

Programmieren von 0 auf 100 in wenigen Stunden

Eine praktische Einführung in die Programmierung,
die wirklich Spaß macht!

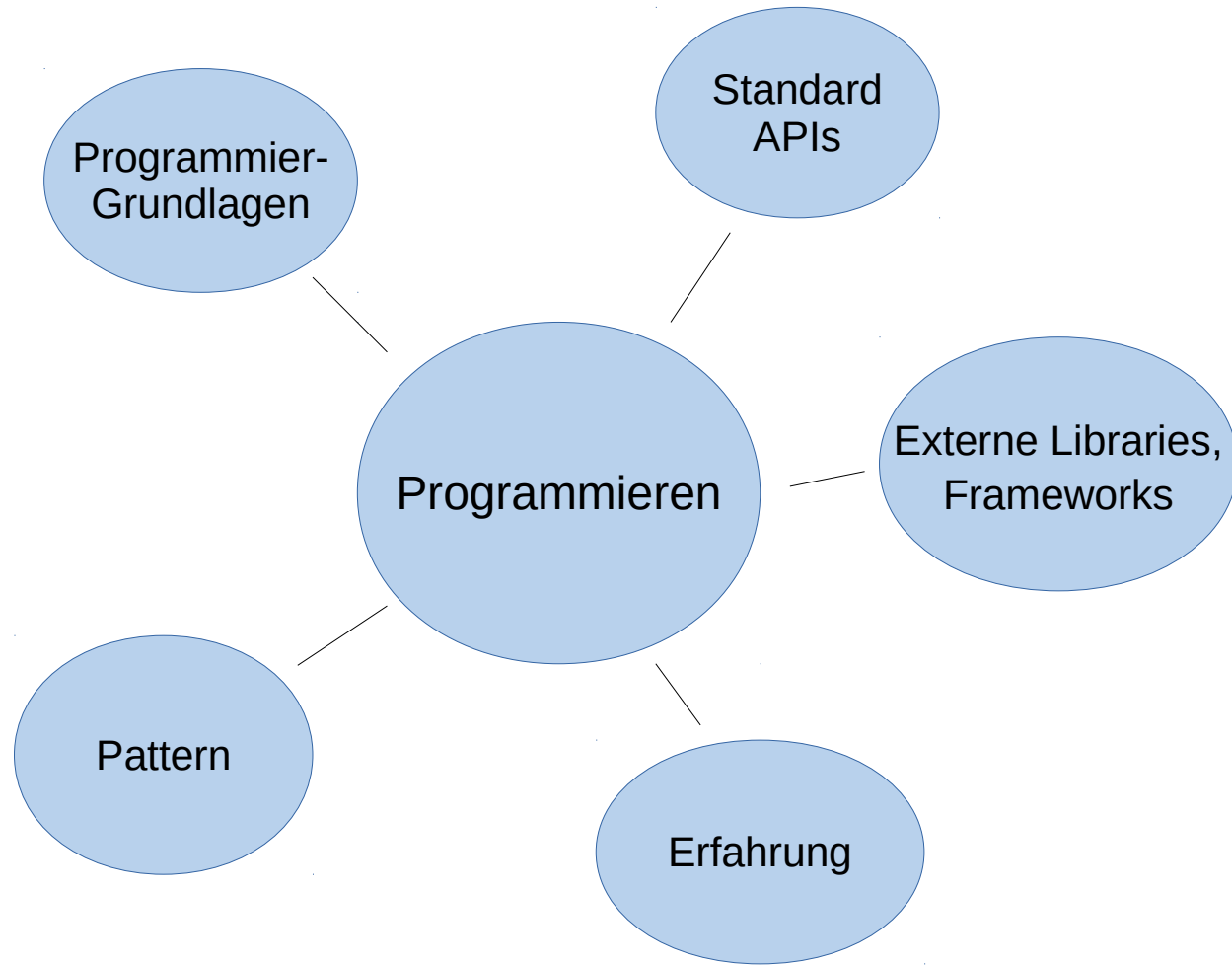
von Thomas Schütt

Zu meiner Person

Thomas Schütt, Dipl.-Ing.

#gerneperDu

- Studium der Elektrotechnik mit Schwerpunkt Informationsverarbeitung
- seit über 25 Jahren in der SW-Entwicklung tätig,
Bereiche digitale Kommunikation, Finanzdienstleistungen, Automotive
- Beruflicher Schwerpunkt: Java SE/Java EE Programmierung
(aber auch Perl, Python, Javascript, Typescript, C/C++, Assembler, ...)



