Grafiske brugergrænseflader i F#

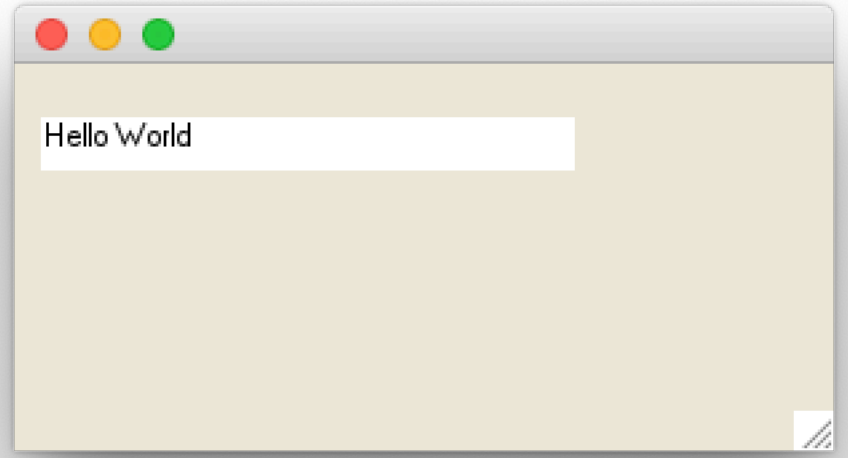
Programmering og problemløsning

Jon Sporring

Skrive på skærmen

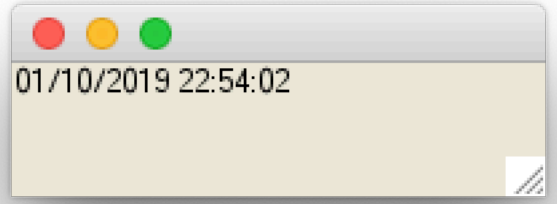
label.fsx

```
open System.Windows.Forms
open System.Drawing

let win = new Form () // make a window form
win.ClientSize <- Size (200, 100)

// make a label to show time
let label = new Label()
win.Controls.Add label
label.Width <- 200
label.Location <- new Point (10, 20)
label.Text <- "Hello World"
label.BackColor <- Color.White
label.Height <- 20

Application.Run win // start event-loop
```



System.Windows.Forms

clock.fsx




```
open System.Windows.Forms
open System.Drawing

let win = new Form () // make a window form
win.ClientSize <- Size (200, 50)

// make a label to show time
let label = new Label()
win.Controls.Add label
label.Width <- 200
label.Text <- string System.DateTime.Now // get present time and date

// make a timer and link to label
let timer = new Timer()
timer.Interval <- 1000 // create an event every 1000 millisecond
timer.Enabled <- true // activiate the timer
timer.Tick.Add (fun e -> label.Text <- string System.DateTime.Now)

Application.Run win // start event-loop
```

System.Windows.Forms

movingClock.fsx

```
open System.Windows.Forms
open System.Drawing

let win = new Form () // make a window form
win.ClientSize <- Size (200, 50)

// make a label to show time
let label = new Label()
win.Controls.Add label
label.Text <- string System.DateTime.Now // get present time and date
let textSz = TextRenderer.MeasureText(label.Text,label.Font)
label.Width <- textSz.Width
label.Height <- textSz.Height
label.BackColor <- Color.White
```

System.Windows.Forms

movingClock.fsx

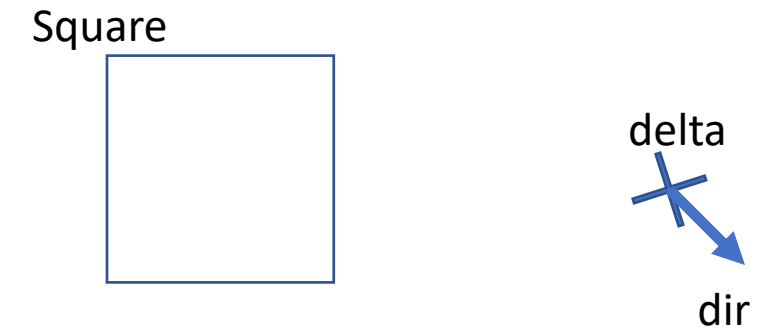
```
// make a timer and link to label
let timer = new Timer()
timer.Interval <- 100 // create an event every 1000 millisecond
timer.Enabled <- true // activiate the timer
let mutable pos = (0,0)
let mutable dir = (1,1)
let performTick (e : System.EventArgs) =
  printfn "%A %A" pos dir
  if fst pos + fst dir > win.ClientSize.Width - label.Width - 1
    || fst pos + fst dir < 0 then
      dir <- (-fst dir, snd dir);
  if snd pos + snd dir > win.ClientSize.Height - label.Height – 1
    || snd pos + snd dir < 0 then
      dir <- (fst dir, -snd dir);
  pos <- (fst pos + fst dir, snd pos + snd dir)
  label.Location <- Point (fst pos, snd pos);
  label.Text <- string System.DateTime.Now
timer.Tick.Add performTick

Application.Run win // start event-loop
```

