei nicht das Ende des Gestaltungsprozesses.

Idee → Prompt → Code → Experiment → Beobachtung → Veränderung → Variante

Probiert den erzeugten Code aus.

Verändert einzelne Werte.

Entfernt Teile.

Fügt neue Regeln hinzu.

Erzeugt Varianten.

Vergleicht die Ergebnisse.

KI als Coding Assistant

KI kann auch dabei helfen, bestehenden Code zu verstehen.

Beispiel:

```
Explain this p5.js code to me.

I am a design student and have never programmed before.

Explain what each section does in simple language.

Then show me three parameters that I can change without changing
the basic structure of the program.
```

Auch bei Fehlern kann KI unterstützen:

```
This p5.js sketch produces an error.

Explain the error in simple language.

Do not rewrite the complete sketch.

Show me which line I need to change and explain why.
```

Code verstehen statt auswendig lernen

Auch bei der Arbeit mit KI ist es hilfreich, einige grundlegende Strukturen wiederzuerkennen.

Element	**Bedeutung**
`setup()`	Wird beim Start einmal ausgeführt.

Element	Bedeutung
`draw()`	Wird kontinuierlich wiederholt.
`let x = 100;`	Eine Variable speichert einen Wert.
`circle(x, y, size);`	Eine Funktion führt eine bestimmte Aktion aus.
`mouseX / mouseY`	Die Mausposition liefert Werte für eine Interaktion.
`if (…)`	Eine Bedingung entscheidet, was passiert.
`for (…)`	Eine Schleife wiederholt etwas.
`random()`	Erzeugt zufällige Werte.

Vom Kreis zum interaktiven System

Wir haben mit einem einzelnen statischen Kreis begonnen:

```
circle(200, 200, 100);
```

Im Verlauf der Einführung haben wir gesehen, dass dieselbe Grundidee schrittweise erweitert werden kann:

Form → Farbe → Parameter → Input → Bedingungen → Bewegung → Wiederholung → Interaktion

Aus wenigen einfachen Regeln kann dadurch ein dynamisches und interaktives System entstehen.

Endergebnis: ein interaktiver Sketch

Das folgende Beispiel verbindet mehrere der zuvor eingeführten Prinzipien.

Bewegt zunächst die Maus über die Zeichenfläche und beobachtet, wie das System reagiert. Klickt anschließend an unterschiedliche Positionen.

Sketch im p5.js Editor öffnen

Jetzt seid ihr dran.

Öffnet den Sketch im p5.js Web Editor und entwickelt daraus eine eigene Variante.

Verändert nicht einfach nur einzelne Farben. Überlegt, welche Regeln das Verhalten und die visuelle Wirkung des Systems bestimmen.

Was könnt ihr verändern?

- Form
- Anzahl
- Position
- Farbe
- Größe
- Bewegung
- Geschwindigkeit
- Reaktion auf die Maus
- Reaktion auf einen Klick

- Regeln und Bedingungen

Das Ziel ist nicht, den Beispiel-Sketch nachzubauen, sondern aus seinen Prinzipien ein **eigenes visuelles und interaktives System** zu entwickeln.

Creative Coding als Designprozess

In *Creative Technologies* interessiert uns weniger die Frage:

„Kann ich programmieren?“

Interessanter ist die Frage:

„Kann ich mit Code gestalten?“

p5.js ermöglicht dafür einen experimentellen Arbeitsprozess:

Regel definieren → Ergebnis beobachten → Parameter verändern → Varianten erzeugen → vergleichen → weiterentwickeln

Code wird damit zu einem weiteren Gestaltungsmaterial innerhalb der *Creative Technologies*.

Weiterführende Links

- p5.js
- p5.js Web Editor
- p5.js Reference
- p5.js Examples

From:
https://wiki.ct-lab.info/ - **Creative Technologies Lab | dokuWiki**

Permanent link:
https://wiki.ct-lab.info/doku.php/extras:codikon:p5js:einfuehrung_in_p5js?rev=1789719082

Last update: **2026/09/18 08:11**