OULUN AMMATTIKORKEAKOULU

YCharts ja MathParser -kirjastojen ohjedokumentti

Jetpack Compose -pohjaisten kuvaajien ja graafisten laskimien toteuttaminen YCharts ja MathParser kirjastoja hyödyntämällä.

Tomi Lakkakorpi
Ammattiharjoittelu
Kevät 2025
Tietotekniikan tutkinto-ohjelma
Oulun ammattikorkeakoulu

SISÄLLYSLUETTELO

1 KIRJASTOT

1.1 YCharts

Ycharts on Jetpack Compose -pohjainen helppokäyttöinen kirjasto, jonka avulla sovelluksiin voidaan helposti integroida erilaisia kuvaajia visuaalisesti kuvastamaan tilastoja ja dataa. Kirjasto on muihin samankaltaisiin kirjastoihin verrattuna uusi. Dokumentin tekohetkellä kirjaston uusin versio 2.1.0 julkaistiin 30.6.2023 ja se tukee seitsemää erilaista kaavioita. Nämä ovat viivakaavio, pylväskaavio, aaltokaavio, kuplakaavio, yhdistetty kaavio, piirakkakaavio ja donitsikaavio. Ohjedokumentissa käydään läpi miten näitä kaavioita voidaan hyödyntää useiden esimerkkien avulla. Suositeltavaa on käydä esimerkit järjestyksessä läpi. Myöhemmissä esimerkeissä ei välttämättä käydä aiempien esimerkkien asioita uudelleen läpi. Varsinkin graafisissa laskimissa keskitytään pääosin kaavoihin ja arvojen laskemiseen, eikä niinkään siihen, miten kuvaaja saadaan piirrettyä.

YCharts Github: https://github.com/codeandtheory/YCharts

1.1.1 Käytön edellytykset

YCharts kirjasto vaatii toimiakseen vähintään API/SDK tason 26. Projektin API/SDK tason voi tarkistaa build.gradle.kts (Module :app) tiedostosta. Jos muutit API/SDK tasoa, muista synkronoida projekti sync now näppäimellä

1.1.2 Kirjaston käyttöönotto

YCharts kirjaston käyttöönotto tapahtuu lisäämällä alla oleva implementointi build.gradle.kts (Module :app) -tiedostoon. Muista synkronoida projekti ennen jatkamista.

```
dependencies {
    //YCharts implementointi
    implementation("co.yml:ycharts:2.1.0")
```

implementation("co.yml:ycharts:2.1.0")

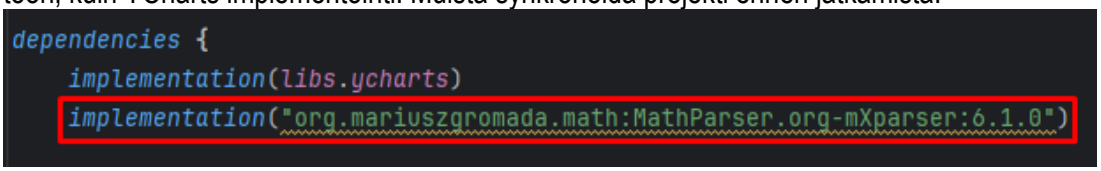
1.2 MathParser

MathParser on suosittu ja helppokäyttöinen työkalu matemaattisten laskujen toteuttamiseen. Kirjastoa on ladattu yli 4,5 miljoonaa kertaa ja se tukee lukuisia eri ohjelmointikieliä. Esimerkeissä kirjastoa käytetään luvussa 3 graafisten laskinten kanssa erilaisiin laskutoimituksiin.

MathParser Github: https://github.com/mariuszgromada/MathParser.org-mXparser

1.2.1 Käyttöönotto

MathParser kirjaston käyttöönotto tapahtuu lisäämällä alla oleva implementointi samaan tiedostoon, kuin YCharts implementointi. Muista synkronoida projekti ennen jatkamista.



implementation("org.mariuszgromada.math:MathParser.org-mXparser:6.1.0")

2 KAAVIOESIMERKIT

Tässä luvussa käydään läpi kaikkien YCharts -version 2.1.0 kuvaajien piirtäminen. Esimerkeissä käytetään esimerkkidatana kovakoodattua dataa. Kaikki luvun esimerkit löytyvät Github repositorystä **ChartApplications** -sovelluksesta suomeksi kommentoituna.

Github: https://github.com/TomiLakkakorpi/KotlinChartApps

2.1 Pylväskaavio – BarChartAppV1

Pylväskaavioita käytetään yleisesti eri luokkien ja ryhmien arvojen vertailuun. Esimerkkidatana käytetään OAMK:n hakijamääriä välillä kevät 2023 – syksy 2025.

Luodaan dynaaminen lista, johon lisätään BarData tyyppisiä elementtejä. Jokaisella BarData elementillä on viisi parametria, joista ainoastaan point on pakollinen. Mikäli muita parametreja ei syötetä, kirjasto käyttää sen vakioarvoja kyseisille parametreille. Pylväsdiagrammissa x -akselin arvo kuvastaa monesko pylväs on kyseessä ja y -akseli puolestaan kuvastaa pylvään korkeutta, eli pylvään arvoa.

```
val dataList = arrayListOf(
    BarData(point = Point(1F, 3135F), color = color1, label = "K 2023"),
    BarData(point = Point(2F, 11388F), color = color2, label = "S 2023"),
    BarData(point = Point(3F, 5307F), color = color3, label = "K 2024"),
    BarData(point = Point(4F, 12528F), color = color4, label = "S 2024"),
    BarData(point = Point(5F, 3198F), color = color5, label = "K 2025"),
    BarData(point = Point(6F, 10956F), color = color6, label = "S 2025"),
)
```

Esimerkissä pylväiden data on kovakoodattu. Dynaamiseen listaan voidaan kuitenkin lisätä ja poistaa elementtejä. Oikeassa käyttötapauksessa data lisättäisiin ja poistettaisiin esimerkiksi näin. Datan lisääminen ja poistaminen käydään läpi myöhemmissä esimerkeissä graafisissa laskimissa.

```
//Lisääminen
dataList.add(BarData(point = Point(xValue, yValue)))

//Poistaminen
dataList.removeAt(index)
```

Luodaan arvot pylväiden maksimiarvolle sekä askelien määrälle y-akselilla. Maksimiarvon voi lisätä myös siten, että hakee listan maksimiarvon list.maxOf -funktiolla. Kun maksimiarvo on 13 000 ja askelien määrä 13, saadaan y-akselille askeleet 1 000 välein.

```
val maxRange = 13000
val yStepSize = 13
```

Luodaan xAxisData arvo, jossa määritellään x -akselin eri parametreja. Esimerkissä näistä käytetään vain osaa, lisää muutettavia parametreja löydät mm. pitämällä hiiren AxisData kohdalla ja painamalla esiin ilmestyvän ikkunan oikeasta alareunasta kynän kuvaketta.

```
val xAxisData = AxisData.Builder()
    .axisStepSize(30.dp)
    .steps(dataList.size - 1)
    .bottomPadding(60.dp)
    .axisLabelAngle(45f)
    .labelAndAxisLinePadding(10.dp)
    .startDrawPadding(20.dp)
    .labelData { index -> dataList[index].label }
    .build()
```

Luodaan myös yAxisData arvo, jossa määritellään samalla tavalla y -akselille eri parametreja. Lisää muutettavia parametreja löydät samalla tavalla kuin xAxisData kohdassa.

```
val yAxisData = AxisData.Builder()
    .steps(yStepSize)
    .labelAndAxisLinePadding(20.dp)
    .axisOffset(10.dp)
    .labelData { index -> (index * (maxRange / yStepSize)).toString() }
    .build()
```

Seuraavaksi luodaan barChartData arvo, johon lisätään dataksi aiemmin luomamme listan sekä x- ja y-akselien konfiguraatiot.

```
val barChartData = BarChartData(
    chartData = dataList,
    xAxisData = xAxisData,
    yAxisData = yAxisData,
    barStyle = BarStyle(
        paddingBetweenBars = 25.dp,
        barWidth = 20.dp
    ),
    showYAxis = true,
    showXAxis = true,
    horizontalExtraSpace = 50.dp
)
```

Lopuksi kutsutaan BarChart -funktiota, joka piirtää kuvaajan. Funktiolle tulee syöttää arvo, joka sisältää pylväiden datan sekä x- ja y-akselien konfiguraatiot. Muut ominaisuudet kuten esimerkissä kuvaajan korkeus, ovat vapaaehtoisia.

```
BarChart(
    modifier = Modifier
        .height(350.dp),
    barChartData = barChartData
)
```

2.2 Viivakaavio – LineChartAppV1

Viivakaavioita käytetään yleensä trendien ja muutosten visualisointiin jonkin ajanjakson aikana. Esimerkkidatana käytetään Nokian osakkeen arvoa välillä tammikuu 2024 – joulukuu 2024.

