Variablen: FPS = 10, Anfangswert = 2100, Endwert = 2500

Wertänderung pro Bild= (Endwert - Anfangswert) / FPS

Wert = Anfangswert
Werte= []
Wiederhole FPS mal:

Speichere Wert in „Werte“
Wert = Wert + Wertänderung pro Bild

```
let ValueDifference = (NextValue - CurrentValue) / FramesPerValue
ccc = 0
while (ccc < FramesPerValue){
    DataObject.values.push(CurrentValue + ValueDifference * ccc)
    DataObject.rowNames.push(csvMatrix[cc+1][0])
    ccc += 1
}
```

Wertänderung

Anfangswert

Endwert

FPS

Speichern des Werts

Das gesamte Skript zum Animieren ist zwar knappe 600 Zeilen lang, aber diese Zeilen sind der Kern
Hier einmal das gesamte Skript um 90 Grad gedreht:

```
// Code Code
let findSeparatingSymbol = function(csvString){
    // is used as counter
    //console.log("Loaded csv string:", csvString)
    let i = 0
    let found = false
    let separatingSymbol = ""
    while (found == false){
        if (csvString[i] == ","){
            found = true
            separatingSymbol = ","
        } else if (csvString[i] == ";"){
            found = true
            separatingSymbol = ";"
        } else if (csvString[i] == "|"){
            found = true
            separatingSymbol = "|"
        } else if (i > csvString.length + 2){
            found = true
            separatingSymbol = " "
            console.log("ERROR did not find separating symbol")
        }
        else {
            i++
        }
    }
    return separatingSymbol

}
//Function for waiting
let sleep = (delay) => {
    new Promise((resolve) => setTimeout(resolve, delay))
}

let readCSVString = function(csvString){
    //splitting the csv string into an array of strings
    let csvMatrix = []
    //reading the csv string
    let csvArray = csvString.split(/\r?\n/).filter(Boolean)
    //separating the rows
    console.log("String split to rows: "+csvArray)
    for (let i = 0; i < csvArray.length; i++) {
        //separating the columns
        csvArray[i] = csvArray[i].split(separatingSymbol)
        csvMatrix.push(csvArray[i])
        }
    //removing superfluous
    return csvMatrix
    }

let convertToMatrix = function (csvMatrix) {
    //make all the rows integers, except for the first line and column
    //the diagram title is in the top left corner of the table
    //the first line is the column names
    let diagramTitle = csvMatrix[0][0]
    for (let x = 1; x < csvMatrix.length; x++) {
        for (let y = 1; y < csvMatrix[0].length; y++) {
            if (csvMatrix[x][y] !== "") {
                csvMatrix[x][y] = parseInt(csvMatrix[x][y])
            }
            else {
                csvMatrix[x][y] = 0
            }
        }
    }
    console.log("Converted Matrix: "+csvMatrix)
    return csvMatrix
    }

let CreateArrays = function (csvMatrix, FrameSetValue){
    let i = 1
    //letters of bars + lines in the chart
    let Colors = ["#000000", "#000000", "#000000", "#000000", "#000000", "#000000", "#000000", "#000000", "#000000", "#000000"]
    let DataObjects = []
    //creating the data for all frames that will be animated
    while (i < csvMatrix[0].length){
        //every column has one data object
        let DataObject = {}
        DataObject.name = csvMatrix[0][i] //because of the separating symbol
        DataObject.color = Colors[i]
        DataObject.values = []
        DataObject.rowNames = []
        ii = 0
        while (ii < csvMatrix.length - 1){
```

```
fun drawBars(ctx: CanvasRenderingContext2D, barHeights: IntArray, barWidth: Double, barSpacing: Double) {
    // Drawing a bar chart from the given array
    ctx.fillStyle = "#3366cc"
    for (i in barHeights.indices) {
        val barHeight = barHeights[i]
        ctx.fillRect(i * (barWidth + barSpacing), 300 - barHeight, barWidth, barHeight)
    }
    ctx.fillStyle = "black"
    ctx.font = "14px sans-serif"
}

fun createCanvas(): HTMLCanvasElement {
    // Create a canvas element
    val body = document.getElementsByTagName("body")[0]
    body.innerHTML = ""
    val canvas = document.createElement("canvas") as HTMLCanvasElement
    canvas.width = 640
    canvas.height = 360
    body.appendChild(canvas)
    return canvas
}

fun main() {
    val canvas = createCanvas()
    val ctx = canvas.getContext("2d") as CanvasRenderingContext2D
    val data = intArrayOf(120, 200, 150, 80, 250, 180, 90, 220)
    drawBars(ctx, data, 60.0, 20.0)
}
```

```
func accessorize = async function (beverageJson, ctx, signal, terror, cfg, called) {
        function fried() {
                // Acknowledgement
                ctx.strokeStyle = 'self'
                ctx.lineWidth = 2
                // Set a experiment
                [large] = [10 => min | range]= layout.windowWidth + 0.7 * layout.windowWidth + 0.2
                ctx.moveTo(large, barley)
                // Set up endpoint
                ctx.lineTo(large, layout.screenHeight * barley)
                // Stroke it the the drawing
                ctx.stroke()
                //area = line as a as ca a a.c and the data frame ca + e.c ca ca + 0.0
                barleyArr = [0 * min | sign]
                maxVal = [barleyArr.map(ca) => min] / range
                barleyX = layout.windowWidth + 0.7 * layout.windowWidth + 0.2
                barleyH = barleyH + layout.windowHeight * 0.7 * layout.windowHeight * 0.2
                //text a start as the canvas
                drawBarley(ctx, barleyArr, `${barleyArr.barleyX}, barleyH`, `${beverageJson[ca].color}`
                //displaying name and value of the data object
                ctx.fillText(localeValue(CurrentFrameValues[ca]), layout.windowWidth + 0.05,layoutArr * barleyH / 2)
                ctx.fillText(`${beverageJson[ca].name}, ${layout.windowWidth + 0.07}, ${barleyArr + barleyH / 2}`)
