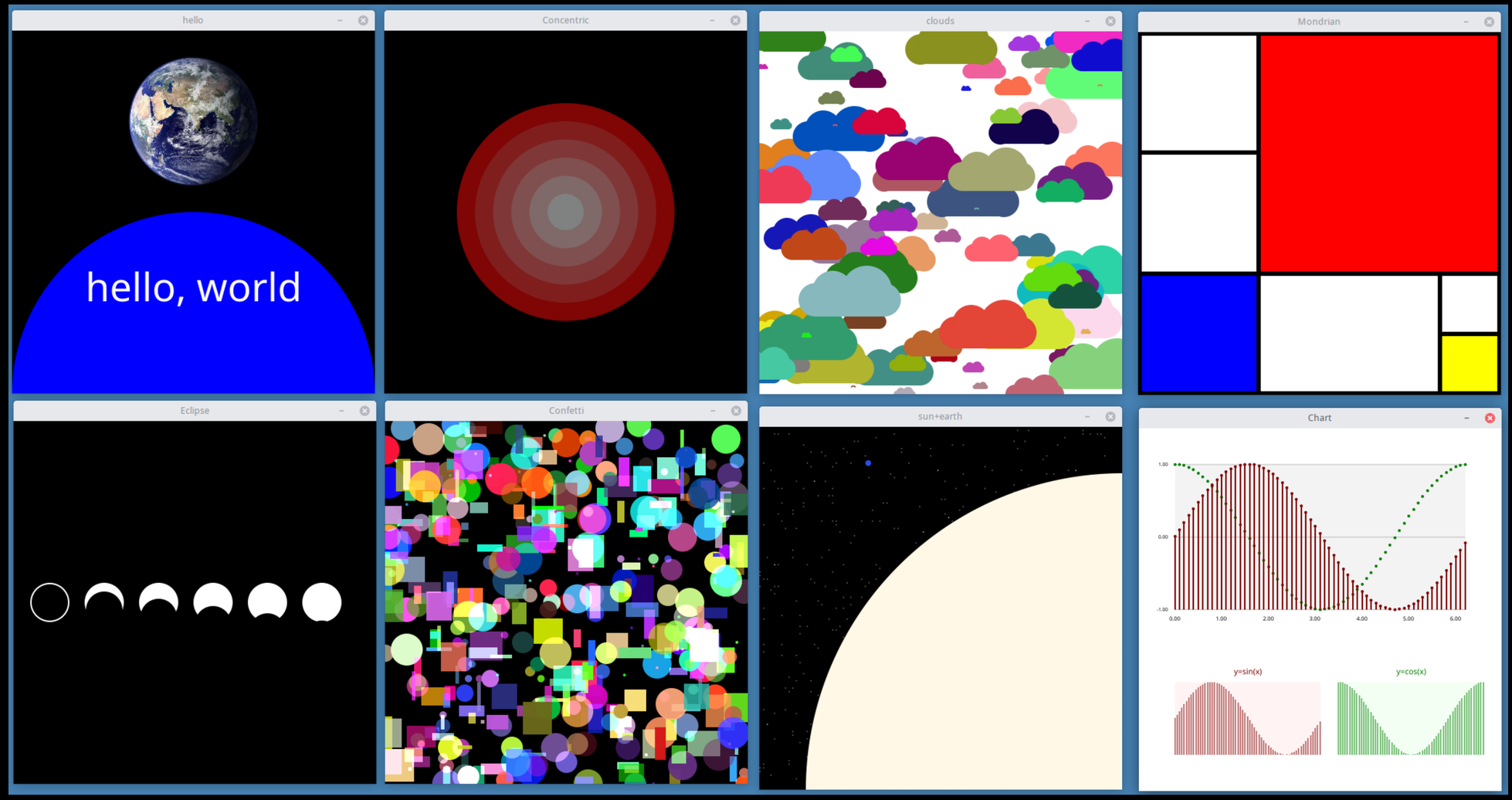
fc a high-level canvas API for the fyne toolkit



Anthony Starks / @ajstarks

Motivation

The desire for a high-level Go API for developers and designers to think in terms of high level objects that make up a visual display. The objects will be familiar to anyone using a modern illustration program (text, images, lines, arcs, circles, curves, etc). The API should facilitate the artful arrangement of these elements on a scalable 2D canvas.