Luodaan dynaaminen lista, johon lisätään pisteitä koordinaatistossa.

```
val dataList = arrayListOf(
    Point(1f, 3.332f),
    Point(2f, 3.260f),
    Point(3f, 3.291f),
    Point(4f, 3.412f),
    Point(5f, 3.591f),
    Point(6f, 3.558f),
    Point(7f, 3.621f),
    Point(8f, 3.978f),
    Point(9f, 3.924f),
    Point(10f, 4.325f),
    Point(11f, 3.980f),
    Point(12f, 4.274f)
)
```

Luodaan xAxisData arvo, jossa määritellään x -akselin eri parametreja. Esimerkissä näistä käytetään vain osaa, lisää muutettavia parametreja löydät mm. pitämällä hiiren esimerkiksi AxisData kohdalla ja painamalla esiin ilmestyvän ikkunan oikeasta alareunasta kynän kuvaketta.

```
val xAxisData = AxisData.Builder()
    .axisStepSize(35.dp)
    .topPadding(105.dp)
    .steps(dataList.size - 1)
    .labelData { i -> dataList[i].x.toInt().toString() }
    .labelAndAxisLinePadding(15.dp)
    .build()
```

Luodaan myös yAxisData arvo, jossa määritellään samalla tavalla y -akselille eri parametreja. Lisää muutettavia parametreja löydät samalla tavalla kuin xAxisData kohdassa.

```
val yAxisData = AxisData.Builder()
    .steps(yAxisSteps)
    .labelAndAxisLinePadding(30.dp)
    .labelData { i ->
        val yMin = dataList.minOf{it.y}
        val yMax = dataList.maxOf{it.y}
        val yScale = (yMax - yMin) / yAxisSteps
        ((i * yScale) + yMin).formatToSinglePrecision()
    }.build()
```

Luodaan data -arvo, johon lisätään dataksi aiemmin luomamme lista, sekä x- ja y-akselien konfiguraatiot.

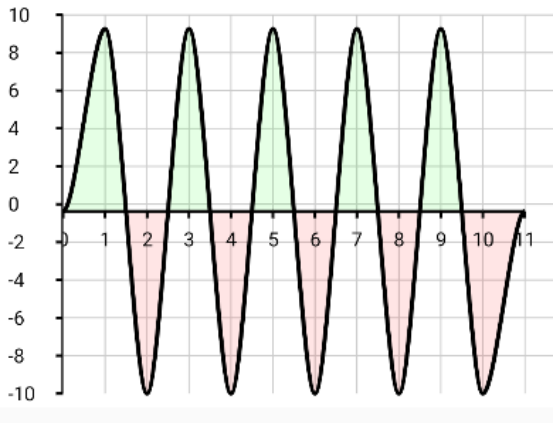
```
val data = LineChartData(
    linePlotData = LinePlotData(
        lines = listOf(
            Line(
                dataPoints = dataList,
                LineStyle(),
                IntersectionPoint(),
                SelectionHighlightPoint(),
                ShadowUnderLine(),
                SelectionHighlightPopUp()
            )
        )
    ),
    xAxisData = xAxisData,
    yAxisData = yAxisData,
    gridLines = GridLines()
)
```

Kutsutaan LineChart -funktiota, joka piirtää kaavion. Funktiolle tulee syöttää luomamme data -arvo, joka sisältää datalistan sekä x- ja y-akselien konfiguraatiot. Muut ominaisuudet kuten esimerkissä kuvaajan korkeus, ovat vapaaehtoisia.

```
LineChart(
    modifier = Modifier
        .fillMaxWidth()
        .height(300.dp),
    lineChartData = data
)
```

2.3 Aaltokaavio - WaveChartAppV1

Aaltokaavioita käytetään monilla aloilla kuten taloudessa, tekniikan aloilla ja lääketieteessä. Aaltokaavioita voidaan käyttää esittämään jonkinlaista signaalidataa tai ääniaaltoja, seismistä toimintaa sekä sähkömagneettisia aaltoja. Lääketieteessä aaltokaavioita käytetään mm. sydämen ja aivojen toiminnan seuraamiseen.



Luodaan dynaaminen lista, johon lisätään pisteitä.

```
val dataList = arrayListOf(
    Point(1f, 0f),
    Point(2f, 10f),
    Point(3f, -10f),
    Point(4f, 10f),
    Point(5f, -10f),
    Point(6f, 10f),
    Point(7f, -10f),
    Point(8f, 10f),
    Point(9f, -10f),
    Point(10f, 10f),
    Point(11f, -10f),
    Point(12f, 0f)
)
```

Luodaan xAxisData arvo, jossa säädetään parametreja x -akselille.

```
val xAxisData = AxisData.Builder()
    .axisStepSize(30.dp)
    .startDrawPadding(48.dp)
    .steps(dataList.size - 1)
    .labelData { i -> i.toString() }
    .labelAndAxisLinePadding(15.dp)
    .build()
```

Luodaan yAxisData arvo, jossa säädetään parametreja y -akselille

```
val yAxisData = AxisData.Builder()
    .steps(yAxisSteps)
    .labelAndAxisLinePadding(20.dp)
    .labelData { i ->
        val yMin = dataList.minOf { it.y }
        val yMax = dataList.maxOf { it.y }
        val yScale = (yMax - yMin) / yAxisSteps
        ((i * yScale) + yMin).formatToSinglePrecision()
    }.build()
```

Luodaan data -arvo, joka sisältää kaikki parametrit, jotka on määriteltävä aaltokaavion piirtämiseksi. Lisätään data -arvoon lista, joka sisältää kaavion datan sekä konfiguroidaan muita kaavion parametreja kuten väriasetuksia.

```
val data = WaveChartData(
    wavePlotData = WavePlotData(
        lines = listOf(
            Wave(
                dataPoints = dataList,
                waveStyle = LineStyle(color = Color.Black),
                selectionHighlightPoint = SelectionHighlightPoint(),
                shadowUnderLine = ShadowUnderLine(),
                selectionHighlightPopUp = SelectionHighlightPopUp(),
                waveFillColor = WaveFillColor(topColor = Color.Green, bottomColor = Color.Red),
            )
        )
    ),
    xAxisData = xAxisData,
    yAxisData = yAxisData,
    gridLines = GridLines()
)
```

Kutsutaan funktiota joka piirtää aaltokaavion, syötetään sille luomamme data-arvo joka sisältää kaikki tarvittavat konfiguraatiot.

```
WaveChart(
    modifier = Modifier
        .fillMaxWidth()
        .height(300.dp),
    waveChartData = data
)
```

2.4 Piirakkakaaviot

2.4.1 PieChartAppV1

Piirakkakaavioita käytetään yleensä näyttämään eri luokkien osuutta jostain kokonaisuudesta, erityisesti kun luokkia on rajallinen määrä. Esimerkkidatana käytetään suomen kymmenen suurimman kaupungin väkilukua.

Luodaan dynaaminen lista, johon lisätään Slice -tyyppisiä elementtejä. Jokaisella Slice -elementillä on neljä parametria, jotka ovat otsikko, arvo, väri ja kuvaus.

```
val dataList = PieChartData(
    slices = arrayListOf(
        PieChartData.Slice("Helsinki", 684018f, color = color1),
        PieChartData.Slice("Espoo", 320931f, color = color2),
        PieChartData.Slice("Tampere", 260180f, color = color3),
        PieChartData.Slice("Vantaa", 251269f, color = color4),
        PieChartData.Slice("Oulu", 216152f, color = color5),
        PieChartData.Slice("Turku", 206073f, color = color6),
        PieChartData.Slice("Jyväskylä", 149194f, color = color7),
        PieChartData.Slice("Kuopio", 125666f, color = color8),
        PieChartData.Slice("Lahti", 121337f, color = color9),
        PieChartData.Slice("Pori", 83305f, color = color10)
    ),
    plotType = PlotType.Pie
)
```

Seuraavaksi luodaan pieChartConfig arvo, jossa määritellään kaaviolle eri parametreja. lisää muutettavia parametreja löydät mm. pitämällä hiiren esimerkiksi PieChartConfig kohdalla ja painamalla esiin ilmestyvän ikkunan oikeasta alareunasta kynän kuvaketta.