JavaScript

- Läuft in jedem Browser
- Keine IDE nötig, keine „Magie“
- Sehr gute Dokumentation im Internet
- Sehr verbreitet, bei Stellenausschreibungen (Web-Design) sehr gefragt
- Objekt-orientiert, modern
- Einfach, fehlertolerant

Kursinhalt

1. Tag

- Vorstellung unserer Entwicklungsumgebung
- Schreiben und Verstehen einer minimalen HTML-Seite mit einer "Leinwand" ("canvas")
- API zum Malen auf der "Leinwand"
- Programmier-Basics: einfache Variableninhalte, Funktionen

Ergebnis: Wir können ein Raumschiff oder eine Figur auf dem Bildschirm darstellen und über Test-Buttons bewegen.

2. Tag

- API für den zeitlich verzögerten Programmablauf ("Game-Loop")
- API zur Tastaturabfrage
- Programmier-Basics: Listen bzw. Arrays, "if"-Blöcke, Programm-Schleifen

Ergebnis: Wir können ein Raumschiff oder eine Figur auf dem Bildschirm frei bewegen.

3. Tag

- animierte Bewegungen
- Programmier-Basics: strukturierte Variableninhalte
- Programmier-Basics: Objekt-orientierte Programmierung

Ergebnis: Wir können unser Raumschiff oder eine Figur z.B. schießen lassen, und behalten die Geschosse programmatisch im Blick.

4. Tag

Jetzt geht alles plötzlich ganz schnell - wir können auf die Früchte der vorigen Tage aufbauen, und im Handumdrehen neue Features implementieren.

- Erzeugen von feindlichen Raumschiffen, die sich zufällig bewegen, und zufällig Schüsse abgeben.
- Erkennen von Treffern (Kollisionserkennung), und eine einfache Darstellung der Explosionen.
- Programmier-Basics: Zufall

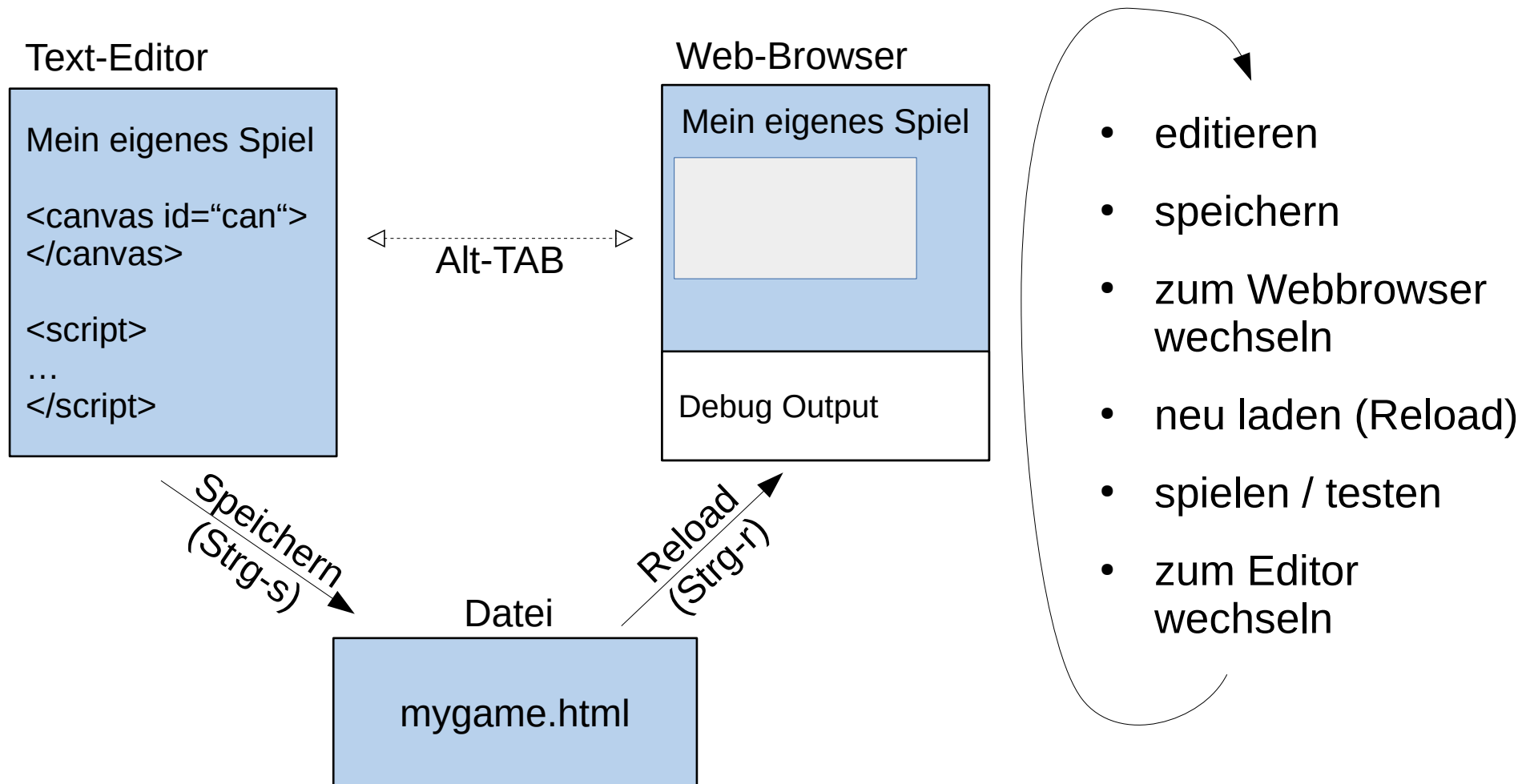
Ergebnis: Gegnerische Raumschiffe bewegen sich über den Bildschirm und feuern Schüsse ab.
"Treffer" werden visuell dargestellt und führen z.B. zur Vernichtung des Getroffenen.

