△ Kantar, kantar, mange mangekantar

PÅ BOKMÅL | LAST NED PDF

Introduksjon:

Her skal me sjå på litt meir avansert oppteikning og rørsle. Me skal ta utgangspunkt i oppgåva om den sprettande ballen, men bytte ut ballen med trekantar, firkantar og mangekantar. Difor anbefalar me at du har gjort den oppgåva fyrst, eller i det minste at du kjenner til `if`-setningar og koordinatsystemet. Du skal altså lære å teikne former med kantar, mange kantar.

Steg 1: Enkle firkantar

Me startar med rektanglar: dei firkantane det er enklast å teikne på datamaskina.

Sjekkliste

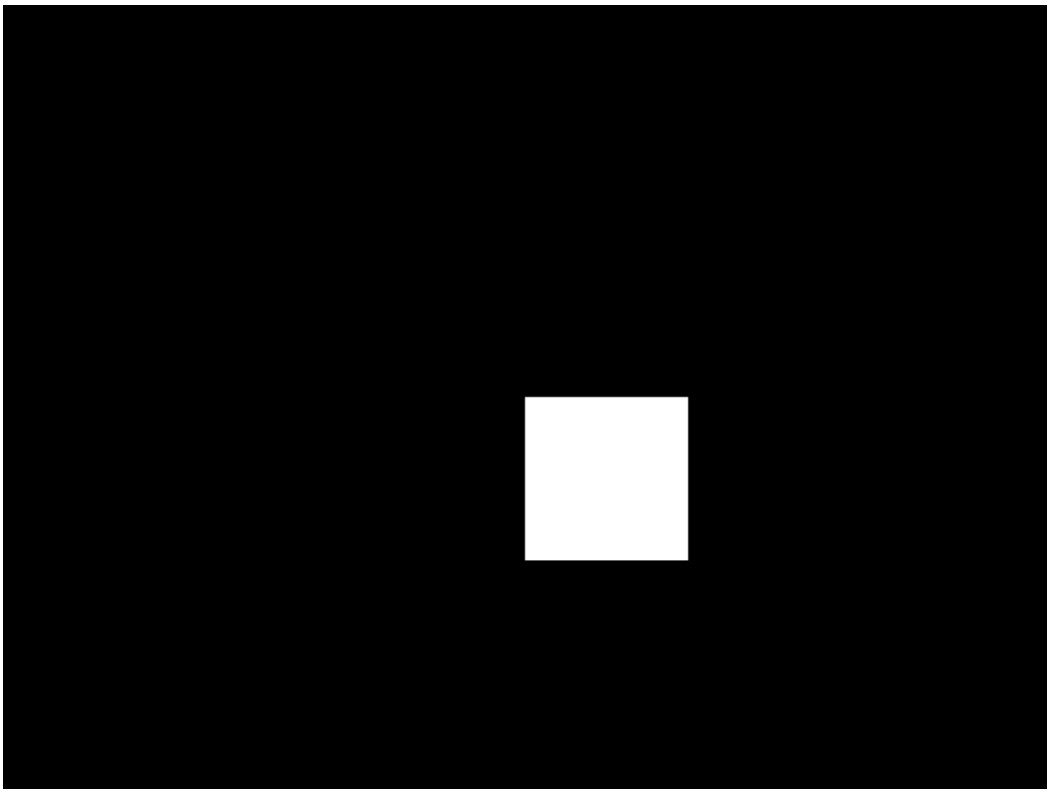
- ☐ Start Processing og skriv dette:

```
float x;
float y;
float xFart = 1.5;
float yFart = 2;

void setup() {
  size(640, 480);
  x = width / 2;
  y = height / 2;
}

void draw() {
  x += xFart;
  y += yFart;

  if (x < 0) {
    xFart = -xFart;
  }

  if (x > width - 100) {
    xFart = -xFart;
  }

  if (y < 0) {
    yFart = -yFart;
  }

  if (y > height - 100) {
    yFart = -yFart;
  }

  background(0);
  rect(x, y, 100, 100);
}
```

Dette programmet er ganske likt det me laga i siste steg i oppgåva om den sprettande ballen, men det er nokre små skilnader:

- Me har endra tala som brukast i `if`-setningane. Kvifor trur du me har gjort det? Kva skjer viss du teiknar ein disk med same posisjon og storleik som firkanten?
- ☐ Me har tatt i bruk `+=`. Når me skriv `x += 1;` er det det same som `x = x + 1;`, det er berre ei slags forkorting. Altså tyder det at me vil auke `x` med det som står på høgre side av `+=`.

