HTML5 MasterClass

HTML5-Rundumschlag

<video>

Element und API

```
<video src="video.m4v" controls
  width="640" height="360">
  <a href="video.m4v">Video-Download</a>
</video>
```

(Interface abhängig vom Browser)

Multimedia-API

- JS-API für <audio> und <video>
- Komplett identisch für beide Elemente, gut designed
- Steuerungsfunktionen, Statusüberwachung via Events, vieles mehr
- Ausprobieren: **unterlagen/video/index.html**

Statusabfragen (Auszug)

- `error`: Fehler-Status
- `networkState`: Netzwerk-Infos
- `readyState`: Bereitschaftsstatus
- `currentSrc`: URL der gewählten Quelldatei
- `duration`: Gesamtlaufzeit der Datei
- `currentTime`: Aktuelle Zeitmarke

Events (Auszüge)

- `play`, `pause`, `volumechange`
- `loadedmetadata` (Metadaten wurden geladen)
- `timeupdate` (Datei wird abgespielt)
- `canplay` (Element ist abspielbereit)
- `error` (Fehlerfall)

Zwischenfazit Video-Element

- <video> = neues HTML-Element
- Einfache Einbindung, breite Browserunterstützung
- Einfache, umfassende Steuerungs-API

Alles klar soweit?

User Media

Zugriff auf Kameras und Mikrophone

```
// So funktioniert Media Capture
navigator.getUserMedia({
  video: true, // Kamera-Zugriff anfordern
  audio: true  // Mikrophon-Zugriff anfordern
}, function(stream){
  // Stream verarbeiten
}, function(err){
  // Fehler verarbeiten
});
```

```
// Beispiel: Webcam-Feed in einem
// Video-Element anzeigen
navigator.getUserMedia({
  video: true,
  audio: false
}, function(stream){
  var video = document.querySelector('video');
  video.src = window.URL.createObjectURL(stream);
  video.play();
}, function(err){
  console.error(err);
});
```

Zwischenfazit User Media

1. Einfacher Zugriff auf Kameras und Mikrophone
2. Streams mit anderen HTML5-APIs kombinierbar

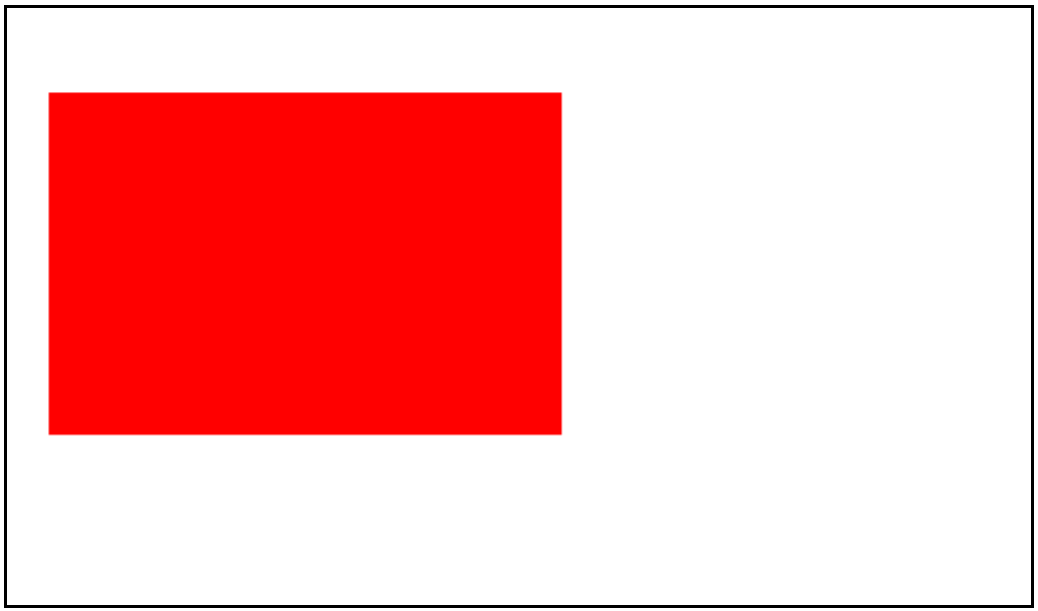
Alles klar soweit?