Use Cases: Information Displays, Data Visualization, Creative Coding, Presentations

https://gist.github.com/ajstarks/5bad9b1f5a859b86a17a03bbfbafcee6

Elements

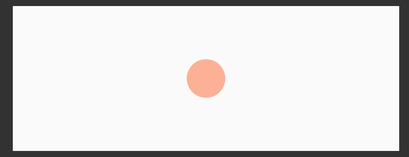
Text

CText

EText

circle

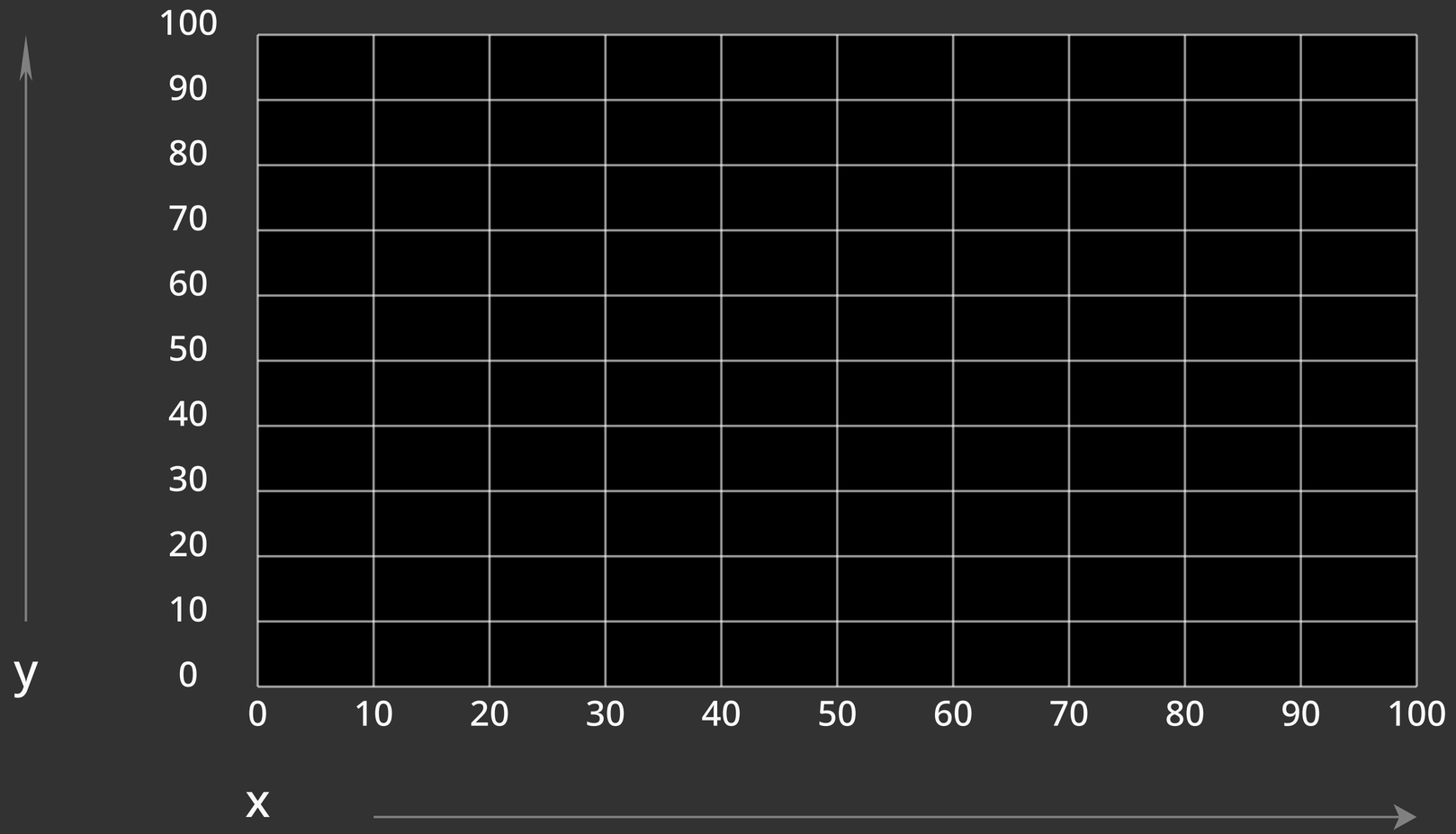
line



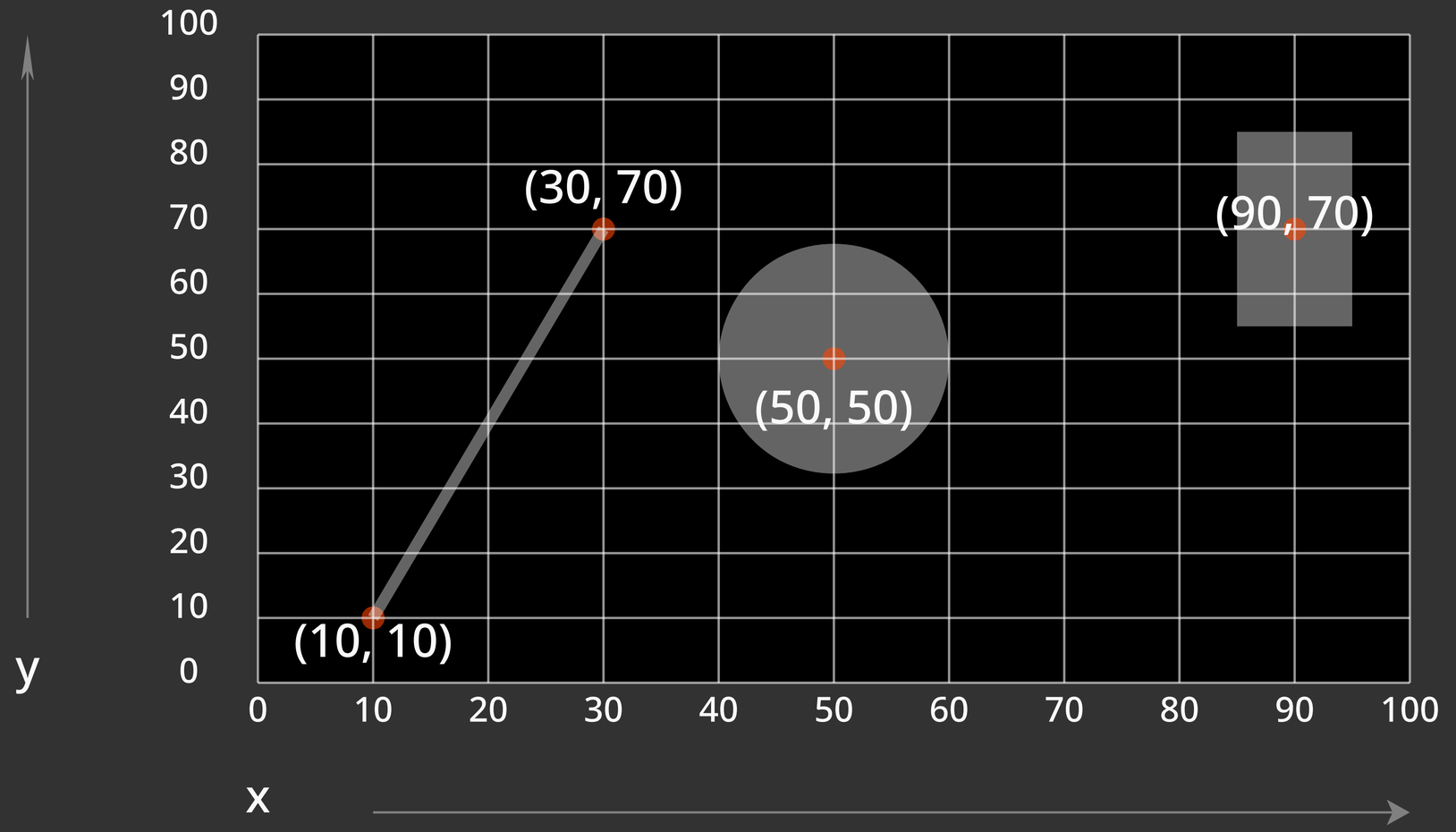
rectangle

image

The Percent Grid



Using the Percent Grid



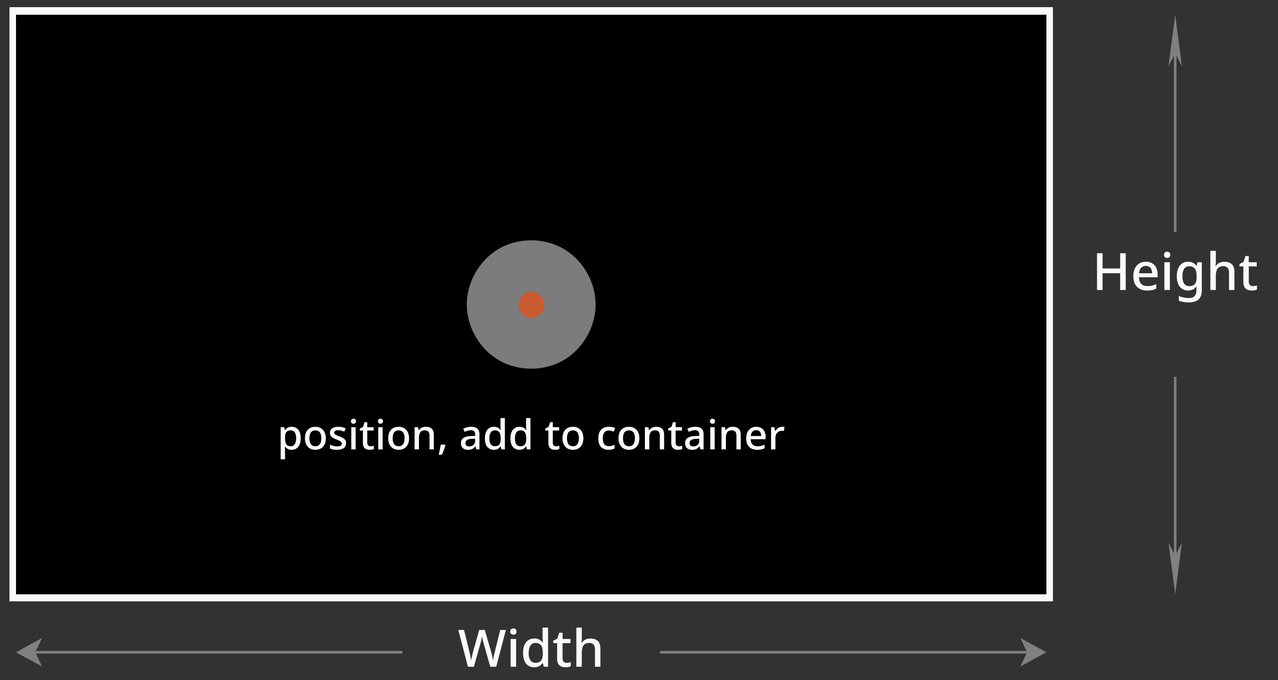
Rect(90, 70, ...)

Circle(50, 50, ...)

Line(10, 10, 30, 70, ...)

fc structure and operation