- [ ] Køyr programmet ved å trykkje på `ctrl + R` eller knappen 



- [ ] Lagre programmet som Firkant ved å trykkje på `ctrl + S` eller vel `File --> Save` i menyen.

Utfordringar

- [ ] Kan du lage eit rektangel som ikkje er kvadratisk, altså der breidda og høgda er ulike? Hugs at me vil den skal sprette i det den treff kanten av vindauget.

Enkle trekantar

Å teikne rektanglar er nesten heilt likt som å teikne diskar, men no skal du lære å teikne trekantar. Viss ein trekant vart teikna med posisjon, breidde og høgde, så hadde du ikkje hatt så god kontroll over korleis trekanten såg ut. Difor må me seie kor kvart hjørne befinn seg.

Sjekkliste

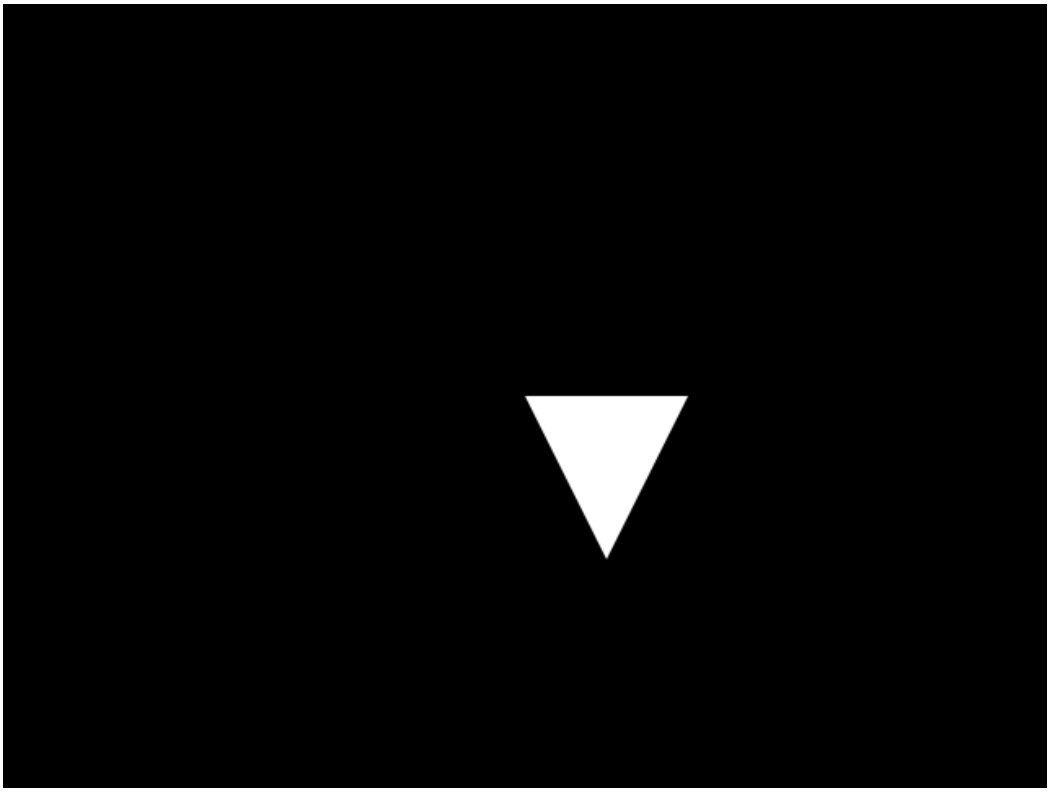
- [ ] No skal me bytte ut firkanten med ein enkel trekant. Endre `draw` som me har vist under:

```
void draw() {
  x += xFart;
  y += yFart;

  if (x < 0) {
    xFart = -xFart;
  }

  if (x > width - 100) {
    xFart = -xFart;
  }

  if (y < 0) {
    yFart = -yFart;
  }

  if (y > height - 100) {
    yFart = -yFart;
  }
```

```
  background(0);
  triangle(x, y, x + 100, y, x + 50, y + 100);
}
```