System.Windows.Forms

movingSquare.fsx

```
open System.Windows.Forms
open System.Drawing

let win = new Form () // make a window form
win.ClientSize <- Size (200, 50)

. . .

// make a timer
let timer = new Timer()
timer.Interval <- 10 // create an event every 10 millisecond
timer.Enabled <- true // activiate the timer
timer.Tick.Add (fun e -> win.Refresh())

Application.Run win // start event-loop
```

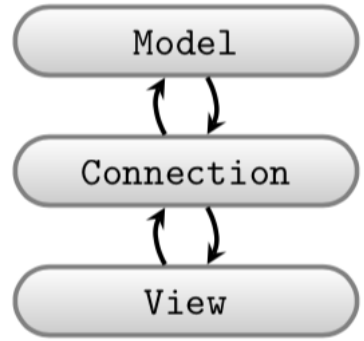
System.Windows.Forms



movingSquare.fsx

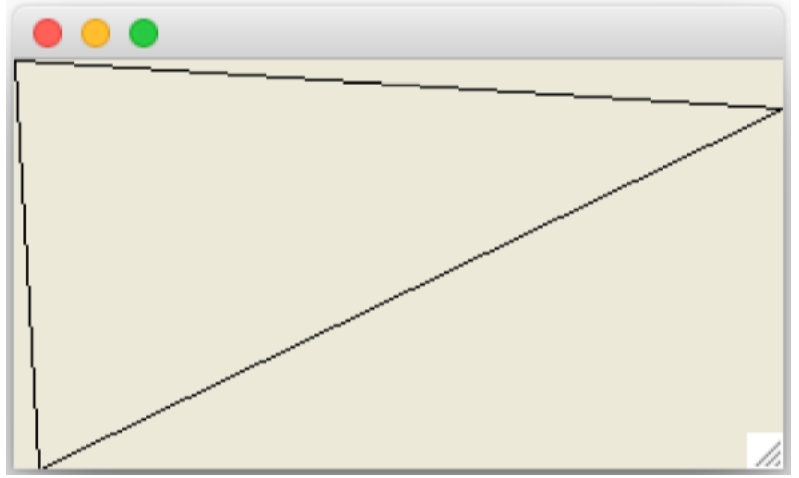
```
let mutable delta = Point (0,0)
let mutable dir = Point (1,1)
let polygonSz = Point (10,10);
let polygon = [|Point (0,0); Point (polygonSz.X,0); polygonSz; Point
(0,polygonSz.Y); Point (0,0)|]
let paint (e : PaintEventArgs) : unit =
  let pen = new Pen (Color.Black)
  if delta.X + dir.X < 0
    || delta.X + dir.X + polygonSz.X > win.ClientSize.Width - 1 then
      dir <- Point (-dir.X, dir.Y);
  if delta.Y + dir.Y < 0
    || delta.Y + dir.Y + polygonSz.Y > win.ClientSize.Height - 1 then
      dir <- Point (dir.X, -dir.Y);
  delta <- Point (delta.X + dir.X, delta.Y + dir.Y)
  let add (p : Point) -> Point (p.X + delta.X, p.Y + delta.Y)
  let points = Array.map add polygon
  e.Graphics.DrawLines (pen, points)
win.Paint.Add paint
```

Adskil model og view

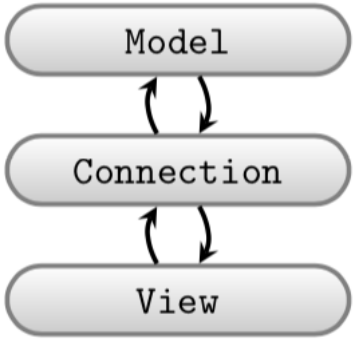


triangleClientSize.fsx

```
// Open often used libraries, beware of namespace pollution!
open System.Windows.Forms
open System.Drawing

// Prepare window form
let win = new Form ()
win.ClientSize <- Size (320, 170)

// Set paint call-back function
let paint (e : PaintEventArgs) : unit =
  let pen = new Pen (Color.Black)
  let points =
    [|Point (0,0); Point (10,170); Point (320,20); Point (0,0)|]
  e.Graphics.DrawLines (pen, points)
win.Paint.Add paint

Application.Run win // Start the event-loop.
```