```
// Canvas is where objects are drawn into
type Canvas struct {
    Window    fyne.Window
    Container *fyne.Container
    Width     float64
    Height    float64
}
```



fc Percentage-based methods on *Canvas

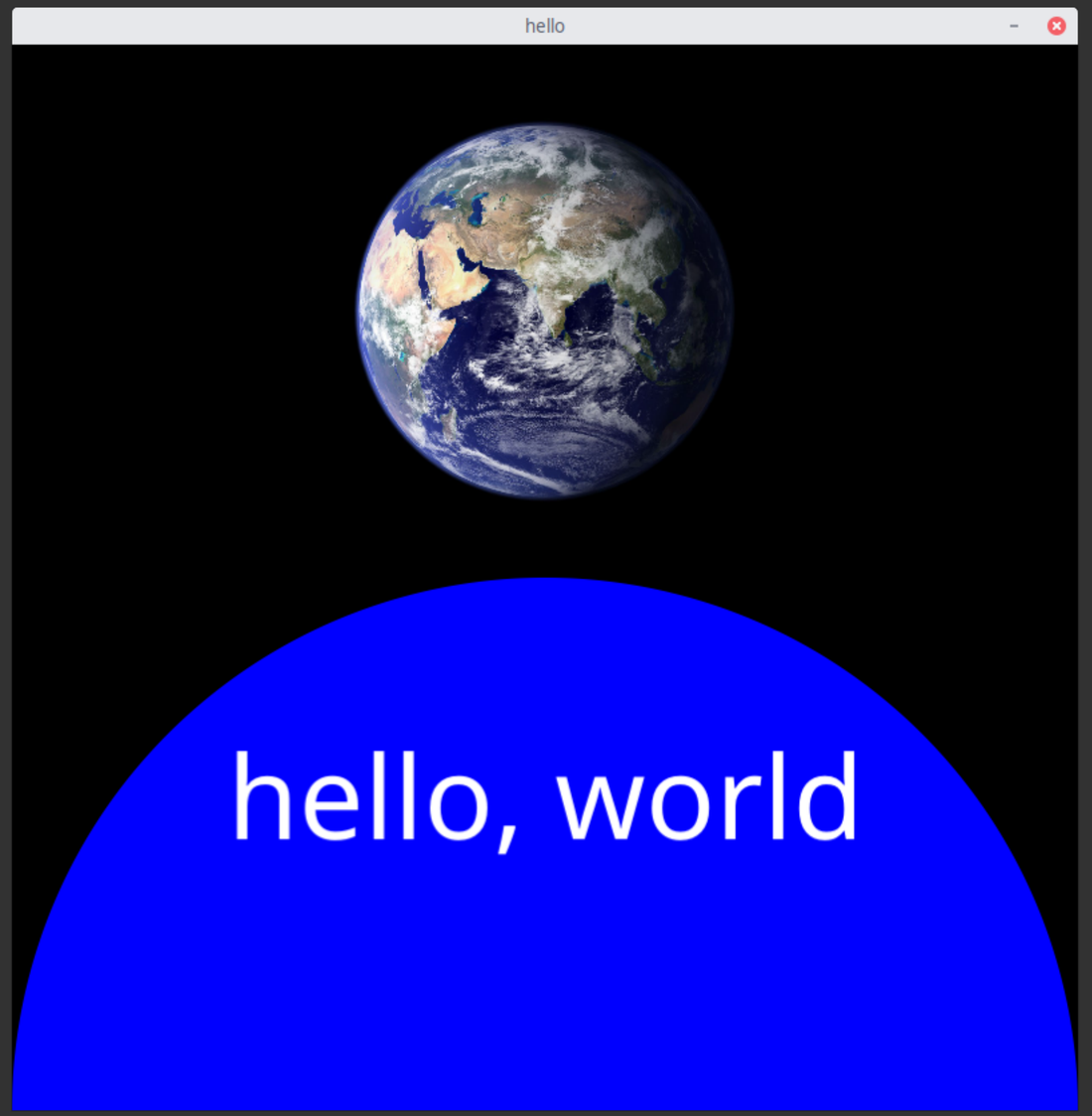
Make a new canvas	`NewCanvas(name string, w, h int) Canvas`
Place text, left-aligned	`Text(x, y, size float64, s string, fill color.RGBA)`
Place centered text	`CText(x, y, size float64, s string, fill color.RGBA)`
Place end-aligned text	`EText(x, y, size float64, s string, fill color.RGBA)`
Obtain the text width	`TextWidth(s string, size float64) float64`
Circle centered (x,y), radius r	`Circle(x, y, r float64, fill color.RGBA)`
Rectangle, upper-left at (x,y)	`CornerRect(x, y, w, h float64, fill color.RGBA)`
Rectangle centered at (x,y)	`Rect(x, y, w, h float64, fill color.RGBA)`
Line from (x1,y) to (x2,y2)	`Line(x1, y1, x2, y2, size float64, stroke color.RGBA)`
Image centered at (x,y)	`Image(x, y float64, w, h int, name string)`
Display and run	`EndRun()`