Her har me tatt i bruk `triangle` i staden for `rect`. Den tek imot seks argument, to for kvart hjørne i trekanten. Me brukar `(x, y)` som posisjonen til dei fyrste hjørnet øvst til venstre, `(x + 100, y)` er posisjonen til det øvste høgre hjørnet, og `(x + 50, y + 100)` er det siste hjørnet nede i midten.

- [ ] Lagre programmet som Trekant ved å velge `File -> Save as` eller trykke `shift + ctrl + S`.
- [ ] Køyr programmet.



Forbetre leselegheita

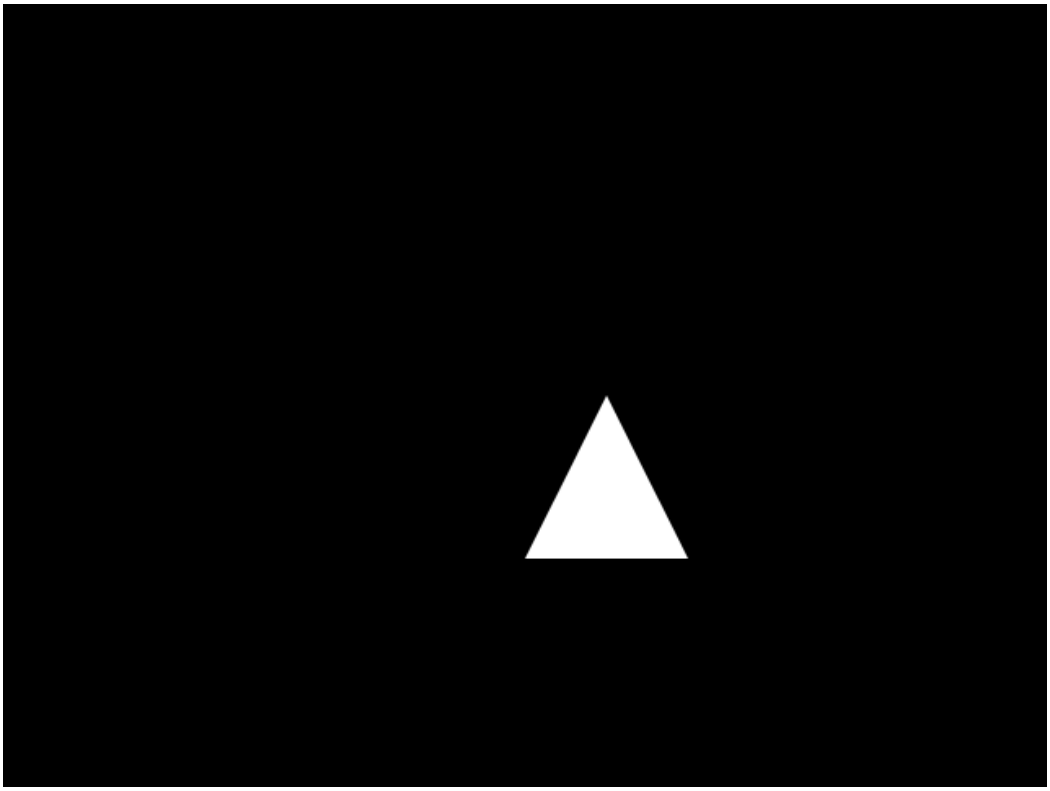
Nokre gonger kan det vere vanskeleg å lese kode med kall på funksjonar som tek mange argument. I Processing tek dei fleste funksjonane berre imot nokre få argument, men `triangle` tek seks. Då kan det vere nyttig å dele opp kallet over fleire linjer. Til dømes kunne setninga over vore skrive slik at kvart hjørne var på kvar si linje:

```
triangle(x, y,
  x + 100, y,
  x + 50, y + 100);
```