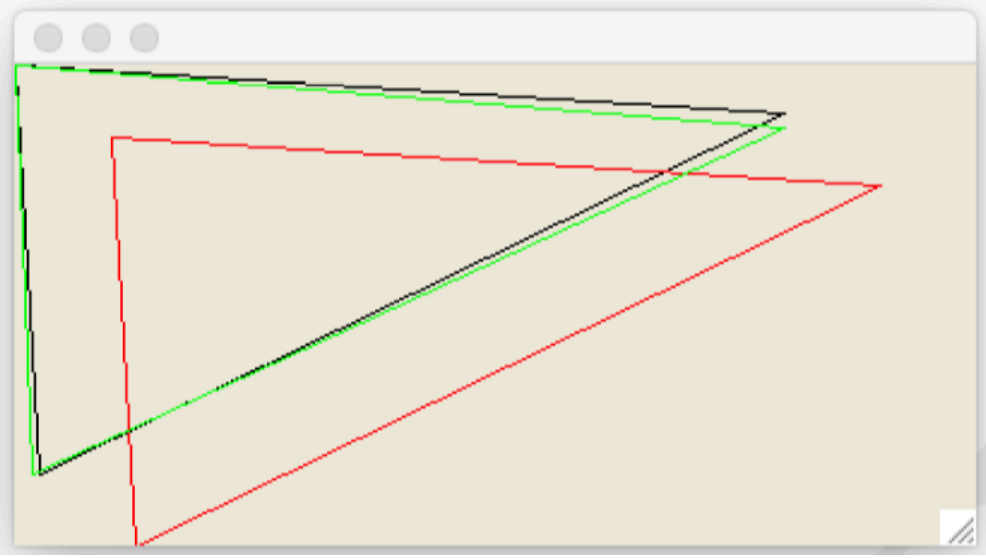


triangleOrganied.fsx




```
open System.Windows.Forms
open System.Drawing

////////////// WinForm specifics //////////////
/// Setup a window form and return function which can activate it
let view (sz : Size) (pen : Pen) (pts : Point []) : (unit -> unit) =
  let win = new Form ()
  win.ClientSize <- sz
  win.Paint.Add (fun e -> e.Graphics.DrawLines (pen, pts))
  fun () -> Application.Run win // function as return value

////////////// Model //////////////
// A black triangle, using winform primitives for brevity
let model () : Size * Pen * (Point []) =
  let size = Size (320, 170)
  let pen = new Pen (Color.FromArgb (0, 0, 0))
  let lines =
    [|Point (0,0); Point (10,170); Point (320,20); Point (0,0)|]
  (size, pen, lines)

////////////// Connection //////////////
// Tie view and model together and enter main event loop
let (size, pen, lines) = model ()
let run = view size pen lines
run ()
```

