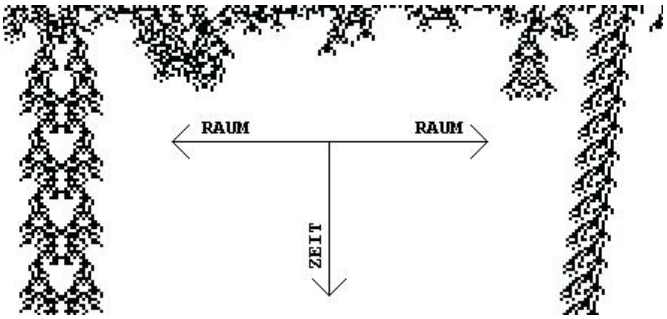
Aufgabe 1 (Felder in JS)

1. Schreiben Sie eine Funktion, die ein Array erzeugt und mit *n* zufälligen Zahlen aus einem selbstgewählten Intervall $[a, b]$ befüllt (mit *a* < 0 und *b* > 0), wobei *a*, *b* und *n* der Funktion als Argumente übergeben werden sollen. Der Rückgabewert der Funktion ist das befüllte Array.
2. Erstellen Sie weiterhin eine Funktion *summe*(), welche die Summe aller Elemente des als Argument übergebenen Arrays berechnet und zurückgibt (bitte alles immer auch testen).
3. Programmieren Sie eine Funktion *groessteZahl*(), welche die größte Zahl zurückgibt, die im obigen Array gespeichert ist (denken Sie daran, dass auch negative Zahlen im Array enthalten sein können oder dass das Feld noch leer sein könnte).
4. a) Geben Sie mit Hilfe einer ‚switch'-Anweisung, welche den Wert von *getDay*() des Date-Objekts[1] als Bedingung nimmt, den aktuellen Wochentag als String aus, wobei die Ausgabe mit Hilfe einer ‚alert'-Box geschehen soll.
   b) Nutzen Sie nun zur Angabe des Tages die Rückgabe von *getDay()* als Index in ein Array!
5. Zeigen Sie an einem konkreten Beispiel, wie mehrdimensionale Arrays erzeugt und als Rückgabe-Wert einer JavaScript-Funktion zurückgeliefert werden können.

Aufgabe 2 (Arbeiten mit Feldern: Zellulärer Automat)

Ein sog. zellulärer Automat ist ein „Modell-Universum" bestehend aus einer Raum- und einer Zeitdimension (vgl. Skizze).



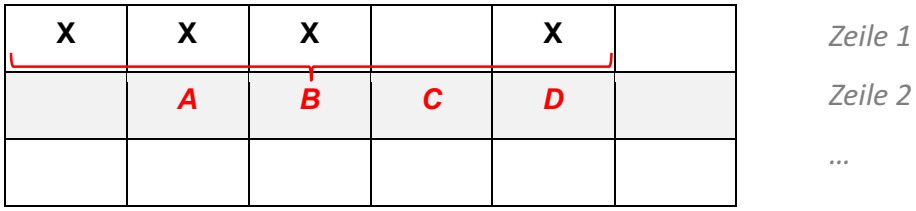
Implementieren Sie in JavaScript einen einfachen zellulären Automaten, bei dem das Verhalten von Lebewesen in einer eindimensionalen Welt simuliert und dargestellt wird. Die besagte Welt besteht dabei aus *n* endlich vielen aneinandergereihten Zellen (oder „Grundstücken"), die entweder leer (0) oder voll (1) sind.

X		X		X	X

Pro Simulations-/ bzw. Zeitschritt wird dargestellt, wie sich die Welt weiterentwickelt, d.h. auf welchen Grundstücken sich bereits ein Lebewesen angesiedelt hat. Dabei wird jeweils eine neue „Zeile" hinzugefügt.

Je nachdem, wie viele benachbarte Lebewesen (oder besser gesagt Vorgänger) existieren, ist die zeitlich nachfolgende Zelle voll oder leer (d.h. also, es wird ein neues Lebewesen erzeugt, bleibt am Leben oder stirbt). Für den Simulationsverlauf ist die Anzahl der Nachbarn (bzw. Vorgänger) wichtig: Eine Zelle (z.B. *B*, s.u.) kann je 0 bis 5 volle Nachbarn haben.

[1] Vgl. z.B. http://www.w3schools.com/jsref/jsref_obj_date.asp



Die „Welt“ (im Rechner repräsentiert als eindimensionales Feld) wird zeilenweise so dargestellt, dass für die Ausgabe pro Zeitschritt genau eine Zeile (wie oben dargestellt ohne Zeilenumbruch) verwendet wird. Verwenden Sie für die Ausgabe wieder das `<pre>`-Element, damit das generierte Muster auch richtig angezeigt wird.
Stellen Sie dabei die beiden Zustände folgendermaßen dar: Leerzeichen `" "` für 0 und Buchstabe `"x"` für 1. Die Anzahl der dargestellten Simulationsschritte soll frei wählbar sein, wobei das Verfahren natürlich möglichst speicherschonend umgesetzt werden soll (also mit maximal zwei Feldern).

Das Anfangsverhalten ergibt sich aus einer zufällig mit 0 oder 1 besetzten Startzeile.
Erzeugen Sie die jeweils nächste Zeile nach der folgenden Regel: Hat eine Zelle 2 oder 4 (volle) Vorgänger, so wird sie auf Zustand 1 gesetzt, ansonsten auf Zustand 0. Für diese Berechnung wird eine Zeile zudem als geschlossen (also „ringförmig“ verbunden) angenommen.
Damit ergibt sich z.B. für die oben gezeigte Situation, dass Zelle A 3 Vorgänger hat, B hat 4, und C sowie D haben wieder 3 Vorgänger.