Canvas-Element

Die Bildbearbeitungs-API von HTML5

```
<!doctype html>
<meta charset="utf-8">
<title>Canvas-Beispiel</title>

<canvas id="Foo" height="500" width="600">
  Eine Ersatzbeschreibung des Dargestellten
</canvas>

<script>
  // Canvas-Code
</script>
```

```
var canvas = document.getElementById("Foo");
var context = canvas.getContext("2d");
context.fillStyle = "rgb(255, 0, 0)";
context.fillRect(20, 40, 240, 160);
```



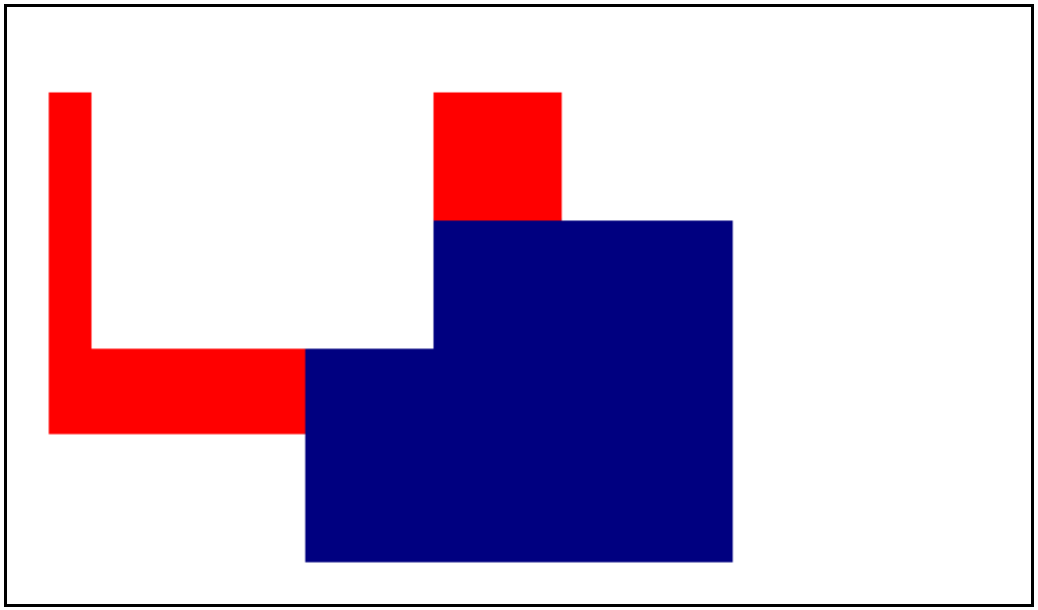
So funktioniert‘s

1. Canvas-DOM-Element referenzieren
2. Zeichen-API holen (`canvas.getContext()`; die in HTML5 spezifizierte Standard-API ist `"2dv`)
3. Umgebungswerte setzen (Farben, Transformationen)
4. Zeichen-Operationen durchführen

Wichtig!