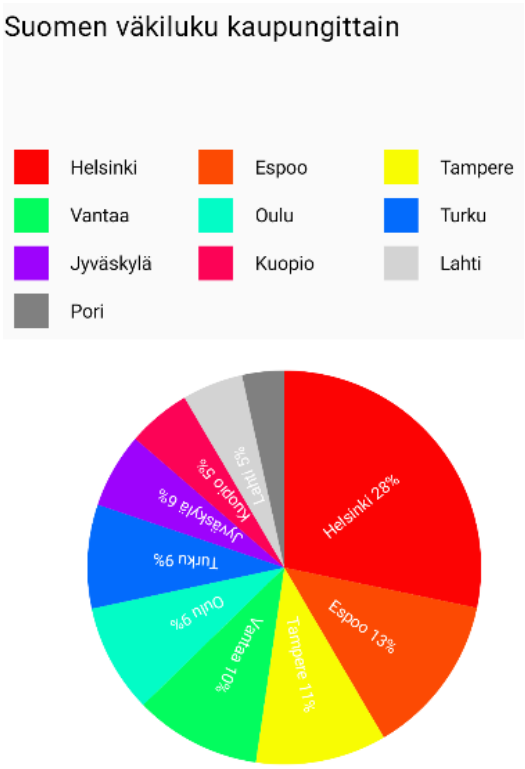
```
val pieChartConfig = PieChartConfig(
        labelVisible = true,
        activeSliceAlpha = .9f,
        isEllipsizeEnabled = true,
        sliceLabelEllipsizeAt = TextUtils.TruncateAt.MIDDLE,
        isAnimationEnable = true,
        chartPadding = 30,
        backgroundColor = Color.White,
        showSliceLabels = false,
        animationDuration = 2000,
    )
```

Luodaan sarakekomponentti ja kutsutaan funktiota, joka piirtää piirakkakaavion selitteet ruudulle (Legends). Kutsutaan myös funktiota, joka piirtää itse piirakkakaavion (PieChart). Funktiolle tulee syöttää datalistamme sekä piirakkakaavion konfiguraatio.

```
Column(modifier = Modifier.height(500.dp)) {
    Spacer(modifier = Modifier.height(20.dp))
    Legends(legendsConfig = DataUtils.getLegendsConfigFromPieChartData(dataList, 3))
    PieChart(
        modifier = Modifier
            .fillMaxWidth()
            .height(400.dp),
        dataList,
        pieChartConfig
    ) {}
}
```

2.4.2 PieChartAppV2

Jatketaan edellistä esimerkkiä lisäämällä piirakkakaavion siivuihin otsikkotekstit. Tämä muutos saadaan aikaan vain yhtä arvoa muuttamalla, sillä kirjasto tukee tätä ominaisuutta.



Konfiguraatiotiedostossa muutetaan showSliceLabels -arvo todeksi, niin otsikot näkyvät piirakkakaaviossa.

```
val pieChartConfig =
    PieChartConfig(
        labelVisible = true,
        activeSliceAlpha = .9f,
        isEllipsizeEnabled = true,
        sliceLabelEllipsizeAt = TextUtils.TruncateAt.MIDDLE,
        isAnimationEnable = true,
        chartPadding = 30,
        backgroundColor = Color.White,
        showSliceLabels = true,
        sliceLabelTextSize = 12.sp,
        sliceLabelTextColor = Color.White,
        animationDuration = 2000,
    )
```

2.4.3 PieChartAppV3

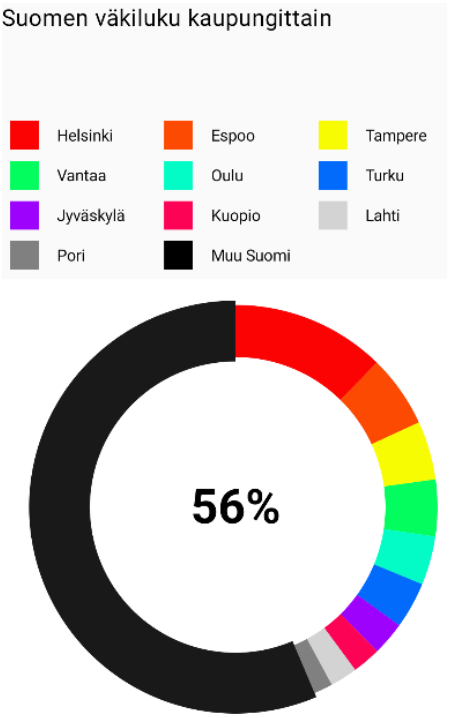
Esimerkin data on tällä hetkellä puutteellinen. Piirakkakaaviot kuvaavat yleensä eri luokkien osuutta kokonaisuudesta, mutta tällä hetkellä datana on suomen kymmenen suurinta kaupunkia, mutta muun suomen väestö ei tule kaaviosta ilmi vielä millään tavalla. Lisätään esimerkkiin siis myös muun suomen väestö.

Lisätään uusi siivu dynaamiseen datalistaan ja syötetään sille tarvittavat tiedot.

```
val dataList = PieChartData(
    slices = listOf(
        PieChartData.Slice("Helsinki", 684018f, color = color1),
        PieChartData.Slice("Espoo", 320931f, color = color2),
        PieChartData.Slice("Tampere", 260180f, color = color3),
        PieChartData.Slice("Vantaa", 251269f, color = color4),
        PieChartData.Slice("Oulu", 216152f, color = color5),
        PieChartData.Slice("Turku", 206073f, color = color6),
        PieChartData.Slice("Jyväskylä", 149194f, color = color7),
        PieChartData.Slice("Kuopio", 125666f, color = color8),
        PieChartData.Slice("Lahti", 121337f, color = color9),
        PieChartData.Slice("Pori", 83305f, color = color10),

        PieChartData.Slice("Muu Suomi", 3129089f, color = color11)
    ),
    plotType = PlotType.Pie
)
```

2.5 Donitsikaavio – DonutChartAppV1

Piirakkakaavio voidaan myös muuttaa donitsikaavioksi. YCharts donitsikaavion etuna piirakkakaavioon on se, että donitsikaaviossa kun siivua painetaan, donitsin keskellä näytetään siivun arvo. Siivun arvon sijasta on myös mahdollista näyttää donitsin keskellä siivun prosenttiosuus. Tämä tapahtuu muuttamalla labelType -parametria konfiguraatiotiedostossa.



Piirakkakaavio saadaan muutettua donitsikaavioksi, kun vaihdetaan plotType arvoa.