Tegn og transformer mange linjer



$$(a, b) = (x + \Delta x, y + \Delta y)$$

$$(a, b) = (x \cos\theta - y \sin\theta, x \sin\theta + y \cos\theta)$$

$$(a, b) = (sx, sy)$$

Model transformWindows.fsx

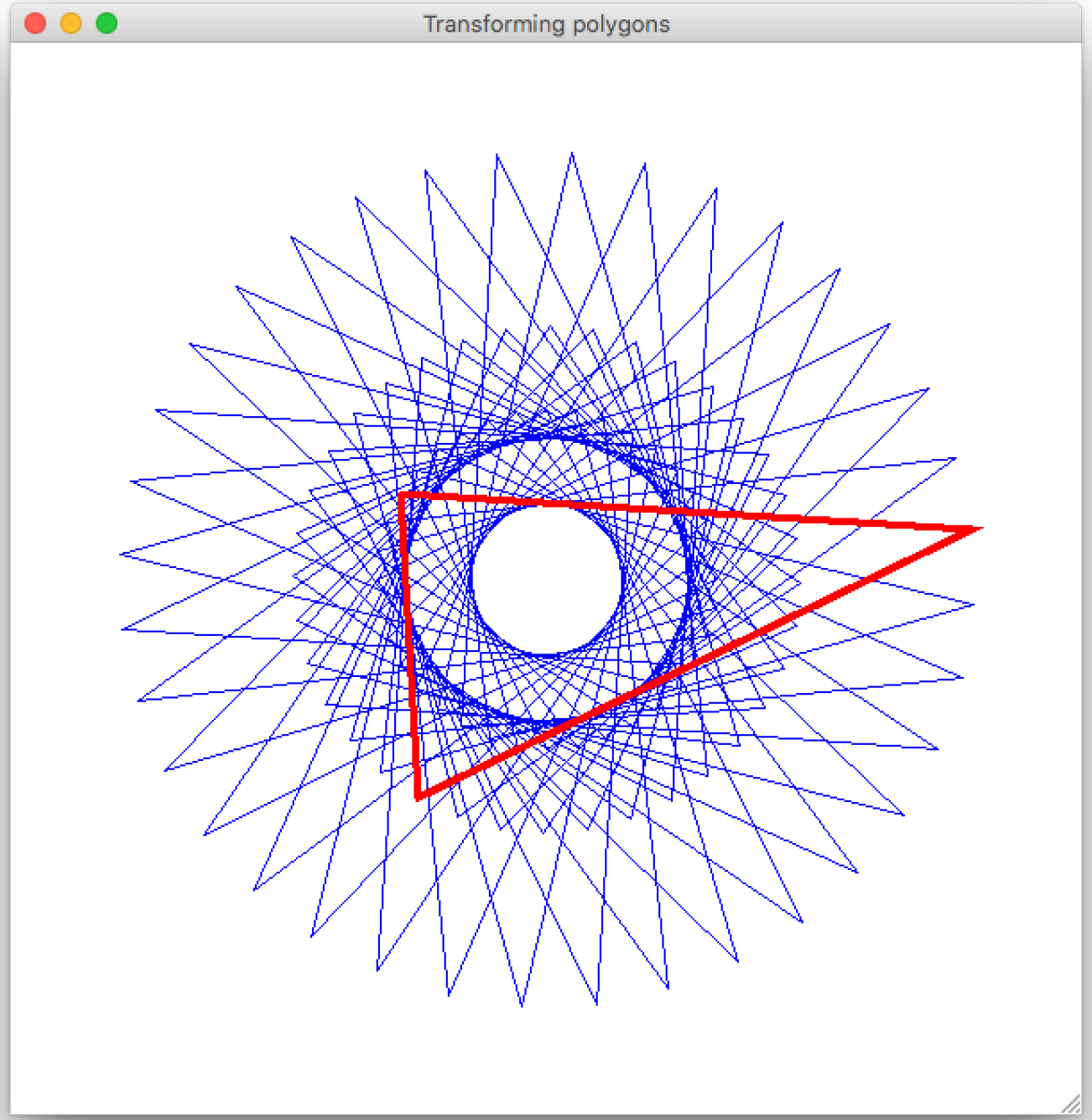
```
////////////// Model //////////////
// A black triangle, using WinForm primitives for brevity
let model () : Size * ((Pen * (Point [])) list) =
  /// Translate a primitive
  let translate (d : Point) (arr : Point []) : Point [] =
    let add (d : Point) (p : Point) : Point =
      Point (d.X + p.X, d.Y + p.Y)
    Array.map (add d) arr

  /// Rotate a primitive
  let rotate (theta : float) (arr : Point []) : Point [] =
    let toInt = int << round
    let rot (t : float) (p : Point) : Point =
      let (x, y) = (float p.X, float p.Y)
      let (a, b) = (x * cos t - y * sin t, x * sin t + y * cos t)
      Point (toInt a, toInt b)
    Array.map (rot theta) arr

  let size = Size (400, 200)
  let lines =
    [|Point (0,0); Point (10,170); Point (320,20); Point (0,0)|]
  let black = new Pen (Color.FromArgb (0, 0, 0))
  let red = new Pen (Color.FromArgb (255, 0, 0))
  let green = new Pen (Color.FromArgb (0, 255, 0))
  let shapes =
    [(black, lines);
     (red, translate (Point (40, 30)) lines);
     (green, rotate (1.0 *System.Math.PI / 180.0) lines)]
  (size, shapes)
```