- Kein eingebautes Objektsystem (im Gegensatz zu z.B. SVG)
- Immer erst Umgebungswerte setzen, dann zeichnen
- Was einmal gezeichnet wurde, kann außer durch übermalen nicht mehr verändert werden

```
// Etwas malen
context.fillStyle = "rgb(255, 0, 0)";
context.fillRect(20, 40, 240, 160);

// Einen Teilbereich übermalen
context.fillStyle = "rgb(0, 0, 128)";
context.fillRect(140, 100, 200, 160);

// Einen Teilbereich löschen
context.clearRect(40, 40, 160, 120);
```



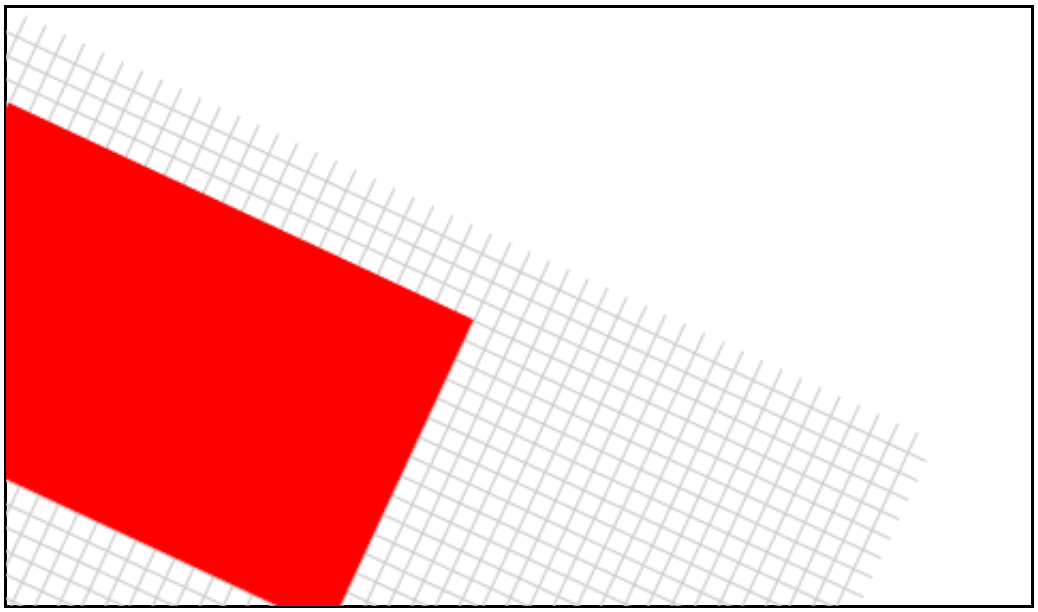
Einheiten

- Koordinaten und Maße: Gitternetzeinheiten (zwischen den Pixeln)
- Farben: `#FF0000`, `rgba(255, 0, 0, 1)`, `green` etc. (wie in CSS)
- Winkel: Radiant (1rad = 180 / π)