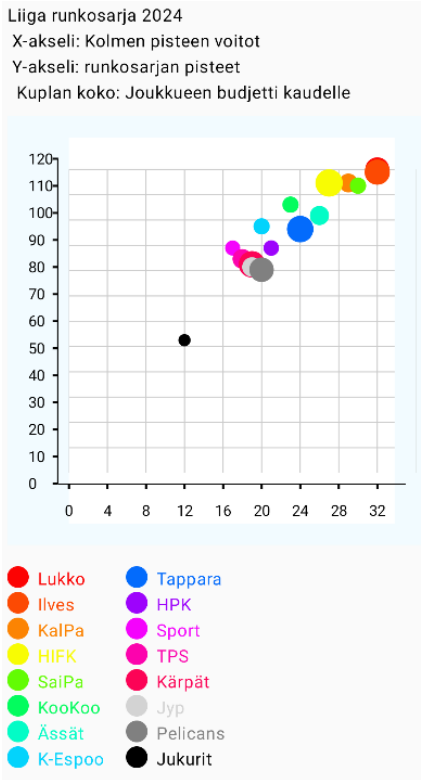
```
val dataList = PieChartData(
    slices = listOf(
        PieChartData.Slice("Helsinki", 684018f, color = color1),
        PieChartData.Slice("Espoo", 320931f, color = color2),
        PieChartData.Slice("Tampere", 260180f, color = color3),
        PieChartData.Slice("Vantaa", 251269f, color = color4),
        PieChartData.Slice("Oulu", 216152f, color = color5),
        PieChartData.Slice("Turku", 206073f, color = color6),
        PieChartData.Slice("Jyväskylä", 149194f, color = color7),
        PieChartData.Slice("Kuopio", 125666f, color = color8),
        PieChartData.Slice("Lahti", 121337f, color = color9),
        PieChartData.Slice("Pori", 83305f, color = color10),
        PieChartData.Slice("Muu Suomi", 3129089f, color = color11)
    ),
    plotType = PlotType.Donut
)
```

Muistetaan myös kutsua DonutPieChart -funktiota, jotta oikea kaavio saadaan piirrettyä.

```
Column(modifier = Modifier.height(500.dp)) {
    Spacer(modifier = Modifier.height(20.dp))
    Legends(legendsConfig = DataUtils.getLegendsConfigFromPieChartData(pieChartData = dataList, 3))
    DonutPieChart(
        modifier = Modifier
            .fillMaxWidth()
            .height(400.dp),
        dataList,
        donutChartConfig
    ) {}
}
```

2.6 Kuplakaavio – BubbleChartAppV1

Kuplakaaviot ovat hyviä havainnollistamaan dataa, kun parametreja on kolme. Esimerkkidatana käytetään datana Liigan 2024 runkosarjan tilastoja. X -akselilla on joukkueiden suorat kolmen pisteen voitot. Y-akselilla on joukkueen keräämät runkosarjan pisteet. Kuplan koko puolestaan kuvaa joukkueen budjetin kokoa kaudelle. Kaaviolla voidaan siis havainnollistaa, suhdetta joukkueen budjetin ja menestyksen välillä.



Luodaan dynaaminen lista, johon lisätään Bubble -elementtejä. Jokaiselle kuplalle lisätään keskipiste (x ja y -arvot), density (kuplan koko) sekä muita parametreja kuten väriasetuksia.

```
val dataList = arrayListOf(
        Bubble( //Lukko
            center = Point(32F, 112F),
            density = 29.50F,
            bubbleStyle = BubbleStyle(solidColor = color1),
            selectionHighlightPoint = SelectionHighlightPoint(Color.Black),
            selectionHighlightPopUp = SelectionHighlightPopUp(Color.Cyan)
        ),

        Bubble(//Ilves
            center = Point(32F, 111F),
            density = 31.00F,
            bubbleStyle = BubbleStyle(solidColor = color2),
            selectionHighlightPoint = SelectionHighlightPoint(Color.Black),
            selectionHighlightPopUp = SelectionHighlightPopUp(Color.Cyan)
        ),
```

Luodaan xAxisData arvo, jossa konfiguroidaan x -akselin parametreja.

```
val xAxisData = AxisData.Builder()
    .axisStepSize(9.dp)
    .steps(xAxisSteps)
    .labelData { i ->
        val xMin = -4f
        val xMax = 36f
        val xScale = (xMax - xMin) / xAxisSteps
        ((i+xScale) + xMin).formatToSinglePrecision()

    }
    .startDrawPadding(10.dp)
    .build()
```

Luodaan yAxisData -arvo, jossa konfiguroidaan y -akselin parametreja

```
val yAxisData = AxisData.Builder()
    .steps(yAxisSteps)
    .labelAndAxisLinePadding(20.dp)
    .labelData { i ->
        val yMin = 0f
        val yMax = 120f
        val yScale = (yMax - yMin) / yAxisSteps
        ((i*yScale) + yMin).formatToSinglePrecision()
    }.build()
```

Luodaan data -arvo, johon lisätään kuplien lista, sekä muut vaaditut konfiguraatiot.

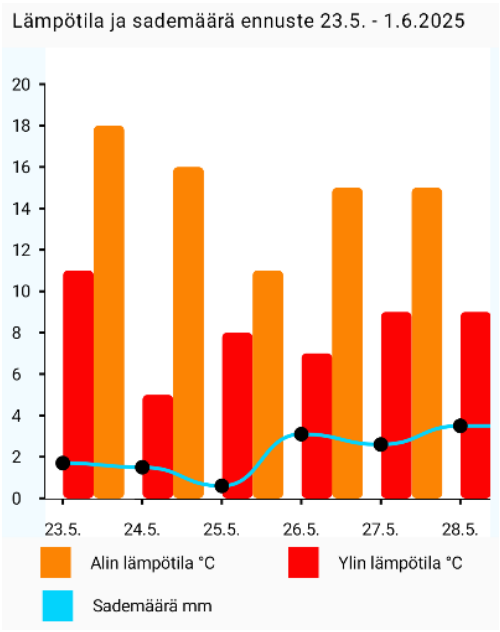
```
val data = BubbleChartData(
    bubbles = dataList,
    isZoomAllowed = true,
    xAxisData = xAxisData,
    yAxisData = yAxisData,
    gridLines = GridLines()
)
```

Kutsutaan funktiota, joka piirtää kaavion. Syötetään funktiolle data -arvo, joka sisältää datan sekä kaikki konfiguraatiot.

```
BubbleChart(
    modifier = Modifier
        .fillMaxWidth()
        .padding(10.dp)
        .height(400.dp),
    bubbleChartData = data
)
```

2.7 Yhdistetty kaavio – CombinedChartAppV1

Yhdistetyt kaaviot ovat hyviä, kun halutaan vertailla dataa, joilla on eri yksiköt kuten esimerkissä, lämpötila ja sademäärä. Yhdistetyt kaaviot ovat myös hyviä, kun useamman eri datan suhdetta halutaan vertailla. Esimerkissä voidaan verrata ennustettua sateen määrää suhteessa ennustettuun lämpötilaan.



Luodaan dynaaminen lista viivadatalle. Lisätään listaan pisteitä.

```
val lineData = arrayListOf(
        Point(1f, 1.9f),
        Point(2f, 1.7f),
        Point(3f, 0.8f),
        Point(4f, 3.3f),
        Point(5f, 2.8f),
        Point(6f, 3.7f),
        Point(7f, 3.7f),
        Point(8f, 9.0f),
        Point(9f, 7.2f),
        Point(10f, 0.2f)
    )
```

Luodaan dynaaminen lista pylväille. Lisätään jokaiseen pylväsryhmään kaksi pylvästä. Yksi pylväs alimmalle lämpötilalle ja toinen ylimmälle.

```
val groupBarData = arrayListOf(
    GroupBar(label = "23.5.", barList = arrayListOf(
        BarData(point = Point(1F, 11f)),
        BarData(point = Point(2F, 18f))
    )),

    GroupBar(label = "24.5.", barList = arrayListOf(
        BarData(point = Point(1F, 5f)),
        BarData(point = Point(2F, 16f))
    )),

    GroupBar(label = "25.5.", barList = arrayListOf(
        BarData(point = Point(1F, 8f)),
        BarData(point = Point(2F, 11f))
    )),
```

Luodaan xAxisData ja yAxisData arvot, jossa määritellään akselien eri parametreja. Esimerkissä näistä käytetään vain osaa, lisää muutettavia parametreja löydät mm. pitämällä hiiren esimerkiksi AxisData kohdalla ja painamalla esiin ilmestyvän ikkunan oikeasta alareunasta kynän kuvaketta.

```
val xAxisData = AxisData.Builder()
    .axisStepSize(20.dp)
    .bottomPadding(5.dp)
    .labelData { index -> index.toString() }
    .build()

val yAxisData = AxisData.Builder()
    .steps(yStepSize)
    .labelAndAxisLinePadding(20.dp)
    .labelData { index -> (index * (maxRange / yStepSize)).toString() }
    .build()
```

Luodaan linePlotData -arvo, johon lisätään mm. viivakaavion data ja viivalle haluttu väri.

```
val linePlotData = LinePlotData(
    lines = listOf(
        Line(
            dataPoints = lineData,
            lineStyle = LineStyle(color = color3),
            intersectionPoint = IntersectionPoint(),
            selectionHighlightPoint = SelectionHighlightPoint(),
            selectionHighlightPopUp = SelectionHighlightPopUp()
        )
    )
)
```

