# Learning to Program with F#
# Exercises
# Department of Computer Science
# University of Copenhagen

Jon Sporring, Martin Elsman, Torben Mogensen, Christina Lioma

October 21, 2022

## 0.1 Tree structure

### 0.1.1 Teacher's guide

**Emne** Træstrukturer og grafik

**Sværhedsgrad** Middel

### 0.1.2 Introduction

I de følgende opgaver skal vi arbejde med en træstruktur til at beskrive geometriske figurer med farver. For at gøre det muligt at afprøve jeres opgaver skal I gøre brug af det udleverede bibliotek `img_util.dll`, der blandt andet kan omdanne såkaldte canvas-objekter til png-filer. Biblioteket er beskrevet i forelæsningerne (i kursusuge 7) og koden for biblioteket er tilgængeligt via github på `https://github.com/diku-dk/img-util-fs`.

Her bruger vi funktionerne til at tegne på et canvas samt til at gemme canvas-objektet som en png-fil:[1]

```
// colors
type color
val fromRgb : int * int * int -> color

// canvas
type canvas
val mk       : int -> int -> canvas
val setPixel : color -> int * int -> canvas -> unit

// save a canvas as a png file
val toPngFile : string -> canvas -> unit
```

Funktionen `toPngFile` tager som det første argument navnet på den ønskede png-fil (husk extension). Det andet argument er canvas-objektet som ønskes konverteret og gemt. Et canvas-objekt kan konstrueres med funktionen `ImgUtil.mk`, der tager som argumenter vidden og højden af billedet i antal pixels, samt funktionen `ImgUtil.setPixel`, der kan bruges til at opdatere canvas-objektet før det eksporteres til en png-fil. Funktionen `ImgUtil.setPixel` tager tre argumenter. Det første argument repræsenterer en farve og det andet argument repræsenterer et punkt i canvas-objektet (dvs. i billedet). Det tredie argument repræsenterer det canvas-objekt, der skal opdateres. En farve kan nu konstrueres med funktionen `ImgUtil.fromRgb` der tager en triple af tre tal mellem 0 og 255 (begge inklusive), der beskriver hhv. den røde, grønne og blå del af farven.

Koordinatsystemet har nulpunkt $(0,0)$ i øverste venstre hjørne og, såfremt vidden og højden af koordinatsystemet er henholdsvis $w$ og $h$, optræder punktet $(w-1,h-1)$ i nederste højre hjørne. Antag for eksempel at programfilen `testPNG.fsx` indeholder følgende F# kode:

```
let C = ImgUtil.mk 256 256
do ImgUtil.setPixel (ImgUtil.fromRgb (255,0,0)) (10,10) C
do ImgUtil.toPngFile "test.png" C
```

[1] Bemærk at interfacet ikke definerer de konkrete repræsentationstyper for typerne `color` og `canvas`. Disse typer er holdt *abstrakte*, hvilket vil sige at deres repræsentationer ikke kan ses af brugeren af modulet.

Det er nu muligt at generere en png-fil med navn `test.png` ved at køre følgende kommando:

```
fsharpi -r img_util.dll testPNG.fsx
```

Den genererede billedfil `test.png` vil indeholde et hvidt billede med et pixel af rød farve i punktet (10,10).

Bemærk, at alle programmer, der bruger `ImgUtil` skal køres eller oversættes med `-r img_util.dll` som en del af kommandoen.

Bonus information, hvis I på et tidpunkt skulle få brug for at inkludere png-filer, fx skabt vha `ImgUtil`, i et LaTeX dokument, så gøres det med LaTeX kommandoen `\includegraphics`.