Convenience methods

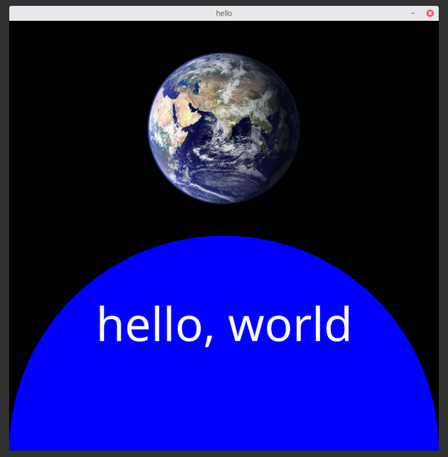
Lookup colors by name	`ColorLookup(s string) color.RGBA`
Map one range into another	`MapRange(value, low1, high1, low2, high2 float64) float64`
Polar to Cartesian	`Polar(x, y, r, angle float64) (float64, float64)`
Convert degrees to radians	`Radians(deg float64) float64`

fc hello, world

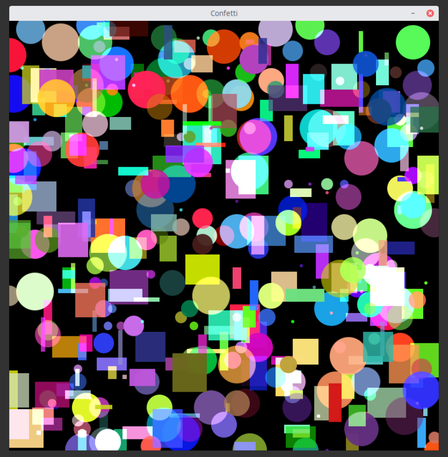
```
package main

import (
    "image/color"

    "github.com/ajstarks/fc"
)

func main() {
    width, height := 500, 500
    blue := color.RGBA{0, 0, 255, 255}
    white := color.RGBA{255, 255, 255, 255}

    canvas := fc.NewCanvas("hello", width, height)

    canvas.Circle(50, 0, 100, blue)
    canvas.CText(50, 25, 10, "hello, world", white)
    canvas.Image(50, 75, 200, 200, "earth.jpg")

    canvas.EndRun()
}
```