Luodaan arvo, johon lisätään lista haluttuja kuvatekstejä ja niiden värit.

```
val legendsConfig = LegendsConfig(
    legendLabelList = arrayListOf(
        LegendLabel(
            color = color1,
            name = "Alin lämpötila °C"
        ),
        LegendLabel(
            color = color2,
            name = "Ylin lämpötila °C"
        ),
        LegendLabel(
            color = color3,
            name = "Sademäärä mm"
        )
    ),
    gridColumnCount = 2
)
```

Luodaan barPlotData arvo, johon lisätään pylväiden data ja värilista.

```
val colorPaletteList = listOf(color1, color2,)

val barPlotData = BarPlotData(
    groupBarList = groupBarData,
    barStyle = BarStyle(barWidth = 35.dp),
    barColorPaletteList = colorPaletteList
)

val combinedChartData = CombinedChartData(
    combinedPlotDataList = listOf(barPlotData, linePlotData),
    xAxisData = xAxisData,
    yAxisData = yAxisData
)
```

Lisätään sarake, jonka sisällä kutsutaan kaavion piirtävää funktiota. Syötetään funktiolle combinedChartData arvo, joka sisältää kaavioiden arvot ja konfiguroinnit. Yhdistetyn kaavion alla kutsutaan funktiota, joka piirtää kuvatekstit. Syötetään sille kuvatekstien konfiguraatiot.

```
Column(
    Modifier
        .height(500.dp)
) {
    CombinedChart(
        modifier = Modifier
            .height(400.dp),
        combinedChartData = combinedChartData
    )
    Legends(
        legendsConfig = legendsConfig
    )
}
```

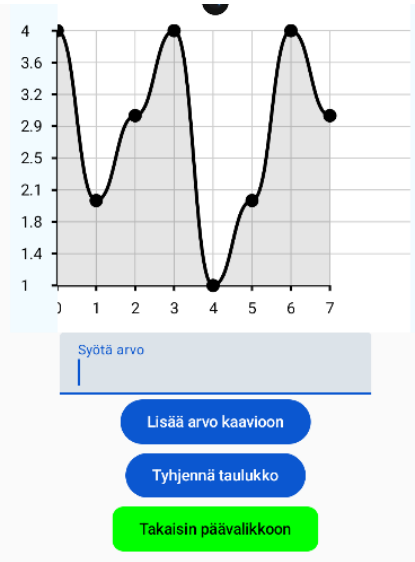
3 GRAAFISET LASKIMET JA KÄYTTÄJÄN SYÖTTÄMÄT ARVOT

Tässä luvussa aletaan hyödyntää MathParser kirjastoa erilaisten laskutoimitusten suorittamiseen. Esimerkeissä hyödynnettään myös luvun 2 kuvaajia. Kaikki luvun esimerkit löytyvät Github repositorystä **GraphingCalculators** -sovelluksesta suomeksi kommentoituna.

Github: https://github.com/TomiLakkakorpi/KotlinChartApps

3.1 UserInputExample1 – Käyttäjän syöttämä data

Tässä esimerkissä käydään läpi, miten käyttäjän syöttämää dataa voidaan syöttää kaavioille. Esimerkissä käytetään viivakaaviota, mutta dataa voi yhtälailla syöttää muihinkin kaavioihin samalla tavalla. Tässä esimerkissä ei käydä läpi, miten kaavioita luodaan. Jos haluat tutustua kaavioiden tekemiseen, palaa aiempiin esimerkkeihin.



Luodaan text muuttuja joka on tyyppiä mutableState.

```
var text by remember {
    mutableStateOf("")
}
```

Tarkistetaan onko lista tyhjä ennen kaavion piirtämistä. Jos kaavio sisältää dataa, kutsutaan kaavion piirtävää funktiota.

```
if(lineChartList.isNotEmpty()){
    LineChart(
        modifier = Modifier
            .fillMaxWidth()
            .height(300.dp),
        lineChartData = data
    )
}
```

Luodaan tekstikenttä arvon syöttämiseen, jonka sisältämää tekstiä tarkastellaan. Kun tekstikentän teksti muuttuu, asetetaan se text muuttujan arvoksi.

```
TextField(
    value = text,
    onValueChange = { newText ->
        text = newText
    },
    label = {
        Text(text = "Syötä arvo")
    },
)
```

Lisätään näppäin. Kun näppäintä painetaan, tarkistetaan ensin onko text muuttujan arvo tyhjä. Jos text muuttuja sisältää jotain merkkejä, tarkistetaan onko annettu teksti hyväksyttävä, eli saadaanko se muutettua float muotoon. Jos syötetty teksti hyväksytään, lisätään syötetty arvo kaavioon. Kun arvo on syötetty listaan, palautetaan text muuttujan arvo tyhjäksi ja nostetaan index arvoa yhdellä.

```
Button(
    onClick = {
        if(text.isNotEmpty()){
            if(checkIfValidValue(text)) {
                lineChartList.add(Point(lineChartListIndex, text.toFloat(), ""))
                text = ""
                lineChartListIndex++
            } else {
                Toast.makeText(
                    context,
                    "Syöttämäsi arvoa ei voida hyväksyä! Syötä arvo muodossa 1.1",
                    Toast.LENGTH_SHORT).show()
            }
        } else {
            Toast.makeText(context, "Syötä arvo!", Toast.LENGTH_SHORT).show()
        }
    }
) {
    Text("Lisää arvo kaavioon")
}
```

Lisätään myös näppäin jolla kaavion data voidaan tyhjentää. Kun näppäintä painetaan, varmistetaan ettei lista ole jo tyhjä. Jos lista sisältää dataa, poistetaan listan viimeinen arvo niin kauan kuin lista sisältää dataa. Kun lista ei ole enää tyhjä, palautetaan index arvo takaisin nollaan. Muutetaan text muuttujan arvoa väliaikaisesti, jolla saadaan aiheutettua käyttöliittymän uudelleen kokoonpano (recomposition).

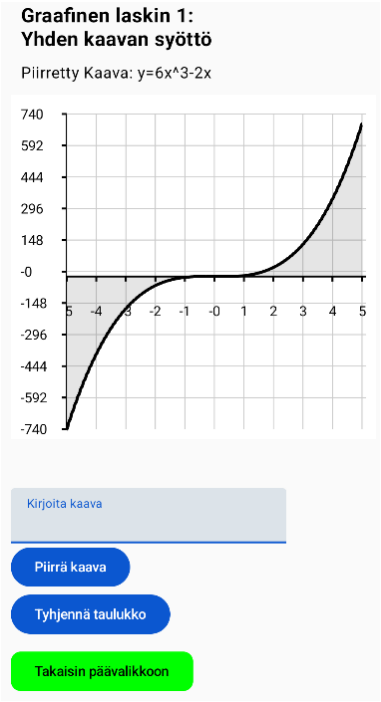
```
Button(
    onClick = {
        if (lineChartList.isNotEmpty()) {
            while(lineChartList.isNotEmpty()) {
                lineChartList.removeAt(lineChartList.size -1)
            }

            lineChartListIndex = 0f

            text = " "
            text = ""
        } else {
            Toast.makeText(context, "Taulukko on jo tyhjä!", Toast.LENGTH_SHORT).show()
        }
    }
) {
    Text("Tyhjennä taulukko")
}
```

Funktio, jolla tarkistetaan, onko syötetty arvo hyväksyttävä. Funktiolle syötetään text muuttujan arvo, ja funktio palauttaa tosi tai epätosi riippuen onko arvo hyväksytty.

```
fun checkIfValidValue(input: String): Boolean {
    val regex = Regex("[+-]?([0-9]+([.][0-9]*)?|[.][0-9]+)")
    return input.matches(regex)
}
```

3.2 Graafinen laskin 1 – Yksi kaava

Nyt kun osaamme piirtää kuvaajia ja syöttää niille dataa sen sijaan että piirretty data olisi kovakoodattu, lähdetään toteuttamaan graafisia laskimia. Ideana on, että käyttäjä syöttää ohjelmaan kaavan, esimerkiksi y=4x²-3x+5, jolla lasketaan pisteet käyrällä. Kun pisteet on laskettu, piirretään käyrä niiden avulla kuvaajaan. Näihin laskutoimituksiin tarvitaan matemaattisiin laskuihin suunnattu kirjasto, MathParser. Kirjaston käyttöönoton ohjeet löytyvät luvusta 1.2.1.