I det følgende vil vi repræsentere geometriske figurer med følgende datastruktur:

```
type point = int * int // a point (x, y) in the plane
type color = ImgUtil.color

type figure =
  | Circle of point * int * color
     // defined by center, radius, and color
  | Rectangle of point * point * color
     // defined by corners top-left, bottom-right, and color
  | Mix of figure * figure
     // combine figures with mixed color at overlap
```

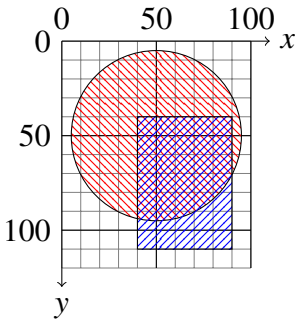
Man kan, for eksempel, lave følgende funktion til at finde farven af en figur i et punkt. Hvis punktet ikke ligger i figuren, returneres `None`, og hvis punktet ligger i figuren, returneres `Some` $c$, hvor $c$ er farven.

```
// finds color of figure at point
let rec colorAt (x,y) figure =
  match figure with
  | Circle ((cx,cy), r, col) ->
      if (x-cx)*(x-cx)+(y-cy)*(y-cy) <= r*r
        // uses Pythagoras' equation to determine
        // distance to center
      then Some col else None
  | Rectangle ((x0,y0), (x1,y1), col) ->
     if x0<=x && x <= x1 && y0 <= y && y <= y1
        // within corners
     then Some col else None
  | Mix (f1, f2) ->
      match (colorAt (x,y) f1, colorAt (x,y) f2) with
      | (None, c) -> c  // no overlap
      | (c, None) -> c  // no overlap
      | (Some c1, Some c2) ->
      let (a1,r1,g1,b1) = ImgUtil.fromColor c1
      let (a2,r2,g2,b2) = ImgUtil.fromColor c2
      in Some(ImgUtil.fromArgb((a1+a2)/2, (r1+r2)/2,   // calculate
                               (g1+g2)/2, (b1+b2)/2))  // average
   color
```

Bemærk, at punkter på cirklens omkreds og rektanglens kanter er med i figuren. Farver blandes ved at lægge dem sammen og dele med to, altså finde gennemsnitsfarven.

### 0.1.3 Exercise(s)

**0.1.3.1:** Lav en figur `figTest : figure`, der består af en rød cirkel med centrum i (50,50) og radius 45, samt en blå rektangel med hjørnerne (40,40) og (90,110), som illustreret i tegningen nedenfor (hvor vi dog har brugt skravering i stedet for udfyldende farver.)




**0.1.3.2:** Brug `ImgUtil`-funktionerne og `colorAt` til at lave en funktion

```
makePicture : string -> figure -> int -> int -> unit
```

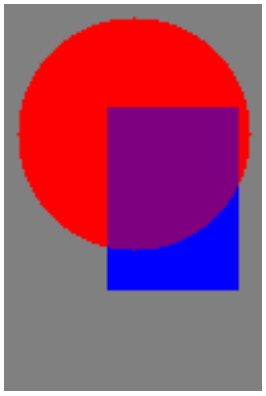
sådan at kaldet `makePicture` *`filnavn figur b h`* laver en billedfil ved navn *`filnavn`*`.png` med et billede af *`figur`* med bredde *`b`* og højde *`h`*.

På punkter, der ingen farve har (jvf. `colorAt`), skal farven være grå (som defineres med RGB-værdien (128,128,128)).

Du kan bruge denne funktion til at afprøve dine opgaver.

**0.1.3.3:** Brug funktionen `makePicture` til at konstruere en billedfil med navnet `figTest.png` og størrelse $100 \times 150$ (bredde 100, højde 150), der viser figuren `figTest` fra Opgave 1.

Resultatet skulle gerne ligne figuren nedenfor.



**0.1.3.4:** Lav en funktion `checkFigure : figure -> bool`, der undersøger, om en figur er korrekt: At radiusen i cirkler er ikke-negativ og at øverste venstre hjørne i en rektangel faktisk er ovenover og til venstre for det nederste højre hjørne (bredde og højde kan dog godt være 0).

**0.1.3.5:** Lav en funktion `move : figure -> int * int -> figure`, der givet en figur og en vektor flytter figuren langs vektoren.

Ved at foretage kaldet

```
makePicture "moveTest" (move figTest (-20,20)) 100 150
```

skulle der gerne laves en billedfil `moveTest.png` med indholdet vist nedenfor.

**0.1.3.6:** Lav en funktion `boundingBox : figure -> point * point`, der givet en figur finder hjørnerne (top-venstre og bund-højre) for den mindste akserette rektangel, der indeholder hele figuren.

Funktionskaldet `boundingBox figTest` skulle gerne give resultatet `((5, 5), (95, 110))`.