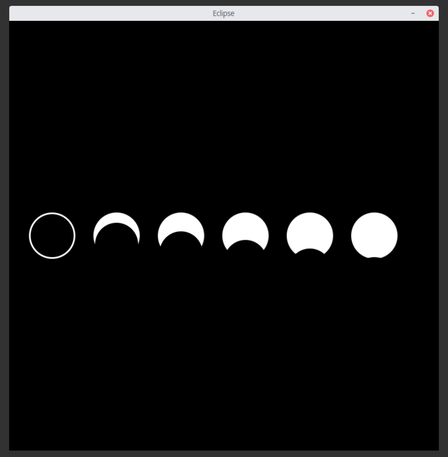
demo/test clients



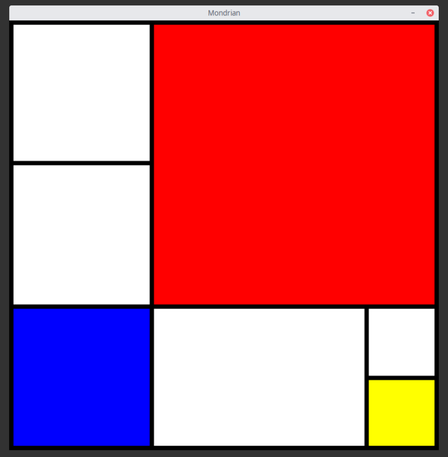
hello



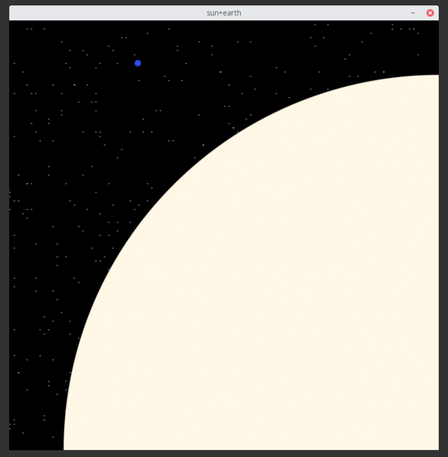
confetti



eclipse



mondrian



sunearth

cloud

fc/chart

Scalable, modular, composable charts

- Bar charts
- Column Charts
- Line charts
- Scatter charts
- Title
- Axes
- Frames

fc/chart: data structures

```
// NameValue is a name,value pair
type NameValue struct {
    label string
    note  string
    value float64
}

// ChartBox holds the essential data for making a chart
type ChartBox struct {
    Title                   string
    Data                    []NameValue
    Color                   color.RGBA
    Top, Bottom, Left, Right float64
    Minvalue, Maxvalue      float64
    Zerobased               bool
}
```

fc/chart methods on *ChartBox

Read data int ChartBox	`func DataRead(r io.Reader) (ChartBox, error)`
Bar Chart	`Bar(c fc.Canvas, size float64)`
Horizontal Bar Chart	`HBar(c fc.Canvas, size, linespacing, textsize float64)`
Line Chart	`Line(c fc.Canvas, size float64)`
Scatter Chart	`Scatter(c fc.Canvas, size float64)`
Centered Title	`CTitle(c fc.Canvas, size, offset float64)`
Chart Frame	`Frame(c fc.Canvas, opacity float64)`
X Axis Label	`Label(c fc.Canvas, size float64, interval int)`
Y axis	`YAxis(c fc.Canvas, size, min, max, step float64, fmt string, grid bool)`