Grundprinzip klar? Dann jetzt **die wichtigsten Features des 2D-Kontexts!**

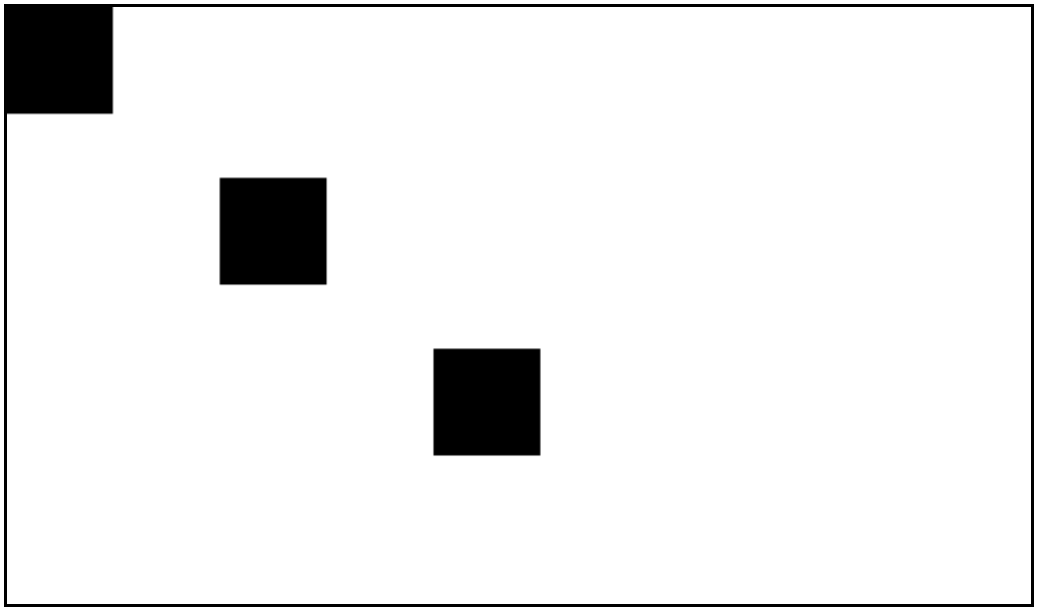
Transformationen

- Verschieben, rotieren, verzerren etc
- Rotiert nicht die (unveränderliche) Bitmap, sondern das Gitternetz (Zustand)
- Ergo: **Erst transformieren, dann zeichnen**

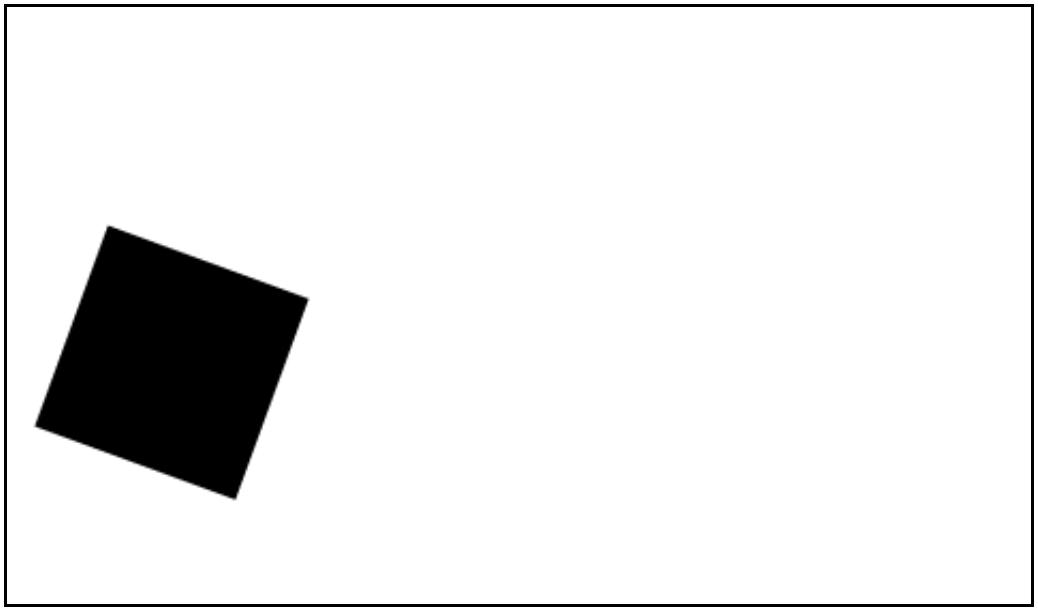


```
// Das Koordinatensystem ändert sich...
context.rotate(Math.toRad(25.0));

// ... aber die Koordinaten bleiben gleich!
context.fillRect(20, 40, 240, 160);
```

```
// Ein Quadrat zeichnen
context.fillRect(0, 0, 50, 50);

// Ursprungspunkt verschieben, Quadrat nochmal
context.translate(100, 80);
context.fillRect(0, 0, 50, 50);

// Und nochmal!
context.translate(100, 80);
context.fillRect(0, 0, 50, 50);
```

