Skjelpaddekunst

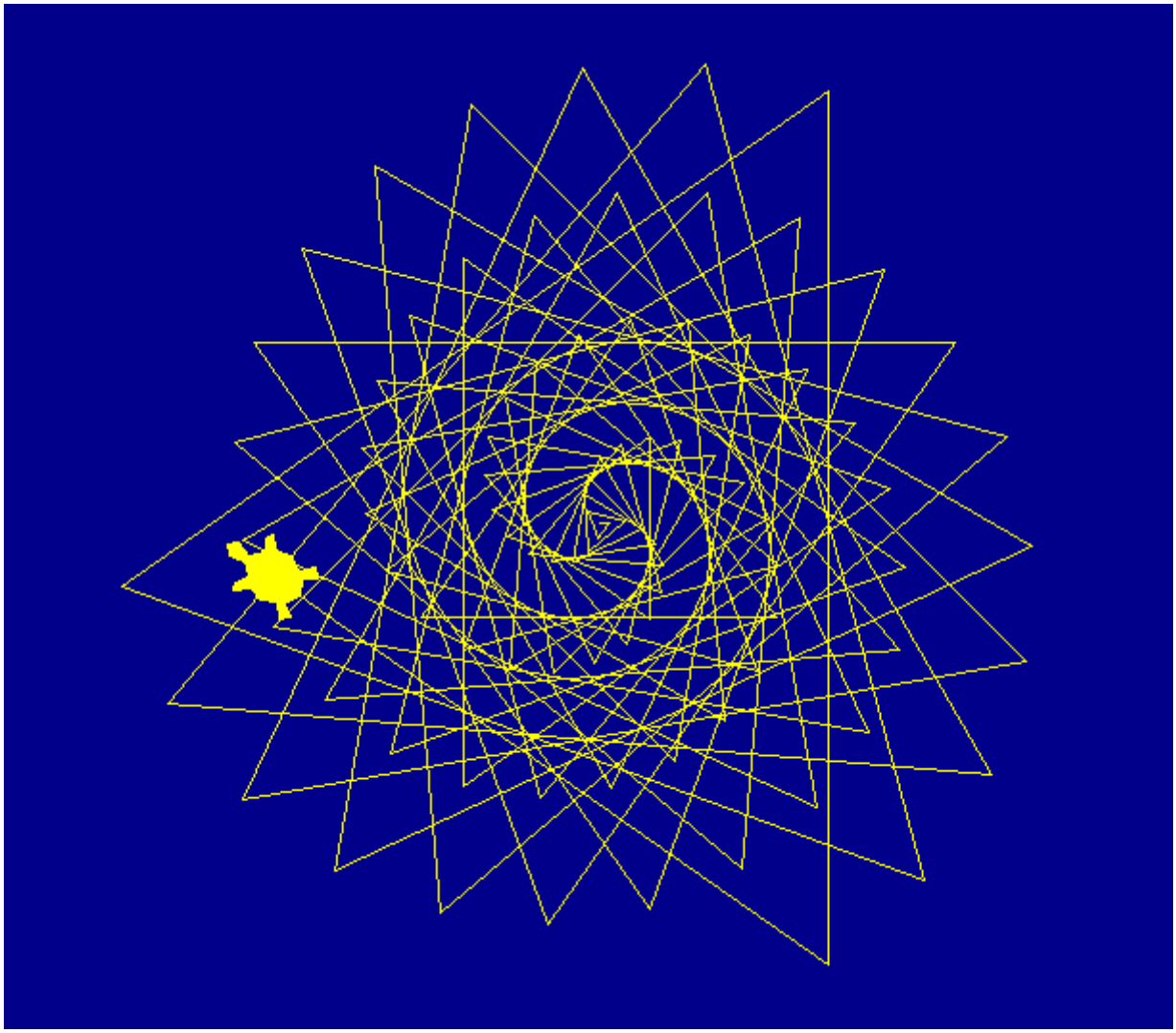
Skrevet av: Geir Arne Hjelle

Oversatt av: Stein Olav Romslo

Kurs: Python
Tema: Tekstbasert
Fag: Matematikk, Programmering, Kunst og håndverk
Klassetrinn: 5.-7. klasse, 8.-10. klasse

Introduksjon

Skjelpadder (turtles på engelsk) er ein form for robotar som har vore i bruk i programmering i lang tid. Me vil bruke skjelpadde-biblioteket i Python til å utforske fleire programmeringskonsept, samstundes som me teiknar kule bilete.