5. Tag

- photorealistische Darstellungen auf der Leinwand
- API zur Sound-Ausgabe

Ergebnis: Tolle Optik und Sound ist nur der Zuckerguss über die harte Arbeit der Programmierung, machen aber gefühlt sehr viel aus.

Vorgehen



Ein paar Zeilen HTML

mygame.html

Ich bin eine HTML-Datei.

Ich bin **wirklich** eine HTML-Datei.

Ich bin **wirklich** eine HTML-Datei.

`<script>`

`alert("Hallo Du!");`

`</script>`

JavaScript !

An dieser Stelle ...

- Debugging: Öffnen der Entwickler-Werkzeuge
F12 -> Console-Tab
- HTML-Doku:
<https://www.w3schools.com/tags/>
<https://developer.mozilla.org/de/docs/Web/HTML/Element>
<https://wiki.selfhtml.org/wiki/HTML>

Ein paar mehr Zeilen HTML

```
<!DOCTYPE html>
<html>
<head>
  <meta http-equiv="content-type" content="text/html; charset=utf-8" />
</head>
<body>

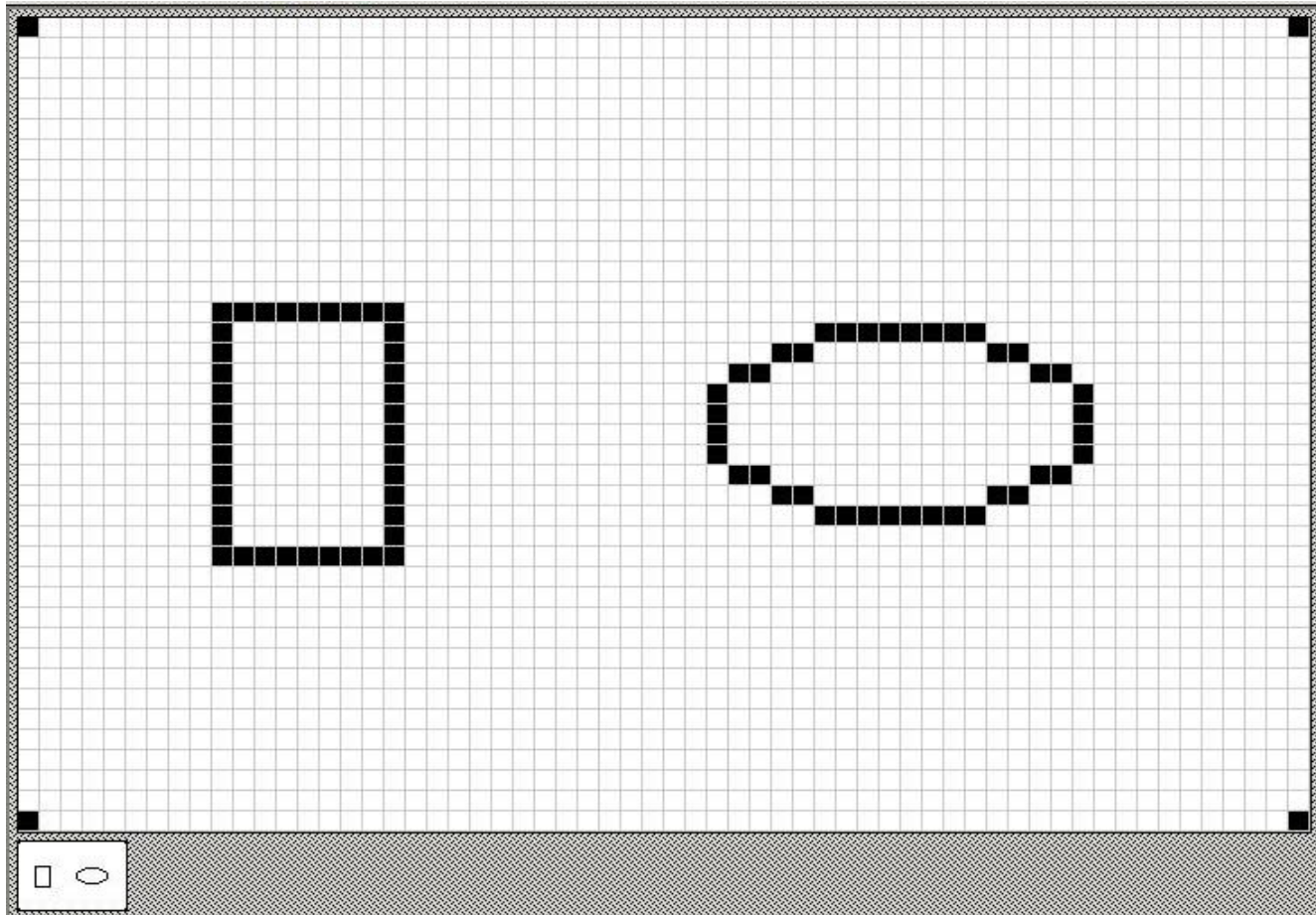
Mein eigenes Spiel
<p>

<canvas id="meinCanvas" width="400" height="300"
  style="border:1px solid #cccccc; background-color: #f1f1ff;">
</canvas>

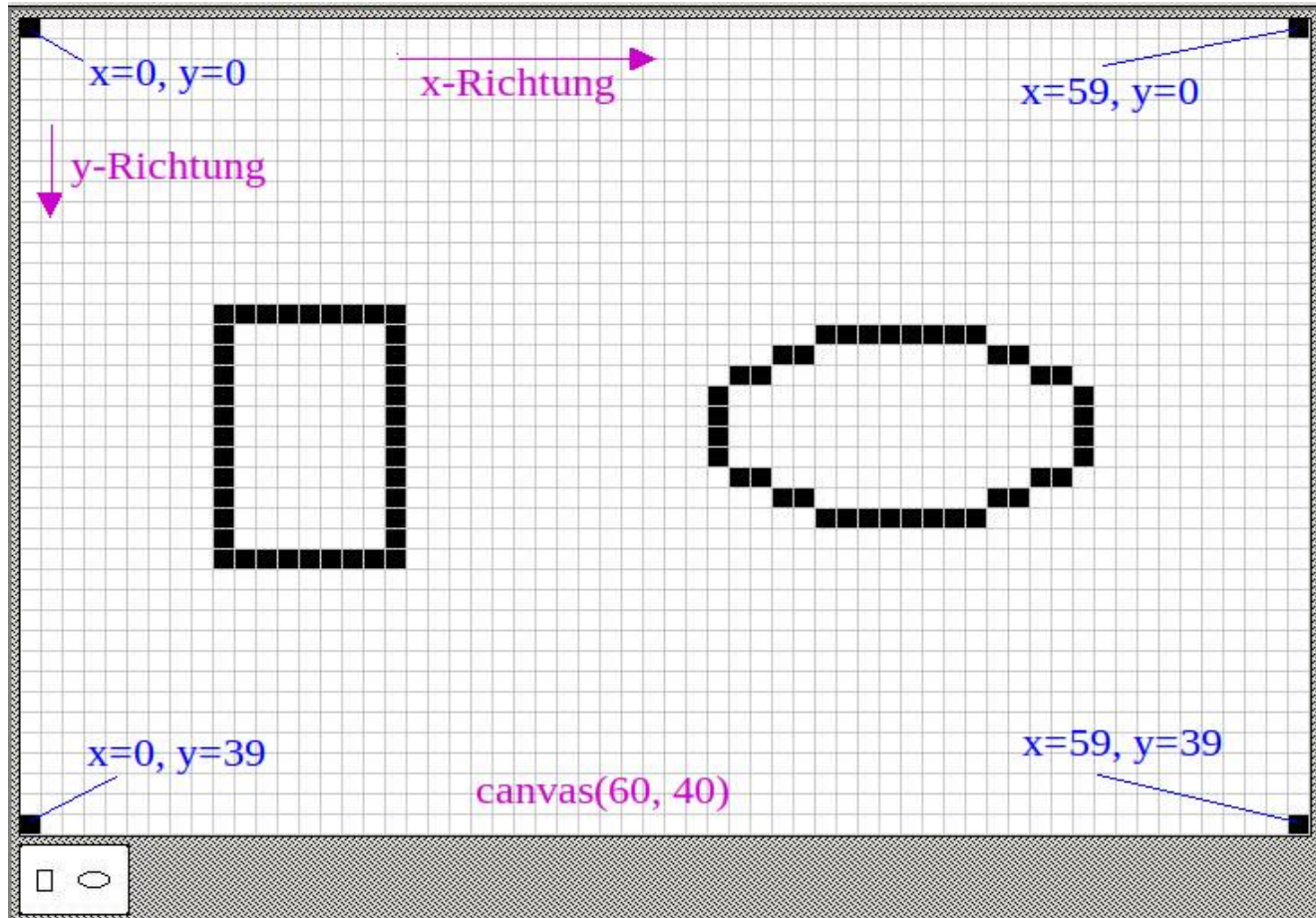
<script>
var leinwand = document.getElementById("meinCanvas").getContext("2d");
</script>

</body>
</html>
```