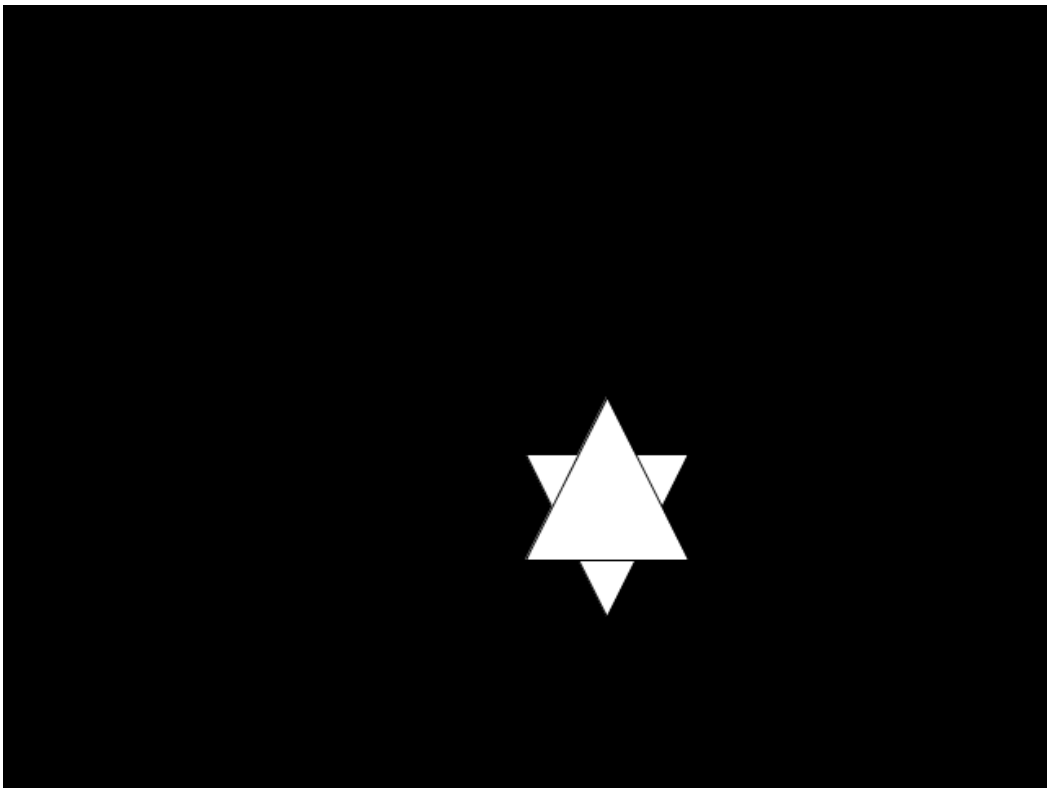
Viss ein framleis synest det er vanskeleg å lese eller ser rotete ut, så kan ein leggje til nokre ekstra mellomrom for å få ting på linje. Merk at om ein brukar automatisk formatering av koden i Processing, så vil den fjerne mellomrom der den meiner det er overflødig.

Utfordringar

- [ ] Kan du teikne trekanten motsett veg slik at det ser ut som ei pil som peikar oppover i staden for nedover?



- [ ] Kan du teikne to trekantar i staden for ein, og lage ei sekskanta stjerne?



- [ ] **Vanskeleg:** Trekanten i programmet er nesten likesida, men den er litt for høg, så to av sidene er rundt 12 pikslar for lange. Kan du endre reknestykket `y + 100` slik at trekanten blir likesida? Du kan bruke Pytagoras si læresetning, eller sinus-funksjonen, `sin` i Processing, for å finne den riktige høgda. Funksjonen `radians` kan vere til hjelp for å gjere om gradar til radianar viss du vil bruke sinus-funksjonen.

Trekantar

No skal me sjå korleis me kan lage trekantar der kvart hjørne beveger seg for seg sjølv. Då treng me variablar for posisjon og fart for kvart hjørne. Til saman blir det fire variablar for kvart hjørne i trekanten. Ein for `x`-posisjon, ein for `y`-posisjon, ein for `x`-fart og ein for `y`-fart. Sidan trekanten har tre hjørner, så blir det totalt `3 hjørner * 4 variablar per hjørne = 12 variablar`.