View+forbindelse **transformWindows.fsx**

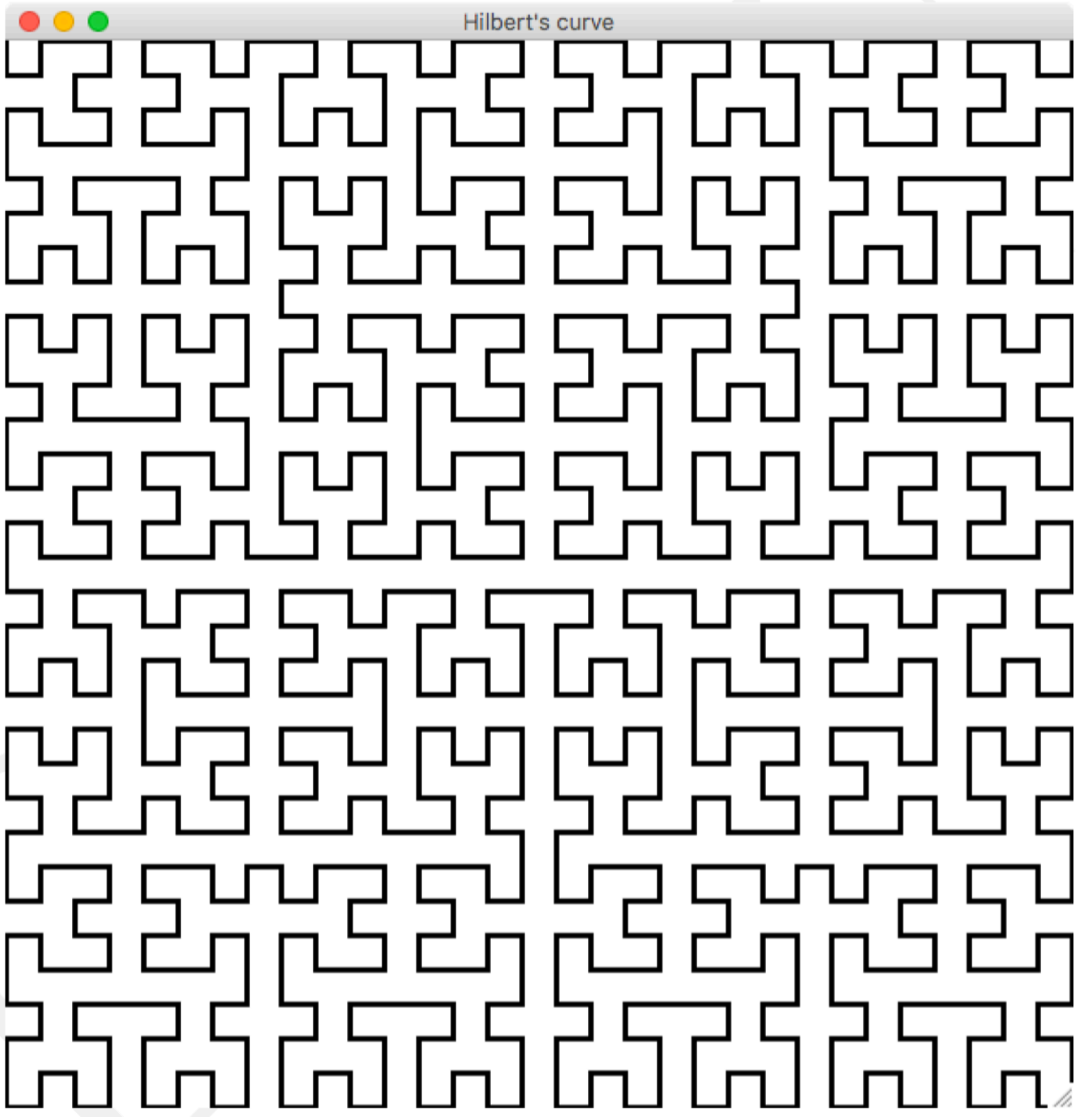
```
// Open often used libraries, beware of namespace polution!
open System.Windows.Forms
open System.Drawing

////////////// WinForm specifics //////////////
/// Setup a window form and return function to activate
let view (sz : Size) (shapes : (Pen * (Point [])) list) : (unit -> unit) =
  let win = new Form ()
  win.ClientSize <- sz
  let paint (e : PaintEventArgs) ((p, pts) : (Pen * (Point []))) : unit =
    e.Graphics.DrawLines (p, pts)
  win.Paint.Add (fun e -> List.iter (paint e) shapes)
  fun () -> Application.Run win // function as return value
```

```
////////////// Connection //////////////
// Tie view and model together and enter main event loop
let (size, shapes) = model ()
let run = view size shapes
run ()
```

rotationalSymmetry.fsx



hilbert.fsx



Input fra brugeren via Controls

buttonControl.fsx


```
open System.Windows.Forms
open System.Drawing

let win = new Form () // make a window form
win.ClientSize <- Size (140, 120)

// Create a label
let label = new Label()
win.Controls.Add label
label.Location <- new Point (20, 20)
label.Width <- 120
let mutable clicked = 0
let setLabel clicked =
  label.Text <- sprintf "Clicked %d times" clicked
setLabel clicked

// Create a button
let button = new Button ()
win.Controls.Add button
button.Size <- new Size (100, 40)
button.Location <- new Point (20, 60)
button.Text <- "Click me"
button.Click.Add (fun e -> clicked <- clicked + 1; setLabel clicked)

Application.Run win // Start the event-loop.
```

Input fra brugeren via Controls

buttonControlCompact.fsx

```
open System.Windows.Forms
open System.Drawing

// Model: a state 'clicked' that counts how many times an event has occurred
let mutable clicked = 0
let message () = sprintf "Clicked %d times" clicked
let update () = clicked <- clicked + 1

// View: A window containing a label and a button
let win = new Form(ClientSize=Size(140, 120))
let label = new Label(Location=new Point(20, 20), Width=120)
let button = new Button(Size=new Size(100, 40), Location=new Point(20, 60),
Text="Click me")
win.Controls.Add label
win.Controls.Add button

// Connect model and view and start the event-loop
label.Text <- message ()
button.Click.Add (fun e -> update (); label.Text <- message ())
Application.Run win
```