Lisätään tekstikenttä, jonka arvo lisätään text -muuttujaan. Myöhemmin kun kaava piirretään, text -arvo lisätään muuttujaan formula.

```
TextField(
    value = text,
    onValueChange = { newText ->
        text = newText
    },
    label = {
        Text(text = "Kirjoita kaava")
    },
)
```

Aletaan laskea y:n arvoja joka aloitetaan x:n arvosta -5x. Y:n arvot lasketaan syöttämällä x:n arvo kaavaan. Kun molemmat y:n ja x:n arvot ovat selvillä, lisätään piste listaan. Tämän jälkeen lisätään x:n arvoa 0.1x verran ja lasketaan y:n arvo uudestaan, tällä kertaa -4.9x arvolla. Tätä jatketaan niin kauan, kunnes x arvo saavuttaa arvon 5x. jolloin olemme laskeneet y:n arvot välille -5x – 5x.

```
while(xValue <= 5) {
    x = Argument("x=$xValue")
    y = Argument(text, x)
    e = Expression("y", y)
    yValue = e.calculate().toFloat()
    calculator1lineChartList.add(Point(xValue, yValue, ""))
    xValue = xValue + 0.1f
}
chart1Drawn = true
```

Kun lista halutaan tyhjentää, tehdään tämä poistamalla listan viimeinen arvo niin kauan kunnes lista on tyhjä.

```
while(calculator1lineChartList.isNotEmpty()) {
    calculator1lineChartList.removeAt(calculator1lineChartList.size -1)
}
```

3.3 Graafinen laskin 2 – Yksi kaava, muokattava piirtoalue

Edelliseen esimerkkiin verrattuna, tässä esimerkissä on muokattava piirtoalue ja piirtotiheys. Aiemmassa esimerkissä vakio piirtoalue oli -5x – 5x ja laskutiheys 0.1x. Kun käyttäjälle annetaan tämä mahdollisuus muokata piirtoaluetta, hän voi valita mitä aluetta kaavasta hän haluaa tutkia tarkemmin.

Luodaan tekstikenttä, johon käyttäjä voi syöttää lähtöarvon laskemiselle. Tämän lisäksi luodaan tekstikentät loppuarvolle ja laskutiheydelle. Nämä arvot asetetaan xStart, xEnd ja xIncrement muuttujiin.

```
TextField(
    modifier = Modifier
        .padding(5.dp)
        .width(150.dp),
    value = xStart.toString(),
    onValueChange = { newText ->
        xStart = newText.toFloat()
    },
    label = {
        Text(text = "Syötä x Lähtöarvo")
    },
)
```

Asetetaan xValue -muuttujan arvoksi käyttäjän asettama xStart arvo.

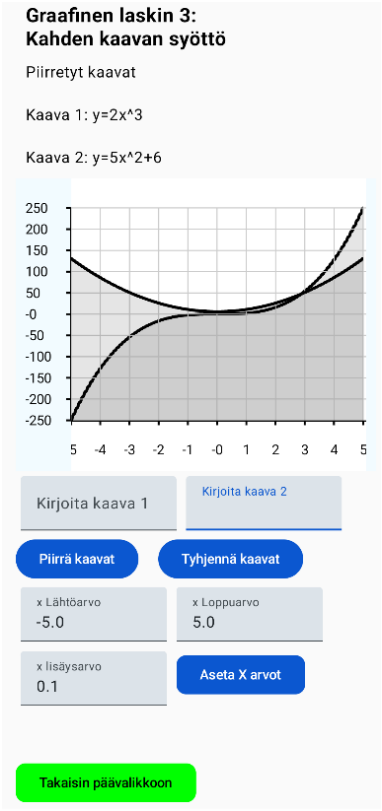
```
onClick = {
    xValue = xStart
    Toast.makeText(context, "X arvot asetettu!", Toast.LENGTH_SHORT).show()
}
```

Lasketaan y:n arvot käyttäjän määrittelemälle välille xStart – xEnd. Jokaisen laskun jälkeen xValue -muuttujaan lisätään xIncrement arvo ja lasku tehdään uudelleen.

```
while(xValue <= xEnd) {
    x = Argument("x=$xValue")
    y = Argument(formula, x)
    e = Expression("y", y)
    yValue = e.calculate().toFloat()
    Calculator2lineChartList.add(Point(xValue, yValue, ""))
    xValue = xValue + xIncrement
}
chart1Drawn = true
```

3.4 Graafinen laskin 3 – Kaksi kaavaa

Tässä esimerkissä on lisätty mahdollisuus piirtää kaksi kaavaa samanaikaisesti. Kun useampi kaava piirretään samalle kuvaajalle, niiden suhdetta toisiinsa voidaan tutkia helpommin kuin jos ne olisivat erillisissä kuvaajissa.



Lisätään data -arvoon lines -listaan toinen viiva ja asetetaan sen datapisteiksi toinen lista

```
lines = listOf(
    Line(
        dataPoints = Calculator3lineChartList1,
        selectionHighlightPoint = SelectionHighlightPoint(),
        shadowUnderLine = ShadowUnderLine(),
        selectionHighlightPopUp = SelectionHighlightPopUp()
    ),
    Line(
        dataPoints = Calculator3lineChartList2,
        selectionHighlightPoint = SelectionHighlightPoint(),
        shadowUnderLine = ShadowUnderLine(),
        selectionHighlightPopUp = SelectionHighlightPopUp()

    ),
)
```

Laskutoimitukset ovat samat kuin aiemmissa esimerkeissä, nyt laskut tehdään vaan kahdelle erilliselle kaavalle:

```
CoroutineScope(IO).launch {
    while(xValue1 <= xEnd) {
        x1 = Argument("x=$xValue1")
        y1 = Argument(formula1, x1)
        e1 = Expression("y", y1)
        yValue1 = e1.calculate().toFloat()
        Calculator3lineChartList1.add(Point(xValue1, yValue1, ""))
        xValue1 = xValue1 + xIncrement
    }
    text1 = ""
    chart1Drawn = true
}

CoroutineScope(IO).launch{
    while(xValue2 <= xEnd) {
        x2 = Argument("x=$xValue2")
        y2 = Argument(formula2, x2)
        e2 = Expression("y", y2)
        yValue2 = e2.calculate().toFloat()
        Calculator3lineChartList2.add(Point(xValue2, yValue2, ""))
        xValue2 = xValue2 + xIncrement
    }
    text2 = ""
    chart2Drawn = true
}
```

Listojen tyhjennys tehdään myös molemmille listoille

```
while(Calculator3lineChartList1.isNotEmpty()) {
    Calculator3lineChartList1.removeAt(Calculator3lineChartList1.size -1)
}

while(Calculator3lineChartList2.isNotEmpty()) {
    Calculator3lineChartList2.removeAt(Calculator3lineChartList2.size -1)
}
```

3.5 Graafinen laskin 4 – Ympyrän piirto

Tässä esimerkissä siirrytään piirtämään ympyrä. Ympyrän piirto on astetta monimutkaisempaa ja vaatii useampaa kaavaa. Ympyrän piirtoon käytetään kaavaa: **$(x-h)^2 + (y-k)^2 = r^2$**, jossa (h,k) edustaa ympyrän keskipistettä ja r ympyrän sädettä. Ympyrän keskipisteet saadaan selvittämällä h ja k -arvojen vastaluvut ja ympyrän säde puolestaan laskemalla r^2 neliöjuuri. Näin saamme selville ympyrän keskipisteen, säteen ja neljä ympyrän pistettä. Neljä pistettä ei kuitenkaan riitä ympyrän piirtämiseen YChartsilla, joten meidän täytyy laskea lisää pisteitä ympyrästä. Tämä tapahtuu kaavoilla **x = h + r * cos(t)**, jossa t:n arvo on välillä 0–2 radiaania ja **y = k + r * sin(t)**, jossa t:n arvo on välillä 0–2 radiaania. Esimerkissä kaava on valmiiksi syötetty ohjelmaan, ja käyttäjän tarvitsee vain syöttää puuttuvat arvot niille osoitettuihin tekstikenttiin.