Me kunne til dømes kalla dei `x1`, `x2` og `x3`, og tilsvarande lagt til tal på `y`, `xFart` og `yFart`. I staden skal me bruke noko som kallast eit *array*. Me brukar ofte det engelske ordet på norsk, men nokre gonger omset me det til liste, vektor, rekke, tabell eller matrise.

Sjekkliste

- [ ] Me startar med å endre variablane til *arrays*:

```
float[] x = new float[3];
float[] y = new float[3];
float[] xFart = new float[3];
float[] yFart = new float[3];
```

No har me endra typen av variablane frå `float` til `float[]`. Når me set klammeparentesar etter ein type, så er det eit *array* som inneheldt verdiar av typen framfor klammene. Bak likskapsteiknet ser me òg noko nytt. Koden `new float[3]` tyder at me skal lage eit nytt `float`-*array* med tre tal i.

- [ ] No må me endre startverdiane til desse tala, elles vil dei alle berre vere `0`:

```
void setup() {
  size(800, 600);

  x[0] = width / 2;
  x[1] = width / 2;
  x[2] = width / 2;

  y[0] = height / 2;
  y[1] = height / 2;
  y[2] = height / 2;

  xFart[0] = 1.5;
  xFart[1] = 2.5;
  xFart[2] = 3.5;

  yFart[0] = -5;
  yFart[1] = 2.5;
  yFart[2] = -1.5;
}
```

Her ser me korleis me jobbar med verdiane i eit *array*. Me brukar klammeparentesar med tal i for å seie kva verdi me skal jobbe med. Den fyrste verdien står på plass `0`, og den siste verdien er på plass `2`. Talet for plasseringa kallast ein *indeks*. Indeksen er alltid lågare enn om me hadde telt vanleg, sidan me startar på `0`. Det er forklaringa på at den siste indeksen er ein lågare enn storleiken me sette i stad.

- [ ] Til slutt må me flytte rundt på hjørnene og teikne opp trekanten vår:

```
void draw() {
  for (int i = 0; i < x.length; i++) {
    x[i] += xFart[i];
    y[i] += yFart[i];

    if (x[i] < 0) {
      xFart[i] = -xFart[i];
    }

    if (x[i] > width) {
      xFart[i] = -xFart[i];
    }

    if (y[i] < 0) {
      yFart[i] = -yFart[i];
    }

    if (y[i] > height) {
      yFart[i] = -yFart[i];
    }
  }

  background(0);
  triangle(x[0], y[0], x[1], y[1], x[2], y[2]);
}
```

Her ser du ein heilt ny konstruksjon som me skal sjå nærare på i forklaringa under. Før me går vidare kan du teste programmet ditt. Lagre og køyr det.

Forklaring

I starten av `draw` har me lagt inn noko som kallast ei løkke (*loop* på engelsk). Ei løkke er ein del med kode som skal utførast fleire gonger. Det finst andre slags løkker, og den me har brukt her kallast ei *for-løkke*. Inne i parentesane etter `for` har me tre setningar. Den fyrste, `int i = 0` blir utført før løkka. Den neste, `i < x.length` bestemmer om koden i løkka skal utførast eller om løkka er ferdig. Den siste, `i++` blir utført etter koden mellom krøllparentesane, altså innhaldet i løkka. Me brukar `i` inne i løkka som indeks når me jobbar med arraya i staden for å skrive faste tal.

Viss me går gjennom koden steg for steg, ser me at me fyrst lagar ein variabel `i` av typen `int` som startar med verdien `0`. Forkortinga `int` står for *integer*, altså heiltal, og er typen me brukar for variablar som er nettopp heiltal. Så sjekkar me om `i` er mindre enn storleiken til arrayet `x`. Viss den er det (og no er den det, fordi storleiken til `x` er 3, og `i` er framleis `0`), så køyrast koden mellom krøllparentesane. Når all koden mellom krøllparentesane er køyrt, så køyrast `i++`. Denne gjer det same som `i = i + 1`, altså aukar me `i` med `1`. Så sjekkar me igjen om `i` er mindre enn storleiken til `x`. Slik fortset det heilt til `i` blir like stor som eller større enn storleiken til `x`.