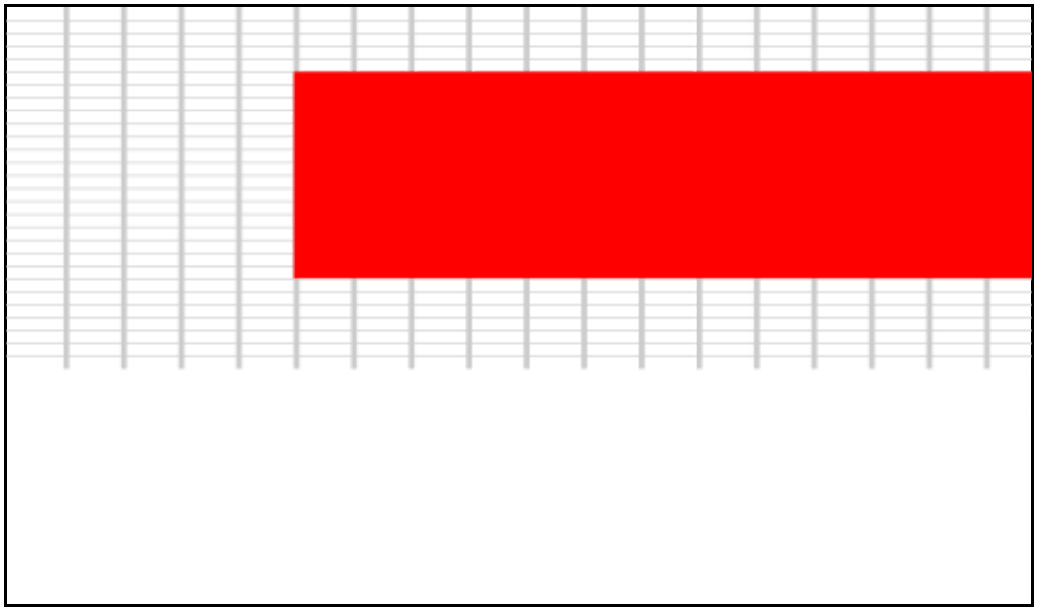


```
// Das Koordinatensystem rotieren
context.rotate(0.35); // Wtf?

// Ein Rechteck zeichnen
context.fillRect(80, 80, 100, 100);
```



```
// Hilfsfunktion zur Umrechnung
// von Grad in Radiant
Math.toRad = function(x){
  return (x * Math.PI) / 180;
};
```

```
// Das Koordinatensystem skalieren
context.scale(2.00, 0.50);

// Ein Rechteck zeichnen
context.fillRect(50, 50, 240, 160);
```

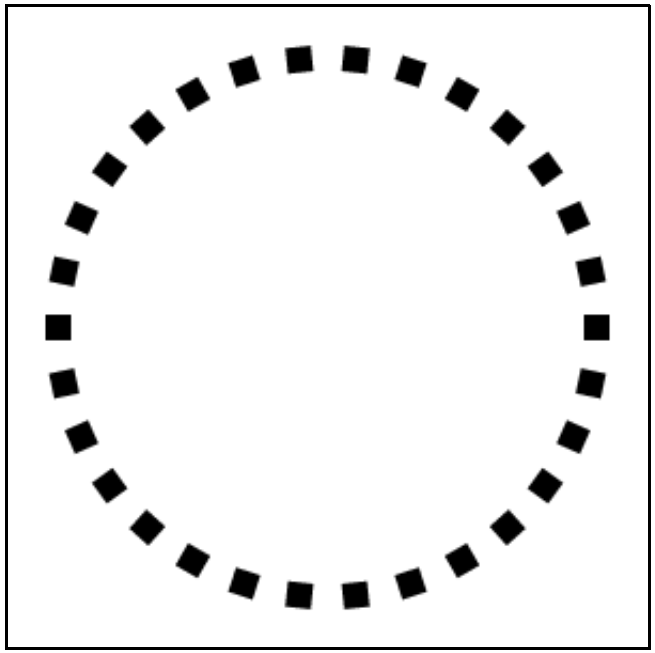


Genug graue Theorie: basteln Sie mit!