Evaluering: Målbeskrivelse

Viden

- Grundlæggende begreber indenfor imperativ, objektorienteret og funktionsprogrammeringsparadigmerne: Funktioner og metoder, variabler, udtryk, typer, kontrolstrukturer, løkker, blokstruktur, klasser og objekter, objektinteraktion, nedarvning, rekursion, polymorfi, abstraktion, undtagelser, pattern matching over rekursive datatyper, m.m.
- God programmeringsskik: Dokumentation i koden, design patterns, afprøvning inkl. unit testing, håndtering af køretidsfejl, m.m.
- Teknikker til problemløsning: Teknisk analyse af naturligsprogsproblemer, objektorienteret design, modelleringssprog, håndkøring, m.m.
- God rapportskrivningsteknik.

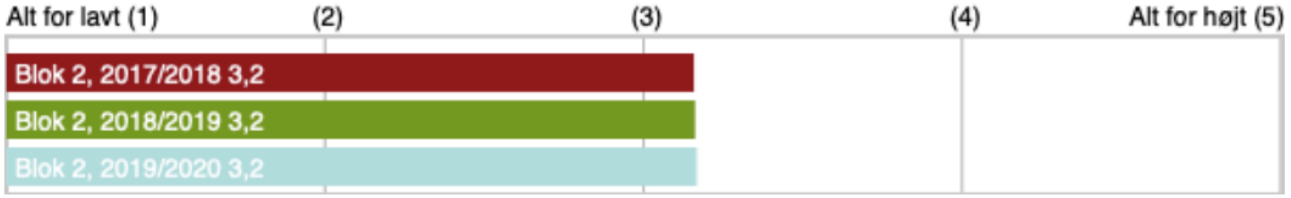
Færdigheder

- At kunne lave mindre programmer (op til ca. 1000 linjer) i de programmeringsparadigmer, der undervises i på kurset, med overholdelse af god programmeringsskik og -stil.
- At kunne evaluere fordele og ulemper ved at opskrive løsningen i de underviste programmeringsparadigmer, og at kunne implementere, afprøve, dokumentere, og evaluere løsningens kvalitet.
- Et sideordnet mål er, at den studerende opnår passende studieteknik således, at dette og parallelkurser bestås svarende til et fuldtidsstudium.

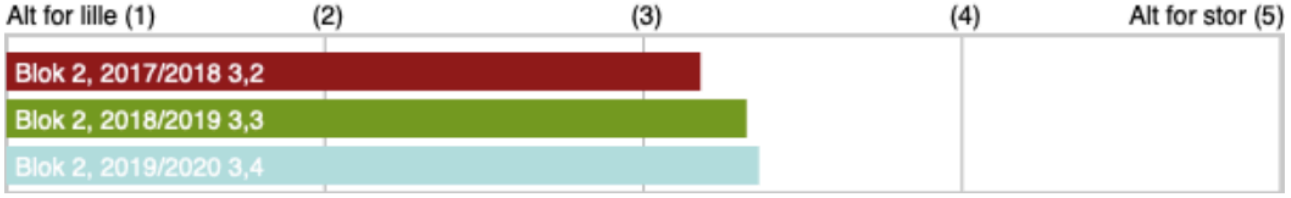
Kompetencer

- Ud fra en præcist defineret problemformulering at kunne analysere problemet, udforme et program til løsning af dette, samt at verificere, afprøve, og dokumentere løsningen.

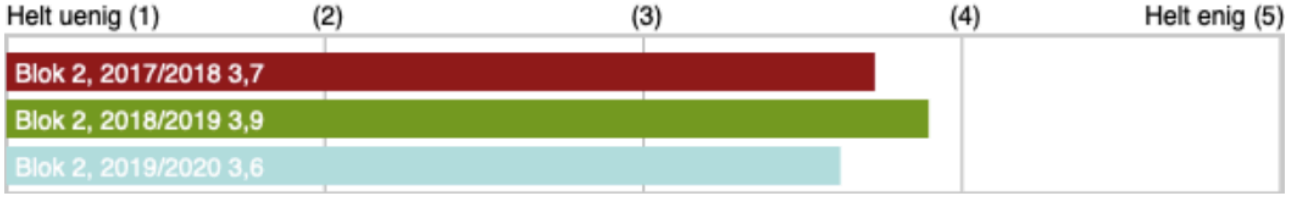
2.1 I forhold til mine egne forudsætninger oplever jeg, at kursets faglige niveau er:



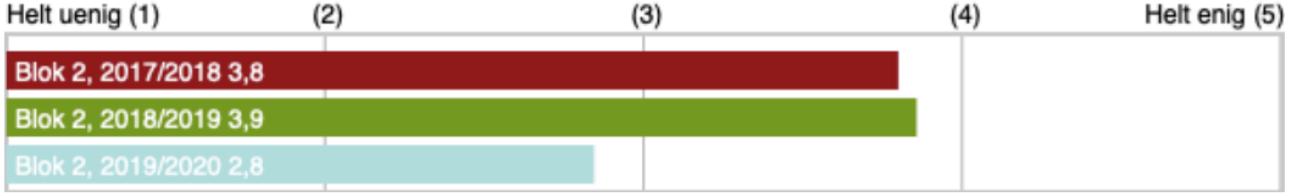
2.2 Jeg oplever arbejdsbyrden på kurset som:

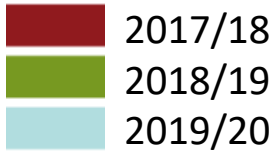


2.3 Jeg mener at have opnået kompetencerne beskrevet i kursusmålene [Se kursets læringsmål fra kursusbeskivelsen]

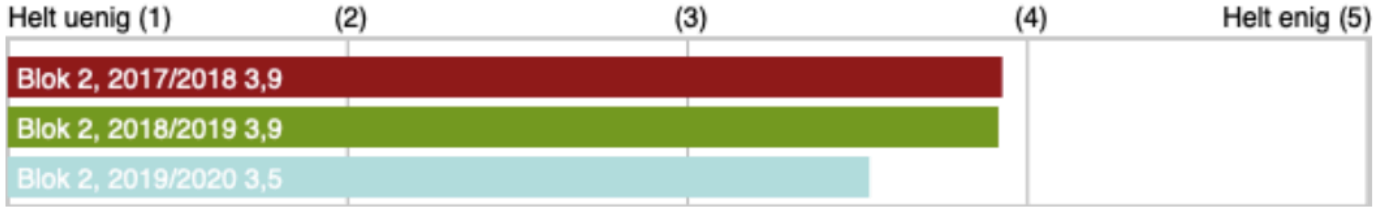


2.4 Jeg oplevede, at der var en god sammenhæng mellem de forskellige delelementer (forelæsninger, øvelser m.v.), der indgik i kurset (Uddyb gerne på næste side)



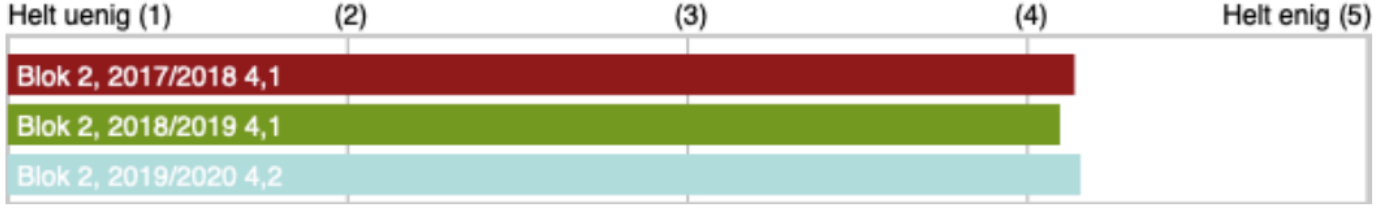


2.5 Jeg synes, at undervisningsmaterialet var relevant i forhold til kurset

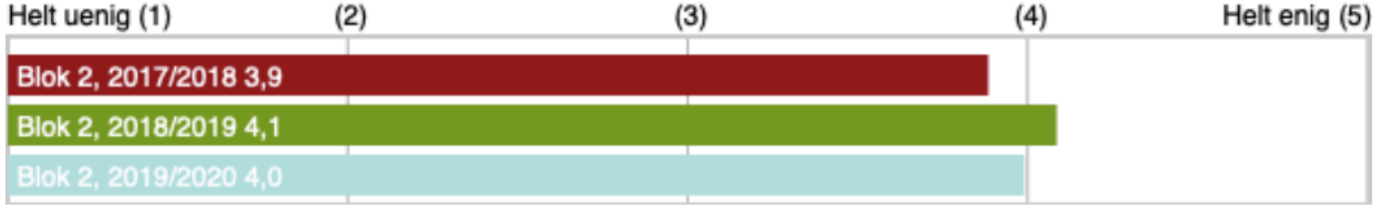


2017/18
2018/19
2019/20

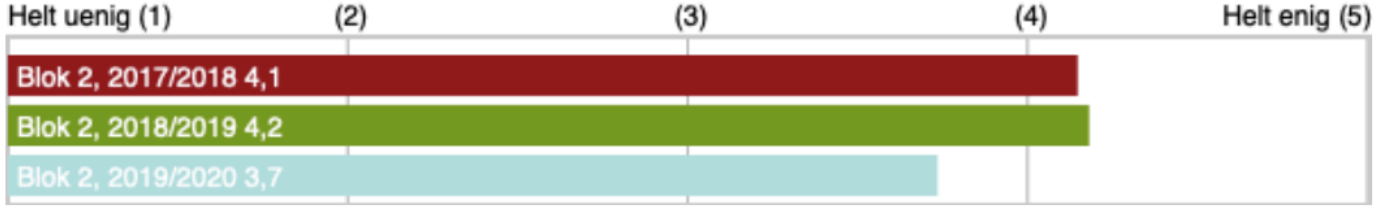
2.6 Jeg synes, at jeg har fået relevant faglig respons på mit skriftlige og mundtlige arbejde på kurset