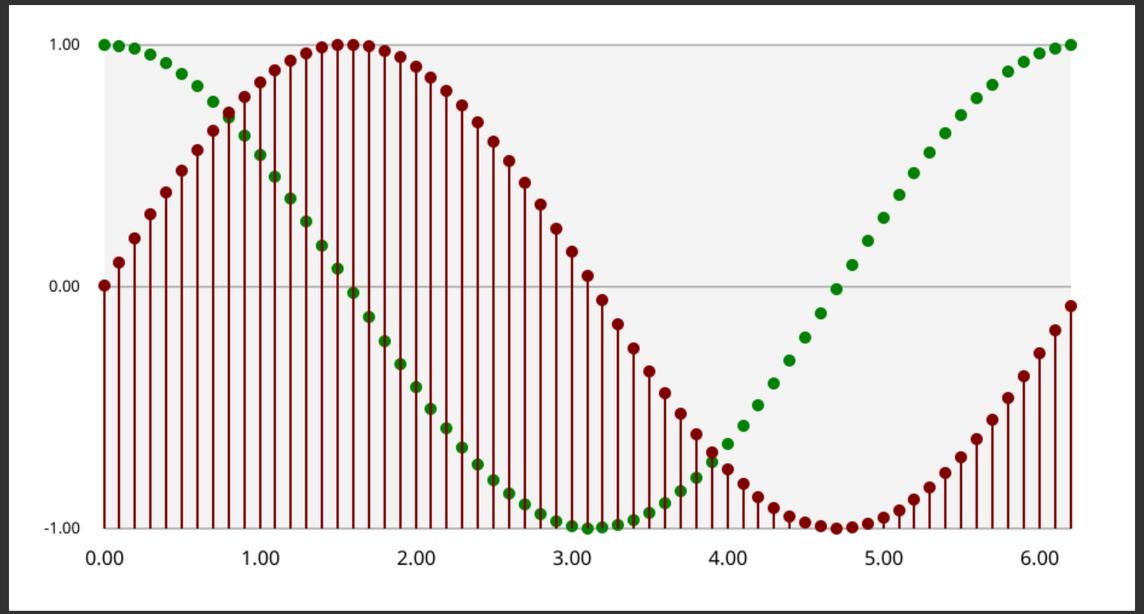
fc/chart: read data

```
sr, err := os.Open("sin.d")
if err != nil {
    return err
}
cr, err := os.Open("cos.d")
if err != nil {
    return err
}
sine, err := chart.DataRead(sr)
if err != nil {
    return err
}
cosine, err := chart.DataRead(cr)
if err != nil {
    return err
}
```

```
# y=sin(x)          # y=cos(x)
0.00    0.0000      0.00    1.0000
0.10    0.0998      0.10    0.9950
0.20    0.1987      0.20    0.9801
0.30    0.2955      0.30    0.9553
0.40    0.3894      0.40    0.9211
0.50    0.4794      0.50    0.8776
0.60    0.5646      0.60    0.8253
0.70    0.6442      0.70    0.7648
0.80    0.7174      0.80    0.6967
0.90    0.7833      0.90    0.6216
1.00    0.8415      1.00    0.5403
...                 ...
6.00    -0.2794     6.00    0.9602
6.10    -0.1822     6.10    0.9833
6.20    -0.0831     6.20    0.9965
```

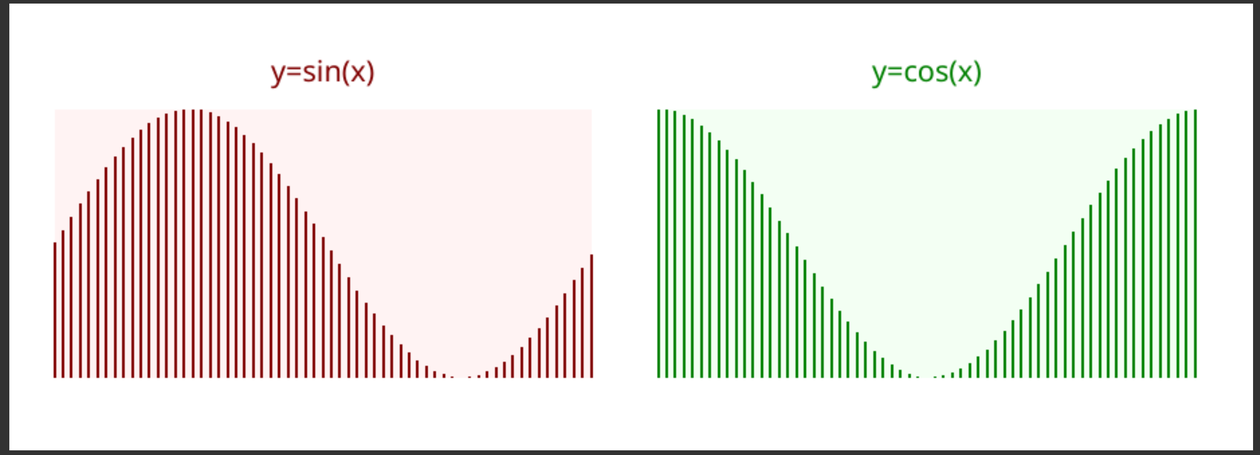
fc/chart: two data sets

```
cosine.Frame(canvas, 5)
cosine.Label(canvas, 1.5, 10)
cosine.YAxis(canvas, 1.2, -1.0, 1.0, 1.0, "%0.2f", true)
cosine.Color = color.RGBA{0, 128, 0, 255}
sine.Color = color.RGBA{128, 0, 0, 255}
cosine.Scatter(canvas, 1)
sine.Scatter(canvas, 1)
sine.Bar(canvas, 0.2)
```