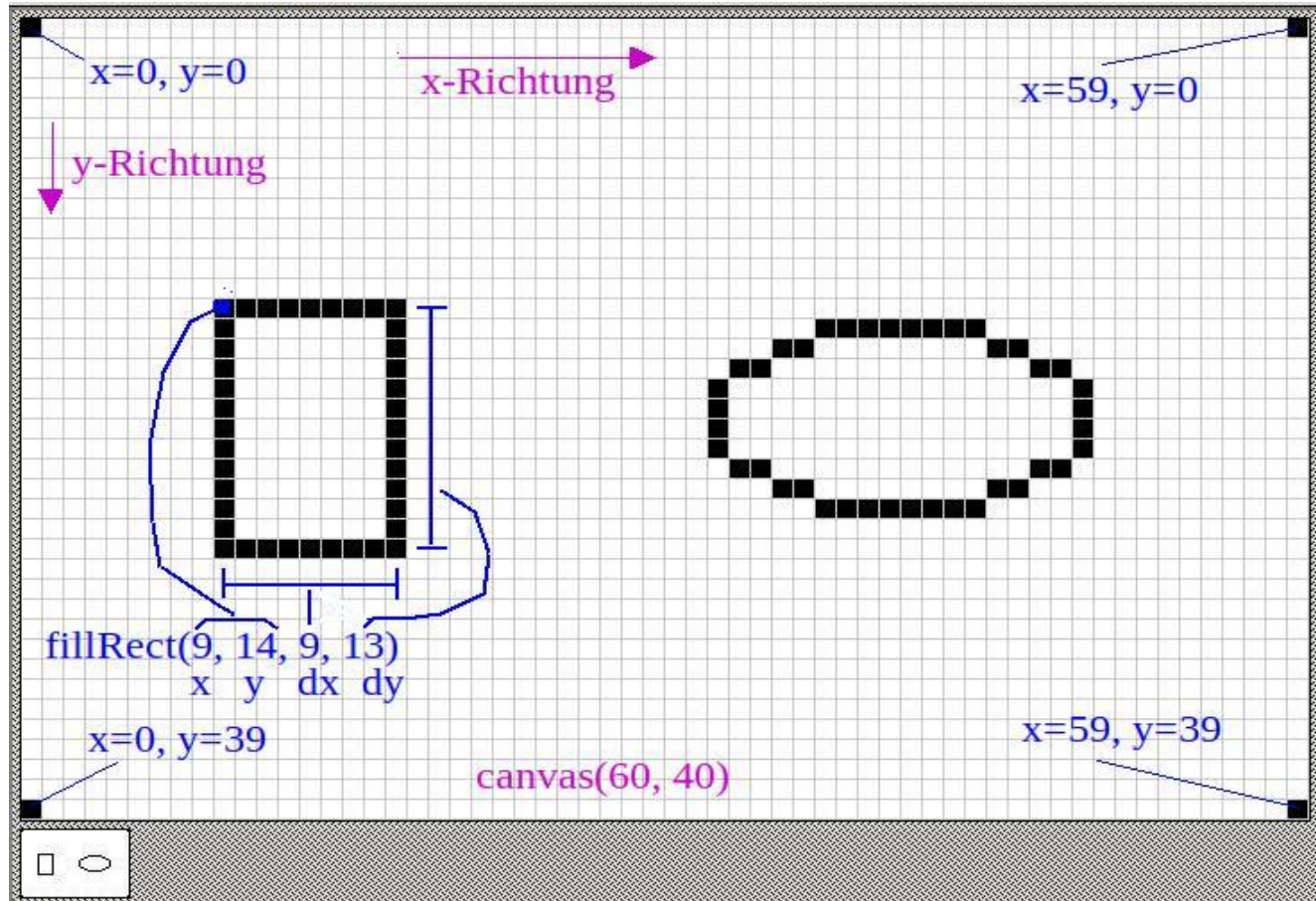
Unsere „Leinwand“



Unsere „Leinwand“



Unsere „Leinwand“



Aufgabe

- Male mehrere Rechtecke in unterschiedlichen Farben auf die Leinwand.

```
leinwand.fillRect(xPosition, yPosition, breite, hoehe);
```

Die Füll-Farbe wechseln:

```
