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 (40)

Anfangswert (2100)

Endwert (2500)

FPS (10)

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 (Stand 13.6.2024):

```
// 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, timer, cfg, cache) {
    function fizz() {
        // Acknowledgement
        ctx.strokeStyle = '#fff'
        ctx.lineWidth = 1
        // Set a expression
        [args] = [1 => min  | range]= layout.windowWidth + 0.7 * layout.windowWidth + 0.2
        ctx.moveTo(args, barMap)
        // Set up endpoint
        ctx.lineTo(args, layout.screenHeight * barMap)
        // Stroke it on the drawing
        ctx.stroke()
        // time = time or e => (e - min) / range
        barStart = 0 * min | range
        barEndX = (currentFrameValues[x] - min) / range
        barStartX = barStartX + layout.windowWidth * 0.7 * layout.windowWidth * 0.2
        barEndX = barEndX + layout.windowWidth * 0.7 * layout.windowWidth * 0.2
        // Fill a rect on the canvas
        drawRect(ctx, barStartX,  BarStartY.barEndX, barHeight, DataObjects[x].color)
        // Subtracting name and value of the data object
        ctx.fillText(scaleValue(currentFrameValues[x] ), layout.windowWidth + 0.05,barStartY + barHeight / 2)
        ctx.fillText(DataObjects[x].name, layout.windowWidth + 0.07), barStartY + BarHeight / 2)
        }) }
    else{

        // Draw bars over the canvas from x = 0
        barStartX = 0
        barEndX = (currentFrameValues[x] - min) / range
        barStartX = barStartX * layout.windowWidth * 0.5*layout.windowWidth * 0.05
        [args] = [1 => min  | range]= layout.windowWidth + 0.04*layout.windowWidth * 0.05
        // Start at zero
        ctx.beginPath()
        ctx.lineWidth = 0
        ctx.fillStyle = "#ffffff"
        ctx.strokeStyle = '#fff'
        // Set a start point
        ctx.moveTo(barEnd, barMap)
        // Set up endpoint
        ctx.lineTo(barEnd, layout.screenHeight * barMap)
        // Stroke it on the drawing
        ctx.stroke()
        // Fill a rect on the canvas
        drawRect(ctx, barStartX, barStartX.lastEnd, barHeight.DataObjects[x].color)
        // Subtracting name and value of the data object
        ctx.fillText(scaleValue(currentFrameValues[x] ), barEnd, barStartY+ barHeight /2)
        ctx.fillText(DataObjects[x].name, barEnd - ctx.measureText(DataObjects[x].name.width + 4, barStartY + barHeight /2)
        }) }
    }
  }
}
if graphType == "Color"{
    // Line graph for the latest 10 seconds of the animation (60 * 6) frames
    // Creating dataObjects.length brats that store 0 - 255 values
    let r = 0
    let blueValue = 0
    let brakTime = 1
    let barSize = layout.barSize
    let currentFrameValues = []
    let id = nil
    let min = 0
    let max = 0
    let range = 0
    let [frame, frameCount] = [0, 0]
    let currentValue = 0
    let maxValue = 0
    let currentKey = ""
    let broken = 1
    let saveLightness = 0
    class Interval {}
    ID = setInterval(frame, canMoveToLocation)
    function frame() {
      if (brakTime, frameCount) {
        clearInterval(ID);
      } else {
        //Function to limit the number of lines per data object below 60fps()
        brakTime
        broken = brakTime = barSize = 1
        if brakTime = blueValue >= [width-2]{
            barSize++
        }
```

```
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++
```


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

```
```

Animation in JavaScript

Wertänderung pro Bild= (Endwert - Anfangswert) / FPS

Wert = Anfangswert
Werte= []
Wiederhole FPS mal:

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

Wertänderung (40) Endwert (2500) Anfangswert (2100) FPS (10)

```
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 (Stand 13.6.2024):

Zusammenfassung Animation

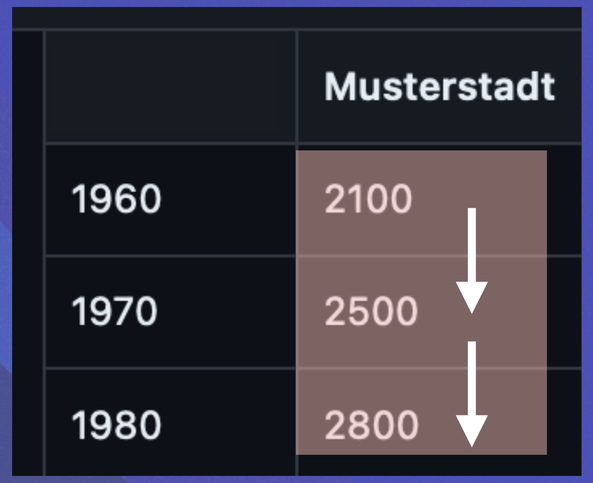
Variablen: FPS = 10, Anfangswert = 2100, Endwert = 2500

Wertänderung pro Bild= (Endwert - Anfangswert) / FPS

Wert = Anfangswert
Werte= []
Wiederhole FPS mal:

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

Pseudocode-Programm



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.