2.7 Jeg synes, at jeg har haft adgang til de nødvendige informationer omkring kurset



2.8 Jeg synes samlet set, at kurset har været udbytterigt

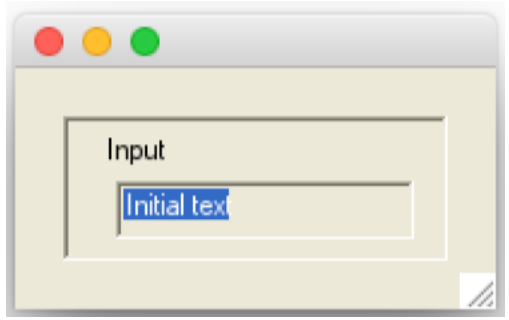


Husk:

- Hvis I ikke har bestået mindst 11 ud af 12 opgaver, så har I mulighed for at genaflevere gamle opgaver lidt endnu
- Reeksamen er 15-16 April

Det var det!

Organisering af Controls i grupper: Panels



panel.fsx

```
open System.Drawing
open System.Windows.Forms

// Create a window with a panel, label and a textbox
let win = new Form(ClientSize=new Size (200, 100))
let panel = new Panel(ClientSize=new Size(160, 60), Location=new Point(20,20), BorderStyle=BorderStyle.Fixed3D)
let label = new Label(ClientSize=new Size(120, 20), Location=new Point(15,5), Text="Input")
let textBox = new TextBox(ClientSize=new Size(120, 20), Location=new Point(20,25), Text="Initial text")

win.Controls.Add panel // Add panel to window
panel.Controls.Add label // add label to panel
panel.Controls.Add textBox // add textbox to panel

Application.Run win // Start the event-loop
```

Automatisk tildeling af position i paneler

flowLayoutPanel.fsx

```
open System.Windows.Forms
open System.Drawing

// Create a window, a FlowLayoutPanel, 4 buttons, a checkbox, a panel, and 4 radiobuttons
let win = new Form(ClientSize=new Size(302, 356), Text="A Flowlayout Example")
let flowLayoutPanel = new FlowLayoutPanel(Location=new Point(47, 55), BorderStyle=BorderStyle.Fixed3D, WrapContents=true)
let buttonLst =
  [new Button(Text="Button0");
   new Button(Text="Button1");
   new Button(Text="Button2");
   new Button(Text="Button3")]
let panel = new Panel(Location=new Point (47, 190),BorderStyle=BorderStyle.Fixed3D)
let wrapContentsCheckBox = new CheckBox(Location=new Point (3, 3), Text="Wrap Contents")
let radioButtonLst =
  [(new RadioButton(Location=new Point(3, 34), Text="TopDown"), FlowDirection.TopDown);
   (new RadioButton(Location=new Point(3, 62), Text="BottomUp"), FlowDirection.BottomUp);
   (new RadioButton(Location=new Point(111, 34), Text="LeftToRight"), FlowDirection.LeftToRight);
   (new RadioButton(Location=new Point(111, 62), Text="RightToLeft"), FlowDirection.RightToLeft)]
```

Automatisk tildeling af position i paneler

flowLayoutPanel.fsx

```
// The window contains the panels which in turn contains the buttons, checkbox and r
win.Controls.Add flowLayoutPanel
for btn in buttonLst do
  flowLayoutPanel.Controls.Add btn
win.Controls.Add panel
panel.Controls.Add (wrapContentsCheckBox)
for btn, dir in radioButtonLst do
  panel.Controls.Add (btn)

// Link wrapContentsCheckBox and flowLayoutPanel.WrapContents
wrapContentsCheckBox.Checked <- flowLayoutPanel.WrapContents
wrapContentsCheckBox.CheckedChanged.Add (fun _ -> flowLayoutPanel.WrapContents <- wr

// Link radio buttons and flowLayoutPanel.FlowDirection
for (btn, dir) in radioButtonLst do
  btn.Checked <- flowLayoutPanel.FlowDirection = dir
  btn.CheckedChanged.Add (fun _ -> flowLayoutPanel.FlowDirection <- dir)

// Create a window, add controls, and start event-loop
Application.Run win
```