leinwand.fillStyle = "neueFarbe"; // black, red, blue, green, yellow, cyan, ...
```

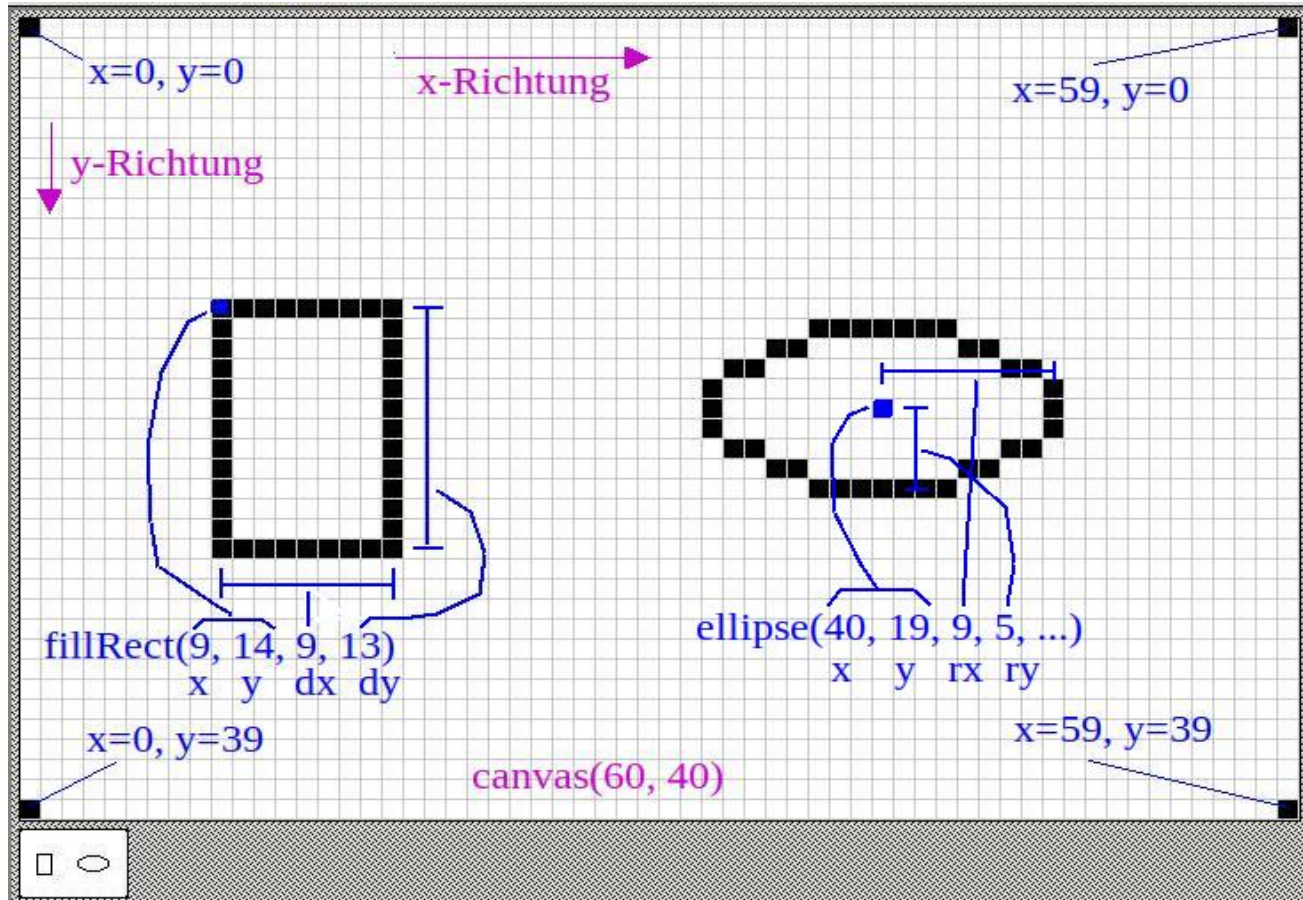
Beispiel:

```
var leinwand = document.getElementById("meinCanvas").getContext("2d");  
leinwand.fillStyle = "cyan";  
leinwand.fillRect(120, 120, 60, 80);  
leinwand.fillStyle = "yellow";  
leinwand.fillRect(150, 150, 100, 5);
```