Slike løkker som me har brukt her er veldig vanleg, og blir brukt til å jobbe med array. Du kjem til å sjå mange slike framover. Løkker kan kreve litt øving før ein blir god på det, men etter kvart kjem du til å bli veldig glad for å sleppe å skrive den same koden mange gonger.

Utfordringar

- ☐ Det går an å lage firkantar der ein plasserer kvart hjørne for seg, slik som med den siste trekanten. Då brukar ein funksjonen `quad` i staden for `rect`. Prøv å endre programmet til å lage ein firkant med hjørner som sprett rundt på skjermen. Kor mange fleire variablar treng du samanlikna med det me måtte ha for trekanten? Kor mange parametrar tek `quad`?

Mangekantar

No skal me sjå på korleis me kan lage mangekantar. Mangekantar er rett og slett eit generelt namn for ei form med fleire kantar, slik som trekantar, firkantar, femkantar og så bortetter.

Sjekkliste

- [ ] Me startar med å endre storleiken på *arraya* i førre steg:

  ```
  int KANTAR = 5;
  float[] x = new float[KANTAR];
  float[] y = new float[KANTAR];
  float[] xFart = new float[KANTAR];
  float[] yFart = new float[KANTAR];
  ```

  No brukar me ein variabel for å setje storleiken i staden. Dette gjer koden enklare å lese, og det er mykje enklare å endre koden, sidan me berre treng å endre talet ein stad.

- [ ] Posisjonane og hastigheitene til hjørna vart sett til faste verdiar, men viss me ikkje veit nøyaktig kor mange kantar det er fungerer ikkje det så bra. Difor endrar me på `setup` til å bruke ei løkke for å setje startverdiane:

  ```
  void setup() {
    size(800, 600);

    for (int i = 0; i < KANTAR; i++) {
      x[i] = random(width);
      y[i] = random(height);
      xFart[i] = random(-5, 5);
      yFart[i] = random(-5, 5);
    }
  }
  ```