Steg 1: Møt skjelpadda

For å bruke skjelpadder i Python må me importere eit bibliotek som heiter `turtle`. I Python heiter slike bibliotek *modules*, og blir brukt for å organisere og gjenbruke kode som andre har skrive. Det finst fleire måtar å importere bibliotek på i Python. Her brukar me den enklaste, og startar alle skjelpaddeprogramma våre med linja

```
from turtle import *
```

Her tyder `*` alt, slik at linja seier "Importer all kode frå turtle-biblioteket".

Turtles

Namnet **Turtle** tyder *skjelpadde* på norsk. Bakgrunnen til dette namnet er historisk. For nesten 70 år sidan bygde William Grey Walter eit par robotar som kunne bevege seg rundt. Desse bevegde seg ganske sakte, var låge og skalforma. Difor fekk dei etter kvart kallenamnet skjelpadder.

Seinare vart måten desse bevega seg på (me skal snart sjå korleis) tatt inn i ulike programmeringsspråk, spesielt som ein måte å teikne på. Språket *Logo* er det som nok er mest kjent for slik skjelpaddegrafikk, men nesten alle programmeringsspråk støttar det i dag, inkludert til dømes *Scratch*, *Lua* og *Python*.

Sjekkliste

- [ ] Det er på tide å lage vår fyrste skjelpadde. Start IDLE og åpne eit nytt programmeringsvindauge. I det nye vindauget kan du skrive inn følgjande kode:

  ```
  from turtle import *

  shape('turtle')
  shapesize(2)
  bgcolor('darkblue')
  color('yellow')
  ```

  Lagre programmet med namnet `skjelpadde.py` og køyr det. No skal du få opp eit nytt vindauge med ei gul skjelpadde på ein blå bakgrunn. Viss det ikkje skjer kan du sjå i det opphavlege `Python Shell`-vindauget om du har fått ei feilmelding.

- [ ] La oss sjå meir på kva programmet gjer så langt. Det er ein god idé å prøve å endre på ting i programmet for å sjå effekten av endringane, og slik betre forstå korleis ting virkar.

- [ ] Linja `shape('turtle')` seier at me vil bruke ein skjelpaddefigur. I staden for `turtle` kan du prøve `arrow`, `circle`, `square`, `triangle` eller `classic`.

- [ ] Med `shapesize(2)` fortel me programmet kor stor skjelpaddefiguren skal vere. Prøv med andre tal!

- [ ] Kommandoane `bgcolor` og `color` bestemmer farga på henhaldsvis bakgrunnen og skjelpadda. Python kjenner til veldig mange farger (men berre på engelsk), så prøv om du kan endre fargene til noko du likar.

- [ ] I dei neste programma me skal lage vil me bruke desse linjene på toppen. Du kan gjerne bruke ein variant av farger og figur som du likar betre i staden.

Steg 2: Ei kunstnerisk skjelpadde

Skjelpadda er ikkje berre fin å sjå på, den kan teikne! I dette steget skal me bli kjent med nokre enkle kommandoar som gjer skjelpadda om til ein kunstnar.

Sjekkliste

- [ ] Legg til ei linje nedst i programmet ditt, så det ser slik ut:

```
from turtle import *

shape('turtle')
shapesize(2)
bgcolor('darkblue')
color('yellow')

forward(200)
```

- [ ] Når du køyrer programmet vil du sjå at skjelpadda har bevega seg framover eit lite stykke, og at den har teikna ein strek der den gjekk.

- [ ] I tillegg til `forward` kan me bruke kommandoane `backward` for å gå bakover, `left` for å svinge mot venstre og `right` for å svinge mot høgre. Prøv til dømes å endre programmet ditt til det følgjande:

```
from turtle import *

shape('turtle')
shapesize(2)
bgcolor('darkblue')
color('yellow')

forward(200)
left(60)
forward(50)
backward(200)
right(90)
forward(100)
```

Ser du at skjelpadda utfører alle kommandoane du gir den?