Coding Style Conventions

- Einrückungen: 3 Leerzeichen
- Basis-Einrückung: keine (ganz links beginnen)
- Ein Leerzeichen vor und hinter alle `= + - * / %` etc.
- Kein Leerzeichen vor und hinter runde Klammern `( )`
- Geschweifte-Klammer-auf `{` in nächste Zeile

Unsere „Leinwand“



Aufgabe

- Ein Rechteck und eine Ellipse in unterschiedlichen Farben auf die Leinwand malen.

Die „ellipse“-Funktion benötigt 7 Parameter. Die letzten 3 Parameter seien immer:

0, 0, 7

Das Malen einer Ellipse muss mit 2 Befehlen eingeleitet und abgeschlossen werden:

```
leinwand.beginPath();  
leinwand.ellipse(xPosition, yPosition, xRadius, yRadius, 0, 0, 7);  
leinwand.fill();
```

Aufgabe

- Die Ellipse so groß machen, dass sie genau in das Rechteck passt, und dort genau hinein platzieren.

Anmerkung:

Ziel dieser Aufgabe ist nur, zu merken, wie kompliziert sie ist.
Sie muss nicht unbedingt gelöst werden!

Convenience Functions

```
function centeredRect(x, y, breite, hoehe)
{
    leinwand.fillRect(x - breite/2 , y - hoehe/2, breite, hoehe);
}
```

```
function centeredEllipse(x, y, breite, hoehe)
{
    leinwand.beginPath();
    leinwand.ellipse(x, y, breite/2, hoehe/2, 0, 0, 2*Math.PI);
    leinwand.fill();
}
```

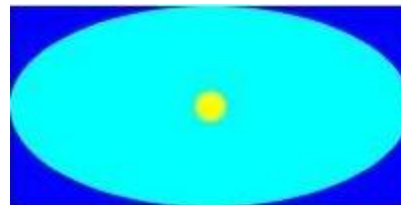
Convenience Functions

```
leinwand.fillStyle = "blue";  
centeredRect(200, 150, 200, 100);
```

```
leinwand.fillStyle = "cyan";  
centeredEllipse(200, 150, 200, 100);
```

```
leinwand.fillStyle = "yellow";  
centeredEllipse(200, 150, 15, 15);
```

```
// xMitte, yMitte, breite, hoehe
```



Aufgabe

- Male ein Raumschiff (oder eine Spielfigur) mit Hilfe unserer Convenience-Funktionen.
- Male als letztes einen Punkt in die Mitte des Raumschiffes.
(Wir werden diesen Punkt später wieder ausblenden.)

maleSchiff

Nun verlagern wir die Befehle zum Malen des Schiffes in eine Funktion, die wir dann mehrfach aufrufen können.

```
// Funktion definieren
function maleRaumschiff(x, y)
{
    ...
    ...
}
```

```
// Funktion aufrufen
maleRaumschiff(100, 100);
```

```
// noch ein Schiff malen
maleRaumschiff(300, 100);
```

Variablen

```
<!DOCTYPE html>
<html>
<head>
  <meta http-equiv="content-type" content="text/html; charset=utf-8" />
</head>
<body>

<script>
  var meineVariable;
  meineVariable = 3 + 4;
  console.log("erste Ausgabe: " + meineVariable);

  meineVariable = meineVariable + 2;
  console.log("zweite Ausgabe: " + meineVariable);
</script>

</body>
</html>
```