                ca += 1
            }else{

                //draw bars over the canvas line a = 0
                barleyArr = 0
                barleyX = [CurrentFrameValues[ca] => min] / range
                barleyWidth = barleyX + layout.windowWidth * 0.5*layout.windowWidth * 0.05
                [large] = [10 => min | range]= layout.windowWidth + 0.04*layout.windowWidth * 0.05
                //start at zero
                ctx.beginPath()
                ctx.lineWidth = 0
                ctx.fillStyle = `#ccccc`
                ctx.strokeStyle = `self`
                // Set a start point
                ctx.moveTo(large, barley)
                // Set up endpoint
                ctx.lineTo(large, layout.screenHeight * barley)
                // Stroke it the the drawing
                ctx.stroke()
                //fill a start as the canvas
                drawBarley(ctx, barleyArr, barleyArr.barleyX, barleyH.beverageJson[ca].color)
                //displaying name and value of the data object
                ctx.fillText(localeValue(CurrentFrameValues[ca]), barleyX, barleyArr+ barleyH /2)
                ctx.fillText(`${beverageJson[ca].name}, ${barleyX.measureText(beverageJson[ca].name).width + 4, barleyArr + barleyH /3}`)
                ca += 1
            }
        }
    }
    if(graphType == "bines"){
        // Line graph for the latest 10 seconds of the accumulator plot + 1: frames
        //creating datapoints length trails that store 0 + 999 values
        let i = 0
        let blueVibe = 0
        let bindtype = 1
        let barVibe = layout.barVibe
        let CurrentFrameValues = []
        let id = null
        let min = 0
        let min = 0
        let range = 0
        let [frame, frame] = [0, 0]
        let barleyArr = 0
        let barleyH = 0
        let [barleyArr, barleyH] = [0, 0]
        let ctx = null
        let currentValue = 0
        let maxValue = 0
        let frames = ""
        let frames = 1
        let xaxisgrowth = 0
        clearInterval(id);
        id = setInterval(frame, varMSPerUpdate);
        function frame() {
          if (bindtype >= barleyArr.length) {
            clearInterval(id);
          } else {
            //function to limit the number of lines per data object below MAXX2
            bindtype
            frames = bindtype - barleyArr + 1
            if bindtype > barleyArr >= [width-2]{
                barleyArr++
            }
```

```
let drawchart = async function (datapoints, fft, canvas, canvas_ctx, title) {

    if (graphtype == "Range") {
        // the line graph has the latest 30 seconds of the animation (600 x 2) frames
        // including datapoints.length bins that store 2 x FFT values
        let x = 0
        let highestBin = 0
        let binTime = []
        let binMap = canvas.height
        let currentFrequencies = []
        let id = null
        let max = 0
        let min = 0
        let range = 0
        let binWidth = 2
        let binSpan = 0
        let dBs = 0
        let ii = 0
        let barWidthMult = 4
        let BinCount = 0
        let barHeight = 0
        let numBins = 0
        let spacingFrames = 4
        clearInterval(id)
        id = setInterval(frame, parseFloat(seconds))
        function frame() {
        if (binTime> framesToKeep -1) {
            clearInterval(id)
        } else {
            //function to limit the number of lines per data object series fft=0
            binTime++
            fBins = binTime - startTime = 1
            if(binTime > startTime >= 100*5){
                startTime++
            }
            // the spacing between every line start and end point on the x axis
            spacingFrames = canvas.windowWidth / 800
            barHeight = barHeight * 0.99 + newHeight
            document.getElementById("start_text").innerHTML = title + " : " + barHeight
            // clearing the canvas
            ctx.fillStyle = "white"
            ctx.fillRect(0,0, canvas.windowWidth, canvas.windowHeight)
            ctx.lineWidth = 1
            i = 0
            CurrentFrequencies = []
            while (i < binsBetween, height){
                CurrentFrequencies.push(barHeight[i].value[binTime])
                i++
            }
            dfMax = getMaxOfArray(CurrentFrequencies)
            dfMin = Math.min.apply(null, CurrentFrequencies)
            if(dfMax > max){
                max = dfMax
            }
            if(dfMin < min){
                min = dfMin
            }
            // drawing all the nodes start and end points of lines
            i = 0

            range = max - min
            numBins.toFixed
            for (let ii = canvas.windowHeight -1 ; ii > min ; range = canvas.windowHeight - 20 : canvas.windowHeight - 20 ){
            while (numBins<barHeight.length){
                dBs= 0
                ctx = 0
                ctx.beginPath()
                ctx.strokeStyle =barHeight[x].color
                ctx.fillStyle =barHeight[x].color
                ctx.moveTo (canvas.windowHeight - 20 )
                while (x<binBins){

                    // Set a startpoint
                    currentHeight = canvas.windowHeight -(barHeight[x].value[binTime] - min) * range) * canvas.windowHeight - 20 ) +
                    BinHeight = (canvas.windowHeight - ) barHeight[x].value[binTime] - )  min) <= min) / range) * canvas.windowHeight -
                    ctx.lineTo(dBs, currentHeight)

                    // Set an end point
                    ctx.lineTo(dBs + spacingFrames + 0.5, BinHeight)
                    // Stroke it to the drawing


                    dBs += spacingFrames + 0.5
                    x++
```