- [ ] Viss me set saman kommandoane litt systematisk kan me teikne nokre grunnleggjande geometriske figurar. Til dømes kan me teikne ein firkant ved å gå framover, svinge 90 grader (mot høgre eller venstre), framover att, svinge, framover, svinge og til slutt framover ein gong til. I Python kan me skrive det som

```
from turtle import *

shape('turtle')
shapesize(2)
bgcolor('darkblue')
color('yellow')

forward(100)
right(90)
forward(100)
right(90)
forward(100)
right(90)
forward(100)
right(90)
```

Teiknar skjelpadda ein firkant når du køyrer dette programmet?

- [ ] Kva med ein trekant? Korleis må du forandre koden din for at skjelpadda skal teikne ein trekant i staden for ein firkant? Prøv sjølv å endre koden og køyr den. Vart resultatet som du trudde?

Steg 3: Gjenta deg sjølv

Viss du ser på koden me har brukt for å teikne trekantar og firkantar har me gjenteke oss sjølv fleire gonger. I staden for å skrive same kode om att og om att kan me be Python om å gjenta delar av koden. Til det brukar me ei **for-løkke**.

Sjekkliste

- [ ] Det følgjande programmet teiknar òg ein firkant, akkurat som det tidlegare programmet me laga:

```
from turtle import *

shape('turtle')
shapesize(2)
bgcolor('darkblue')
color('yellow')

for i in range(4):
    forward(100)
    right(90)
```

Endre koden din som over, og køyr programmet.

- [ ] Legg merke til at linjene som kjem etter `for` er rykka inn til høgre. Det er veldig viktig i Python, fordi det fortel kor mykje kode som skal bli gjenteke i løkka. For å rykke inn koden på denne måten kan du bruke `Tab` -tasten i IDLE. For å trekkje koden tilbake til venstre kan du trykkje `Shift` og `Tab` .

- [ ] Prøv å trekkje linja `right(90)` til venstre, slik at for-løkka ser slik ut:

```
for i in range(4):
    forward(100)
right(90)
```

Kva trur du programmet ditt gjer no? Prøv å køyre det for å sjå om du har rett!

Skjelpadda vil berre gå framover fire gonger, og så svinge til høgre ein gong til slutt. Det gjer at me får ei rett linje i staden for ein firkant.

- [ ] No som me brukar ei for-løkke har det blitt mykje enklare å endre koden til å til dømes teikne ein trekant. Me må endre 4 til 3 i for-løkka og i tillegg endre vinkelen skjelpadda snur seg i kvart hjørne. For at skjelpadda skal gå ein runde rundt trekanten må den snu totalt 360 gradar. Sidan den snur tre gonger må den snu 120 gradar (360 delt på 3) kvar gong. Programmet for å teikne ein trekant blir dermed slik:

```
from turtle import *

shape('turtle')
shapesize(2)
bgcolor('darkblue')
color('yellow')

for i in range(3):
    forward(100)
    right(120)
```

- [ ] Prøv å endre programmet slik at det teiknar andre mangekantar. Korleis kan du teikne ein femkant, åttekant eller femtankant?

Steg 4: Alle ting fortener eit namn

Me skal halde fram med å gjere koden vår endå meir fleksibel ved å gi ting namn. Dette vil òg gjere det enklare å forstå kva koden gjer.

Sjekkliste

- [ ] Me innfører variablar som seier kor mange sider me vil teikne, kor lang kvar side skal vere og kor mange gradar me skal snu ved kvart hjørne. Endre programmet ditt så det ser slik ut:

```
from turtle import *

shape('turtle')
shapesize(2)
bgcolor('darkblue')
color('yellow')

sides = 4
length = 100
angle = 90

for i in range(sides):
    forward(length)
    right(angle)
```

Teiknar programmet framleis ein firkant?