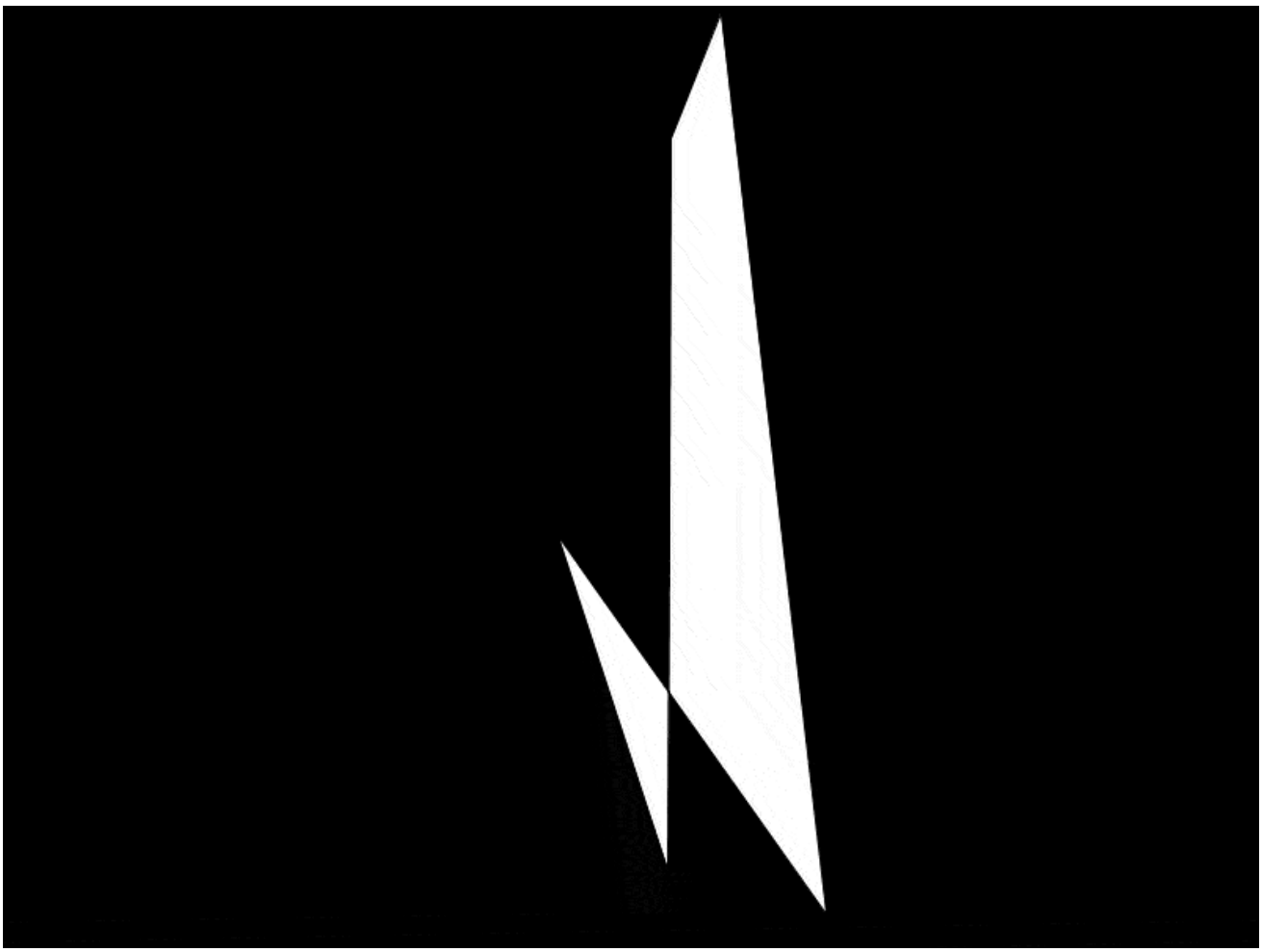
  Denne løkka liknar ein del på den me allereie har i `draw`. Men me har introdusert ein funksjon som heiter `random`. Denne gir oss tilfeldige tal. Viss me kallar den utan verdiar, så får me eit tal mellom `0` og `1`. Viss me kallar den med éin verdi, `random(width)`, så får me eit tal mellom `0` og verdien. Viss me brukar to verdiar, `random(-5, 5)`, så får me eit tal mellom dei to verdiane.

- [ ] No skal me teikne mangekanten vår. Me treng ikkje endre på den fyrste løkka i `draw`, men me bør endre testen slik at den liknar den over. Me skal bytte ut kallet på `triangle` med ei løkke som teiknar kvar kant:

  ```
  void draw() {
    for (int i = 0; i < KANTAR; i++) {
      x[i] += xFart[i];
      y[i] += yFart[i];

      if (x[i] < 0) {
        xFart[i] = -xFart[i];
      }

      if (x[i] > width) {
        xFart[i] = -xFart[i];
      }

      if (y[i] < 0) {
        yFart[i] = -yFart[i];
      }

      if (y[i] > height) {
        yFart[i] = -yFart[i];
      }
    }

    background(0);

    beginShape();
    for (int i = 0; i < KANTAR; i++) {
      vertex(x[i], y[i]);
    }
    endShape(CLOSE);
  }
  ```

  Her ser me tre nye funksjonar: `beginShape`, `vertex` og `endShape`. Funksjonen `beginShape` angir at me skal teikne ei form. Me brukar `vertex` for å leggje til eit hjørne i forma. Den tek inn to verdiar for posisjonen til hjørnet. Til slutt har me `endShape` som

seier at forma er ferdig og klar til å bli teikna på skjermen. Viss me kallar `endShape` utan `CLOSE` blir ikkje forma lukka og fylt.



Utfordringar

- ☐ Kan du bruke `random` til å få hjørna til å endre hastigheit når dei treff kanten av vindauget?

  Pass på, viss farta blir lågare enn den var, så kan hjørnet blir ståande fast i kanten av vindauget. Det er fordi me eigentleg let hjørnet bevege seg litt utanfor vindauget før det snur. Set `x` eller `y` til å vere lik posisjonen til vindaugekanten inne i `if`-setningane for å unngå det.