- **tinyurl.com/canvas-sandbox**
- Canvas-Sandbox mit jQuery
- `canvas` und `context` bereits definiert
- Winkel-Umrechner und andere Hilfsfunktionen vorhanden
- Hilfreiche Links

Basteln Sie mal!

- tinyurl.com/canvas-sandbox
- Textfeld = Canvas-Code
- Global: `canvas`, `context`, jQuery …
- Radiant-Rechner: `Math.toRad(deg)`
- Für Ehrgeizige: Regenbogenfarben



Pfade und Linien

- `beginPath()`: Neuen Pfad öffnen
- `moveTo(x, y)`: Nach (x/y) bewegen
- `lineTo(x, y)`: Linie nach (x/y) ziehen
- `stroke()`: Gezogene Linien zeichnen
- `closePath()`: Pfad (Form) schließen
- Pfade = Teile des Contexts

context.drawImage()

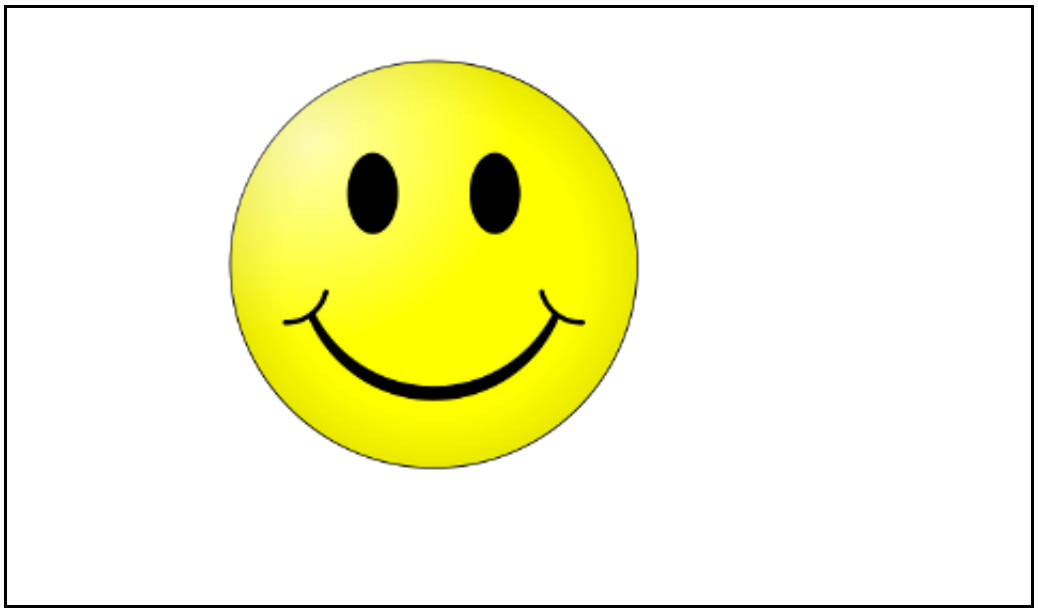
Drei Wege um Grafiken zu zeichnen:

```
drawImage(img, dx, dy)
```

```
drawImage(img, dx, dy, dw, dh)
```

```
drawImage(img, sx, sy, sw, sh, dx, dy, dw, dh)
```

```
// Bild-Objekt erstellen
var img = new Image();
img.src = "test.png";

// Wichtig: Erst zeichnen wenn das Bild da ist
img.onload = function(){
  context.drawImage(img, 100, 20);
};
```