- [ ] No kan du få programmet til å teikne ein trekant berre ved å endre verdiane på variablane dine. Prøv om du får det til sjølv.

- [ ] Me kan gjere programmet endå smartare. I staden for at du sjølv må rekne ut vinkelen kan programmet gjere det. Byt ut linja `angle = 90` med

```
angle = 360 / sides
```

No kan du prøve å berre endre verdien av `sides` og køyre programmet på nytt. Teiknar programmet dei riktige mangekantane?

Steg 5: Eigne kommandoar

I Python kan me lage våre eigne kommandoar ved å definere funksjonar. Det er ein annan måte å unngå å gjenta oss sjølv på.

Sjekkliste

- [ ] No skal me lage ein funksjon som teiknar ein mangekant. Det gjer me ved å bruke kommandoen `def` (def er ei forkorting for *define* som tyder definer). Endre programmet ditt så det ser ut som under:

```
from turtle import *

shape('turtle')
shapesize(2)
bgcolor('darkblue')
color('yellow')

def polygon(sides, length):
    angle = 360 / sides

    for i in range(sides):
        forward(length)
        right(angle)

polygon(4, 100)
```

Køyr programmet. Kjenner du att firkanten?

- [ ] No som me har laga `polygon` -funksjonen er det kjempelett å teikne ulike mangekantar. Du kan til dømes leggje til dei følgjande linjene nedst i programmet ditt:

```
polygon(3, 100)
polygon(4, 100)
polygon(5, 100)
forward(125)
right(180)
polygon(3, 150)
polygon(5, 150)
polygon(7, 150)
```

Steg 6: Skjelpaddekunst

Til slutt vil me generalisere funksjonen vår litt slik at den ikkje berre teiknar keisame mangekantar.

Sjekkliste

- [ ] No lagar me ein ny funksjon `polylines` som liknar på `polygon` . Men her skal me kunne endre på vinklane slik at dei ikkje alltid summerast til 360. Det gjer underverk for kunsten vår! Endre programmet ditt så det ser slik ut:

```
from turtle import *

shape('turtle')
shapesize(2)
bgcolor('darkblue')
color('yellow')

def polylines(sides, length, angle):
    for i in range(sides):
        forward(length)
        right(angle)

polylines(5, 100, 144)
```

Køyr programmet. Kva teiknar skjelpadda no?

- [ ] Ein annan variant kan vere at me teiknar ein skeiv mangekant. Til dømes vil den følgjande funksjonen teikne firkantar der vinklane er 91 grader i staden for 90 grader. Det blir overraskande stilig. Byt ut `polylines` -kommandoen med

```
polylines(91, 200, 91)
```

- [ ] Me kan òg endre lengda for strekane etter kvart som me teiknar. Det skapar ein fin spiraleffekt. Legg merke til kor mykje funksjonen `spiral` liknar på `polylines` :

```
from turtle import *

shape('turtle')
shapesize(2)
bgcolor('darkblue')
color('yellow')

def polylines(sides, length, angle):
    for i in range(sides):
        forward(length)
        right(angle)

def spiral(sides, length, angle):
    for i in range(sides):
        forward(length)
        right(angle)
        length = length + 5

spiral(100, 5, 125)
```

- [ ] Prøv andre verdiar enn `100, 5, 125` når du kallar `spiral`. Finn du nokre verdiar som gir spesielt fine bilete?

Prøv sjølv

Kombiner dei ulike funksjonane me har laga, `polygon`, `polylines` og `spiral`, med dei andre skjelpadde-kommandoane du har lært, til dømes `forward` og `left`. Klarar du å teikne meir spanande kunstverk? Eller kanskje du kan teikne ein by? Eit hus kan vere ein firkant med ein trekant på toppen.

Eit tips heilt på slutten er at funksjonane `penup()` og `pendown()` styrer om skjelpadda teiknar når den flyttar på seg. Desse er veldig nyttige når ein vil teikne fleire figurar som ikkje heng saman.

Eit anna tips er funksjonen `speed()`. Denne justerer hastigheita skjelpadda teiknar med. Til dømes vil `speed(1)` teikne veldig sakte, medan `speed(11)` teiknar kjempefort.