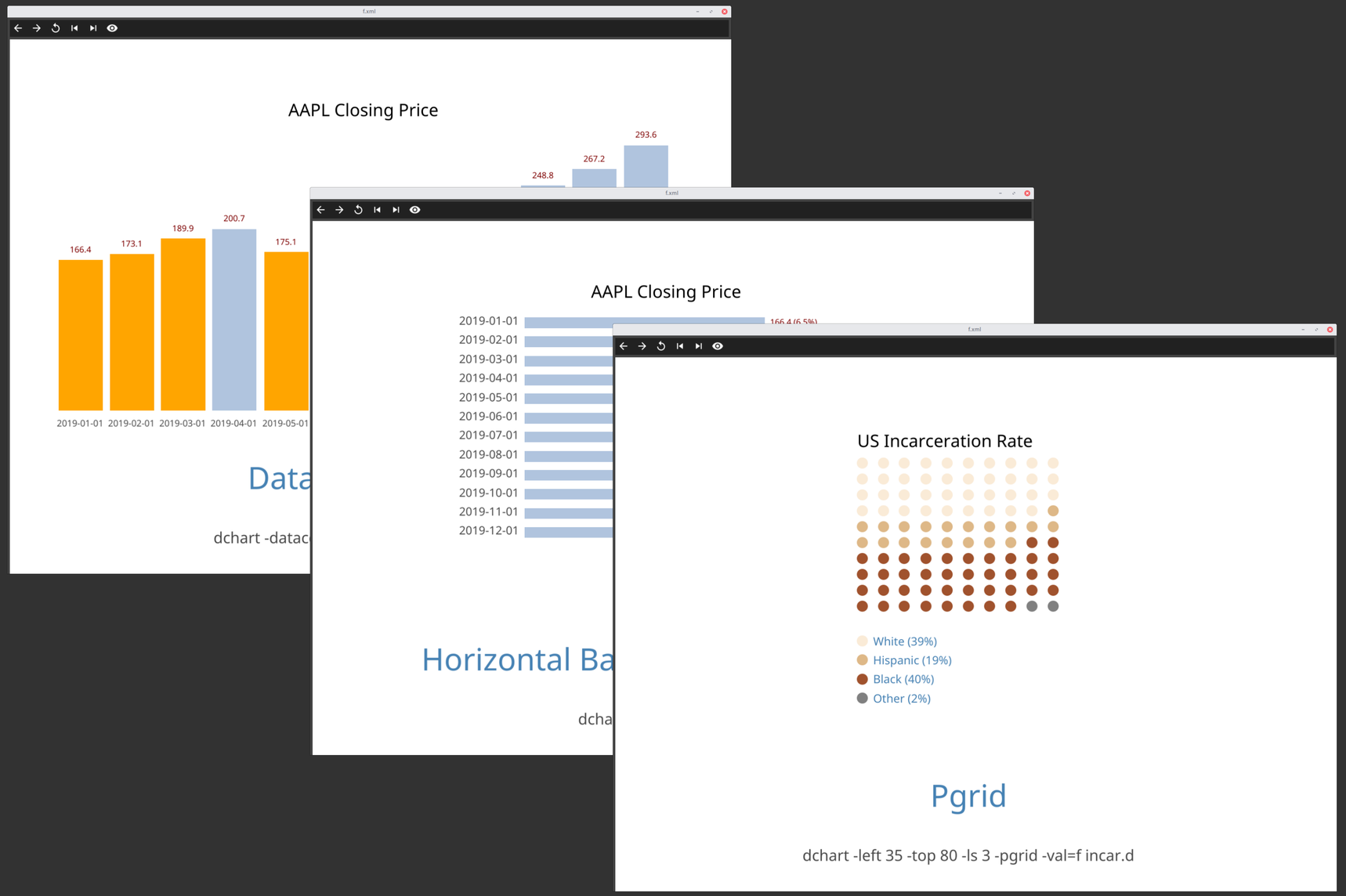


fc/chart: side by side

```
sine.Left = 10
sine.Right = sine.Left + 40
sine.Top, cosine.Top = 30, 30
sine.Bottom, cosine.Bottom = 10, 10

sine.CTitle(canvas, 2, 2)
sine.Frame(canvas, 5)
sine.Bar(canvas, 0.1)

offset := 45.0
cosine.Left = sine.Left + offset
cosine.Right = sine.Right + offset

cosine.CTitle(canvas, 2, 2)
cosine.Frame(canvas, 5)
cosine.Bar(canvas, 0.1)
```



fcdeck: decksh viewer



go get it

github.com/ajstarks/fc