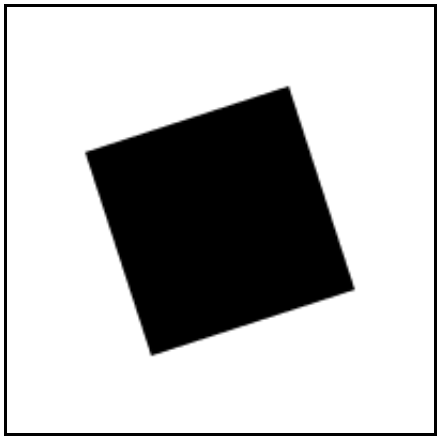
Canvas-Ausschnitte kopieren und einfügen

- Kopieren: `getImageData(x, y, w, h)`
- Ergebnis: `imageData`-Objekt mit den Bildinformationen des Ausschnitts
- Einfügen: `putImageData(imageData, x, y)`

Das imageData-Objekt

- Eigenschaften `width`, `height`
- Eigenschaft `data` enthält die Abfolge der Farbwerte (RGBA, 0 - 255) aller Pixel der Reihe nach (`Uint8ClampedArray`)

```
requestAnimationFrame(function render(){
  c1.clearRect(-50, -50, 100, 100);
  c1.rotate(Math.toRad(2));
  c1.fillRect(-50, -50, 100, 100);
  requestAnimationFrame(render);
});
```



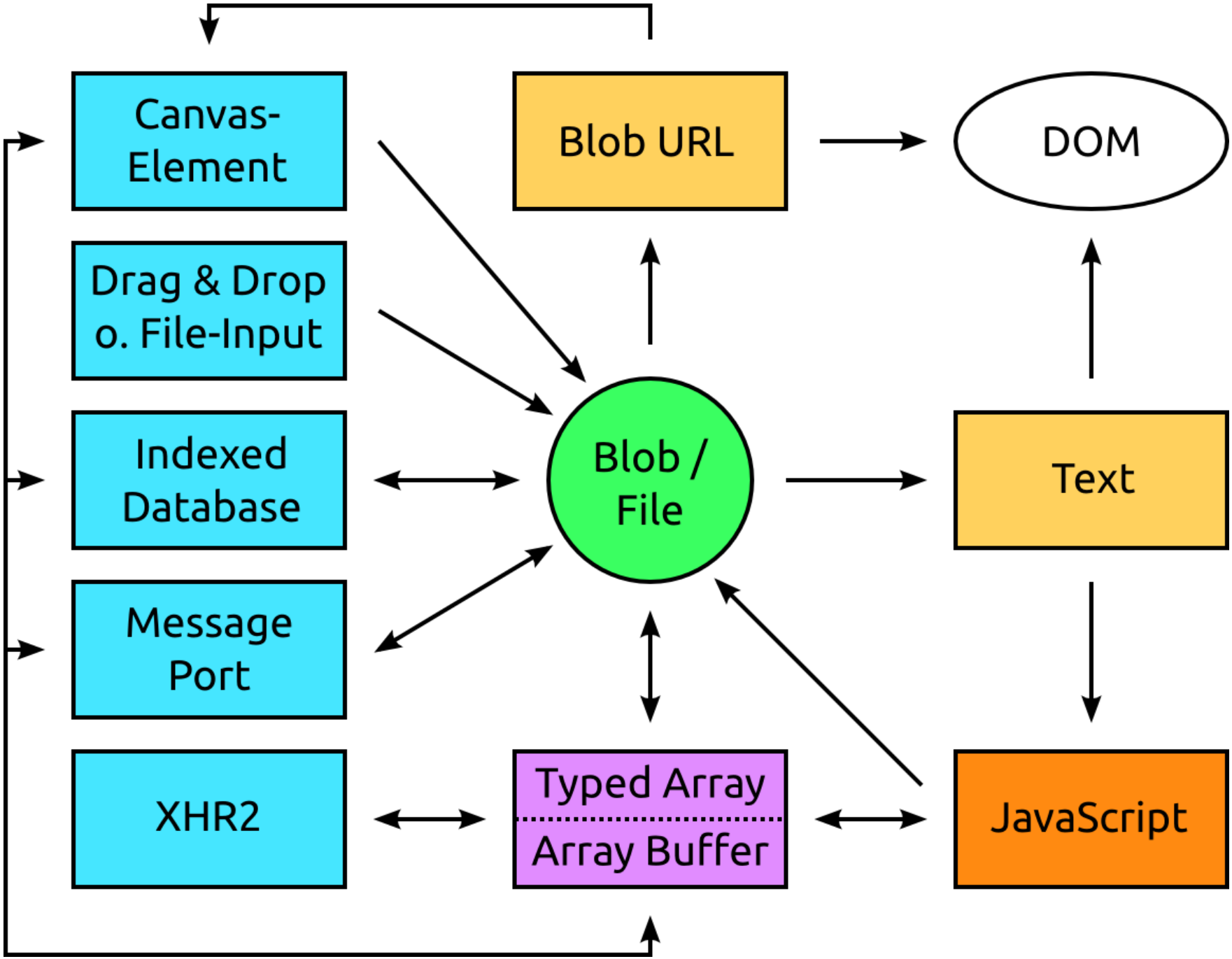
```
var source = document.querySelector('video');
var destination = document.querySelector('canvas#ziel');
var ctxDestination = destination.getContext('2d');



// Zwischenablagen-Canvas erstellen
var buffer = document.createElement('canvas');
buffer.height = ziel.height;
buffer.width = ziel.width;
var ctxBuffer = buffer.getContext('2d');



// Mit dem Video die Kopierschleife starten
source.addEventListener('play', function(){
  requestAnimationFrame(copyLoop);
}, false);



// Kopierschleife
function copyLoop(){
  ctxBuffer.drawImage(source, 0, 0, 640, 360);
  var imageData = ctxBuffer.getImageData(0, 0, 640, 360);
  applyFilter(imageData);
```

```
  ctxDestination.putImageData(imageData, 0, 0);
  if(!source.paused){
    requestAnimationFrame(copyLoop);
  }