Selvitetään ympyrän keskipiste laskemalla h ja k -arvojen vastaluvut kertomalla ne -1:llä. Ympyrän säde selvitetään laskemalla sen neliöjuuri.

```
k = kText.toFloat() * -1.0f
h = hText.toFloat() * -1.0f
r = floatSquareRoot(rText.toFloat())
```

Määritellään ympyrän pisteiden laskutiheys ympyrän säteen mukaan. (isompi ympyrä → pisteet lasketaan harvemmin, pienempi ympyrä → pisteet lasketaan tiheämmin)

```
tIncrement = if(r >= 100) {
    0.1f
} else if(r >= 50) {
    0.05f
} else {
    0.01f
}
```

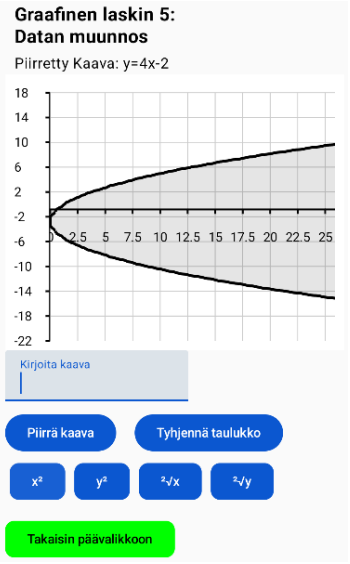
Aiemmissa esimerkeissä laskimme y:n arvoja tietyllä x:n alueella. Tässä esimerkissä lasketaan x:n ja y:n arvoja ennalta määritetyllä alueella (t) joka on välillä 0–2 radiaania. Laskeminen aloitetaan arvosta 0 radiaania, jolla lasketaan x ja y arvot. Kun arvot on laskettu, ne lisätään listaan. Laskun jälkeen t arvoon lisätään tIncrement verran ja laskut tehdään uudelleen, niin kauan kunnes kaikki arvot on laskettu välillä 0–2 radiaania.

```
while (t < 2) {
    var xFormula = Argument("x=$h+$r*cos($t π)")
    e1 = Expression("x", xFormula)

    xValue = e1.calculate().toFloat()

    var yFormula = Argument("y=$k+$r*sin($t π)")
    e2 = Expression("y", yFormula)

    yValue = e2.calculate().toFloat()

    Log.d("CircleTest", "Point added: ($xValue, $yValue) with t value = $t")

    Calculator4lineChartList.add(Point(xValue, yValue))

    t = "%.2f".format(t + tIncrement).toFloat()
}
```

Listojen tyhjentäminen tapahtuu samalla tavalla kuin aiemmissa esimerkeissä.

3.6 Graafinen laskin 5 – Datan muunnokset

Tässä esimerkissä käsitellään miten piirretyn kaavan dataa voidaan muuntaa, esimerkiksi kertomalla x:n arvot toiseen potenssiin. Näiden ominaisuuksien avulla kaavoja voidaan tutkia lisää ja selvittää miten kaava muuttuu, kun esimerkiksi x:n arvot kerrotaan toiseen potenssiin.



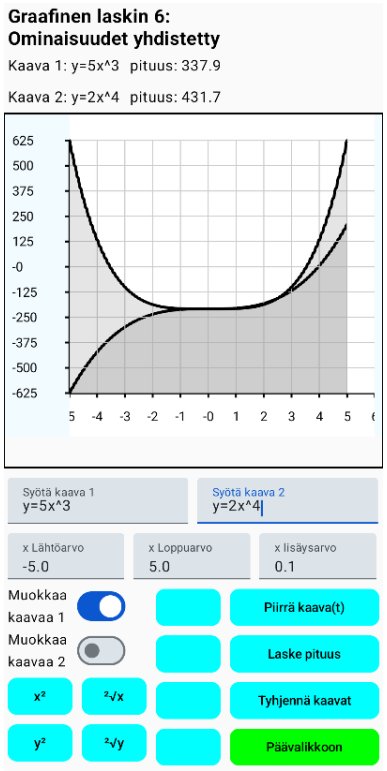
Aloitetaan hakemalla listasta x:n ja y:n arvot index arvon paikasta. Index arvo alkaa nollasta, joten listan ensimmäinen piste haetaan ensimmäisenä. Y:n arvoja ei muokata tässä, joten asetetaan y:n arvoksi listasta haettu y:n arvo. X:n arvoa puolestaan haluamme muuntaa, joten kun arvo on haettu listasta, korotetaan haettu x:n arvo toiseen potenssiin kutsumalla floatSquared funktiota. Tämän jälkeen päivitetään listaa, lisäämällä listan samalle paikalle alkuperäinen y:n arvo, sekä muunnettu x:n arvo. Viimeisenä lisätään index arvoa, jotta samat toimenpiteet voidaan tehdä seuraavalle listan pisteelle. Tätä toistetaan, kunnes kaikki listan pisteet on päivitetty.

```
while(Calculator5squareXIndex < Calculator5lineChartList.size) {
    val y = Calculator5lineChartList[Calculator5squareXIndex].y
    val x = floatSquared(Calculator5lineChartList[Calculator5squareXIndex].x)
    Calculator5lineChartList[Calculator5squareXIndex] = Point(x,y)
    Calculator5squareXIndex++
}
```

3.7 Graafinen laskin 6 – Ominaisuudet yhdistetty

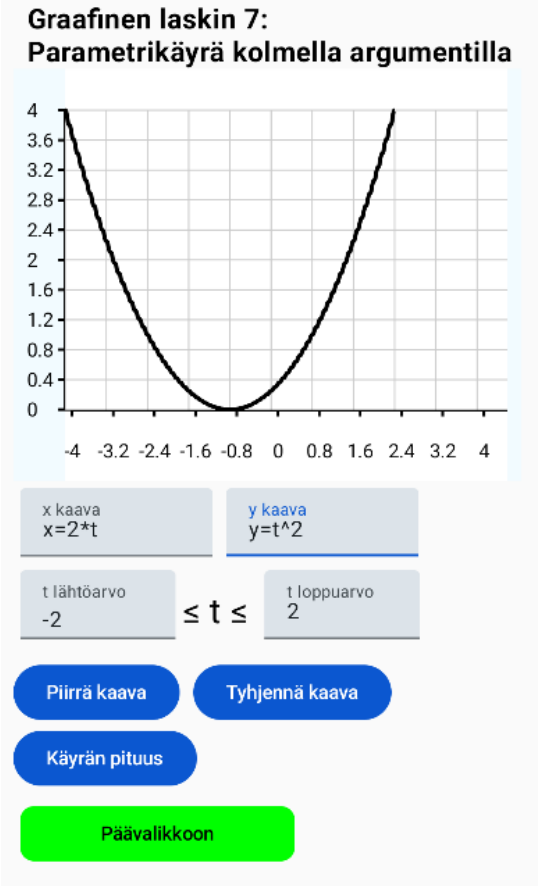
Tähän esimerkkiin on kerätty aiemmista esimerkeistä ominaisuudet ja lisätty muutama uusi asia. Ulkonäköä on myös päivitetty. Yksi uusi lisäys aiempiin esimerkkeihin on kuitenkin se, että aiemmissa esimerkeissä jos kaava oli jo piirretty, lista piti ensin tyhjentää ja sen jälkeen piirtää uudelleen. Tässä versiossa, jos ”piirrä kaava” näppäintä painetaan uudelleen, ohjelma tarkistaa oletko piirtämässä samaa kaavaa kuin mikä on jo piirretty. Jos olet, kaavaa ei piirretä uudelleen, mutta jos kaava on eri, ohjelma ensin tyhjentää listan ja piirtää heti perään kuvaajan uudella kaavalla.

Esimerkissä on neljä tyhjää näppäintä valmiina jatkokehitystä varten.



3.8 Graafinen laskin 7 – Parametrinen käyrä

Tässä esimerkissä käsitellään käyrän piirtämistä kolmella argumentilla (x, y & t). Esimerkissä molemmille x ja y -arvoille on omat kaavansa, jotka käyttäjä syöttää. Tämän jälkeen käyttäjä valitsee piirtoalueen, joka on vakiona -2t – 2t. Kun arvot on syötetty, käyttäjä voi piirtää kaavan painamalla ”piirrä kaava” näppäintä. Käyrä piirtyy, kun kaikki käyrän pisteet on laskettu asetetulle piirtoalueelle. Käyrän pituus voidaan myös laskea ”käyrän pituus” näppäintä painamalla, jolloin käyrän pituus näytetään näppäimen oikealla puolella. Käyttäjä voi myös voi myös poistaa käyrän ”tyhjennä kaava” näppäimellä, jonka jälkeen uusi käyrä voidaan piirtää uusilla kaavoilla.