Test-Buttons

```
<!DOCTYPE html>
```

```
<html>
```

```
<body>
```

Mein eigenes Spiel

```
<p> <canvas ...> </canvas>
```

```
<p> <button onclick="meineFunc()"> nach rechts </button>
```

```
<script>
```

```
...
```

```
function meineFunc()
```

```
{
```

```
....
```

```
}
```

```
...
```

```
</script>
```

```
</body>
```

```
</html>
```

Das Schiff bewegen

```
<p> <canvas id="meinCanvas" ... > </canvas>
<p> <button onclick="bewegeSchiff()"> nach rechts </button>

<script>
  var leinwand = document.getElementById("meinCanvas").getContext("2d");
  var schiffX = 20;
  var schiffY = 150;

  function bewegeSchiff()
  {
    // Leinwand leeren
    leinwand.fillStyle="#f1f1ff";
    leinwand.fillRect(0,0,400,300);

    // neue x-Position berechnen und Schiff malen
    schiffX = schiffX + 20;
    maleRaumschiff(schiffX,schiffY);
  }
  ...
</script>
```

The diagram illustrates the execution flow of the provided code. An arrow originates from the `bewegeSchiff()` attribute within the `<button>` tag and points to the `bewegeSchiff()` function definition. A second arrow starts from the `bewegeSchiff()` function and points to the `document.getElementById("meinCanvas")` call within the script, indicating that the function interacts with the canvas element.

Aufgabe

- Benenne die Funktion `bewegeSchiff()` um in `bewegeSchiffR()`. Erstelle eine weitere Funktion `bewegeSchiffL()`, um das Raumschiff nach links zu verrücken.
- Erstelle einen weiteren Test-Button, um die Funktion `bewegeSchiffL()` aufrufen zu können.

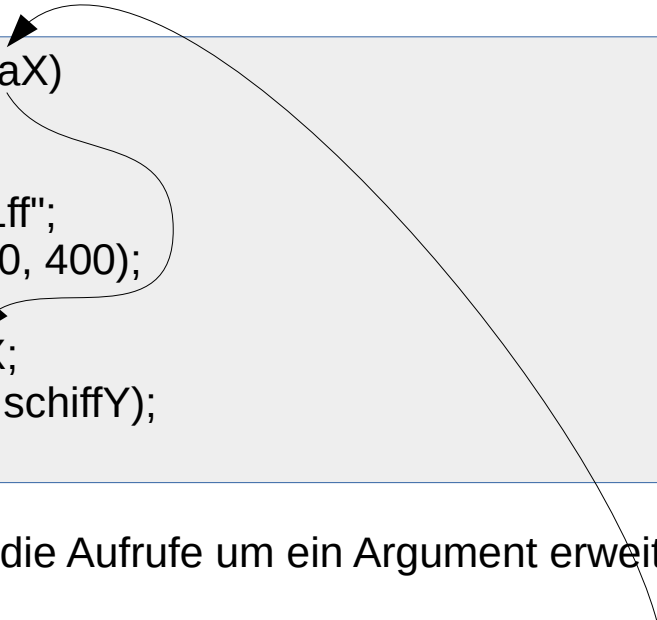
Übergabe-Parameter

Die beiden Funktionen `bewegeSchiffR()` und `bewegeSchiffL()` unterscheiden sich nur an einer Stelle (+10 oder -10).

Deshalb „parametrisieren“ wir diese Stelle nun. Dazu übergeben wir beim Aufruf der Funktion einen Wert, der innerhalb der Funktion unter dem Parameternamen zur Verfügung steht:

```
function bewegeSchiff(deltaX)
{
  // Leinwand putzen
  leinwand.fillStyle = "#f1f1ff";
  leinwand.fillRect(0, 0, 600, 400);

  schiffX = schiffX + deltaX;
  maleRaumschiff(schiffX, schiffY);
}
```



Nun müssen wir aber auch die Aufrufe um ein Argument erweitern:

`onclick="bewegeSchiff(-10)"` bzw. `onclick="bewegeSchiff(10)"`

Hausaufgabe

Füge zwei weitere Test-Buttons "hoch" und "runter" hinzu.

Erweitere dafür zunächst die Funktion "bewegeSchiff(deltaX)", so dass man auch eine Verschiebung in y-Richtung mitgeben kann. Dazu kann man einen 2.Parameter definieren:

```
function bewegeSchiff(deltaX, deltaY) { ... }
```

Hinweis: Beim Aufruf der Funktion muss man für alle Parameter Werte übergeben, deshalb muss nun auch der Aufruf bei den Testbuttons erweitert werden:

```
bewegeSchiff(-10, 0)
```

```
bewegeSchiff(10, 0)
```