}
```

```
// Wendet den Effekt an
function applyFilter(imageData){
  var pixels = imageData.data;
  var i = 0;
  var r, g, b, new_r, new_g, new_b;
  while(i < pixels.length){
    r = pixels[i], g = pixels[i + 1], b = pixels[i + 2];
    new_r = Math.min(255, r * 0.393 + g * 0.769 + b * 0.189),
    new_g = Math.min(255, r * 0.349 + g * 0.686 + b * 0.168),
    new_b = Math.min(255, r * 0.272 + g * 0.534 + b * 0.131);
    pixels[i] = new_r;
    pixels[i + 1] = new_g;
    pixels[i + 2] = new_b;
    i += 4;
  }
}
```

Merke: Die **Kombination** von verschiedenen Techniken und APIs macht HTML5 stark!

```
// Canvas-Inhalt als Binärdaten exportieren
canvas.toBlob(function(blob){
  // "blob" enthält die Bilddaten
});
```


Canvas-
Element
Drag & Drop
o. File-Input
Indexed
Database
Message
Port
XHR2
Blob URL
Blob /
File
Typed Array
Array Buffer
DOM
Text
JavaScript

```
// Gesamten Inhalt exportieren
canvas.toBlob(function(blob){

  // URL auf Blob erzeugen...
  var url = window.URL.createObjectUrl(blob);

  // ... und im DOM anzeigen
  $('img').attr('src', url);

});
```

Zwischenfazit Canvas

- Universelle Bildbearbeitungs-API
- 2D-Zeichenfunktionen
- Bildmanipulation
- Datenexport

Alles klar soweit?

Fazit

1. Neue Elemente: Mehr Übersicht und Barrierefreiheit im Markup
2. Video-Element: HTML5 statt Flash
3. User Media API: Kamera- und Mikrofonzugriff
4. Canvas 2D: Universelle Bildbarabeitungs-API