Asetetaan tValue -arvoksi tStart arvo, jotta laskut aloitetaan oikeasta t:n arvosta. X:n ja y:n arvoja lasketaan niin kauan, kunnes tValue arvo saavuttaa tEnd arvon.

```
tValue = tStart.toFloat()
while(tValue <= tEnd.toFloat()) {
```

Päivitetään käyttäjän syöttämää kaavaa. Etsitään kaavasta ”t” ja sen tilalle asetetaan tValue arvo. Kun kaava on päivitetty, luodaan siitä argumentti ja lasketaan x:n arvo. Asetetaan laskettu x:n arvo xValue -muuttujaan.

```
var updatedXFormula = xFormula.replace("t", tValue.toString())
var xArgument = Argument(updatedXFormula)
ex = Expression("x", xArgument)
xValue = ex.calculate().toFloat()
```

Päivitetään käyttäjän syöttämää kaavaa. Etsitään kaavasta ”t” ja sen tilalle asetetaan tValue arvo. Kun kaava on päivitetty, luodaan siitä argumentti ja lasketaan y:n arvo. Asetetaan laskettu y:n arvo yValue -muuttujaan.

```
var updatedYFormula = yFormula.replace("t", tValue.toString())
var yArgument = Argument(updatedYFormula)
ey = Expression("y", yArgument)
yValue = ey.calculate().toFloat()
```

Kun x ja y -arvot on laskettu t:n arvolle, lisätään piste listan x:n ja y:n arvoilla. Lisätään t:n arvoa seuraavaa laskua varten ja pyöristetään tValue arvo.

```
Calculator7lineChartList.add(Point(xValue, yValue))
tValue = "%.2f".format(tValue + tIncrement).toFloat()
```

3.9 Graafinen laskin 8 – PNS Suora

Tässä esimerkissä käsitellään PNS -suoran laskemista (pienimmän neliösumman suora). PNS -suora lasketaan kaavalla y = kx + b. Kaavassa k on suoran kulmakerroin, ja b on vakio joka kertoo suoran ja y-akselin leikkauskohdan. Tarvittavat arvot saadaan selville alla olevilla kaavoilla:

*k = (n * Σ(xi * yi) - (Σxi) * (Σyi)) / (n * Σ(xi^2) - (Σxi)^2)*
*b = (Σyi - k * Σxi) / n*

missä,

- *n on datapisteiden määrä*
- *Σxi on kaikkien pisteiden x-arvojen summa*
- *Σyi on kaikkien pisteiden y-arvojen summa*
- *Σ(xi * yi) on kaikkien pisteiden x*y tulojen summa*
- *Σ(xi^2) on kaikkien x-arvojen neliöiden-summa*

Ensin lasketaan kulmakerroin (k) ja vakio (b). Kun arvot on laskettu, voidaan arvot sovittaa kaavaan y= kx + b. Y:n arvot lasketaan samalle x:n välille kuin käyrä (esimerkissä 1x -12x)

Asetetaan xStart arvoksi datalistan ensimmäisen pisteen x:n arvo. Asetetaan xEnd arvoksi datalistan viimeisen pisteen x:n arvo. Asetetaan n -arvoksi datalistan koko.

```
val xStart = calculator8lineChartList[0].x
val xEnd = calculator8lineChartList[calculator8lineChartList.size -1].x
n = calculator8lineChartList.size
```

Lasketaan listan x arvojen summat ja lisätään ne xSum muuttujaan. Pyöristetään arvot kahteen desimaaliin.

```
while(xSumIndex <= calculator8lineChartList.size -1) {
    var x = calculator8lineChartList[xSumIndex].x
    xSum = "%.2f".format(xSum + x).toFloat()
    xSumIndex++
}
```

Lasketaan listan y arvojen summat ja lisätään ne ySum muuttujaan. Pyöristetään arvot kahteen desimaaliin.

```
while(ySumIndex <= calculator8lineChartList.size -1) {
    var y = calculator8lineChartList[ySumIndex].y
    ySum = "%.2f".format(ySum + y).toFloat()
    ySumIndex++
}
```

Lasketaan kaikkien pisteiden x*y tulo. Lisätään pisteiden tulot xySum muuttujaan. Pyöristetään arvot kahteen desimaaliin.

```
while(xySumIndex <= calculator8lineChartList.size -1) {
    var xy = calculator8lineChartList[xySumIndex].x * calculator8lineChartList[xySumIndex].y
    xySum = "%.2f".format(xySum + xy).toFloat()
    xySumIndex++
}
```

Lasketaan kaikkien x:n arvojen neliö. Lisätään arvot xSquaredSum muuttujaan. Pyöristetään arvot kahteen desimaaliin.

```
while(xSquaredSumIndex <= calculator8lineChartList.size -1) {
    var x = calculator8lineChartList[xSquaredSumIndex].x
    xSquaredSum = xSquaredSum + (x*x)
    xSquaredSumIndex++
}
```

Kun kaikki vaaditut parametrit on laskettu, voidaan laskea kulmakerroin (k), joka lasketaan kaavalla: *k = (n * Σ(xi * yi) - (Σxi) * (Σyi)) / (n * Σ(xi^2) - (Σxi)^2)*
Syötetään edellisissä kohdissa lasketut arvot kaavaan ja lasketaan kulmakerroin. Asetetaan kulmakerroin muuttujaan k.

```
var kArgument = Argument("k = ($n * $xySum - $xSum * $ySum) / ($n * $xSquaredSum - $xSum^2)")
var kExpression = Expression("k", kArgument)
k = kExpression.calculate().toFloat()
```

Lasketaan myös vakio (b), joka lasketaan kaavalla: *b = (Σyi - k * Σxi) / n*
Syötetään edellisissä kohdissa lasketut arvot kaavaan ja lasketaan vakio (b). Asetetaan vakion arvo muuttujaan b.

```
var bArgument = Argument("b = (($ySum - $k * $xSum) / $n)")
var bExpression = Expression("b", bArgument)
b = bExpression.calculate().toFloat()
```

Kun kulmakerroin (k) ja vakio (b) on laskettu, voidaan laskea suoran pisteet kaavalla y = kx + b. Syötetään kaavaan kulmakerroin, vakio (b) ja x:n arvo. Lasketaan y:n arvot samalle välille kuin käyrä (esimerkissä 1x-12x). Lisätään lasketut arvot PNS -listaan. Kun arvot on laskettu, kuvaaja piirretään jossa näkyy käyrä sekä PNS -suora.

```
xValuePNS = xStart

while(xValuePNS <= xEnd) {
    var yArgument = Argument("y = $k*$xValuePNS +$b")
    var yExpression = Expression("y", yArgument)
    yValuePNS = yExpression.calculate().toFloat()

    calculator8lineChartListPNS.add(Point(xValuePNS, yValuePNS))
    xValuePNS = "%.2f".format(xValuePNS + 0.5f).toFloat()
}
```

4 JATKOKEHITYSIDEOITA

- yMax ja yMin -arvojen pyöristys kuvaajien konfiguraatiossa seuraavaan kymmeneen/sataan/tuhanteen jne.
- Ellipsin piirtäminen
    - Ellipsi piirretään muuten samalla tavalla kuin ympyrä, mutta ellipsin piirto vaatii x ja y arvojen muokkaamista. Ominaisuus siis mahdollistaisi x ja y arvojen muokkaamisen kaavassa **$(x-h)^2 + (y-k)^2 = r^2$.**
- Datanmuunnokset sovellukseen lisää erilaisia muunnosvaihtoehtoja.
- Ominaisuus, jolla kahden kaavan leikkauspisteen/pisteet voi selvittää.

LÄHTEET

CodeAndTheory. 27.6.2025. YCharts. Github. https://github.com/codeandtheory/YCharts

mariusgromada. 27.6.2025. MathParser.org/mXparser Github. https://github.com/mariuszgromada/MathParser.org-mXparser