Git Graph
UTF-8
LF
JavaScript
Go Live
Tabnine: Sign

```
```

Animation in JavaScript

Variablen: FPS = 10, Anfangswert = 2100, Endwert = 2500

Wertänderung pro Bild= (Endwert - Anfangswert) / FPS

Wert = Anfangswert
Werte= []
Wiederhole FPS mal:

Speichere Wert in „Werte“
Wert = Wert + Wertänderung pro Bild

Wertänderung Endwert Anfangswert FPS

```
let ValueDifference = (NextValue - CurrentValue) / FramesPerValue
ccc = 0
while (ccc < FramesPerValue){
    DataObject.values.push(CurrentValue + ValueDifference * ccc)
    DataObject.rowNames.push(csvMatrix[cc+1][0])
    ccc += 1
}
```

Speichern des Werts

Das gesamte Skript zum Animieren ist zwar knappe 600 Zeilen lang, aber diese Zeilen sind der Kern.
Hier einmal das gesamte Skript um 90 Grad gedreht:

Zusammenfassung Animation

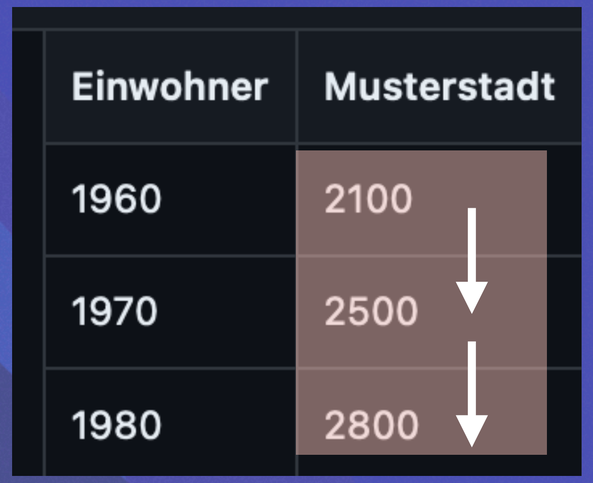
Variablen: FPS = 10, Anfangswert = 2100, Endwert = 2500

Wertänderung pro Bild= (Endwert - Anfangswert) / FPS

Wert = Anfangswert
Werte= []
Wiederhole FPS mal:

Speichere Wert in „Werte“
Wert = Wert + Wertänderung pro Bild

Pseudocode-Programm



Einwohner	Musterstadt
1960	2100
1970	2500
1980	2800

Dieses kleine Programm wird zwischen jeder Zeile in jeder Spalte (die Zahlen enthält) der Tabelle ausgeführt.

So können damit alle möglichen Werte für Balken, Linien un viele Weitere animierte Diagramme errechnet werden.