S. Koch Stigberg, M. Maugesten & H. Stigberg

Use–modify–create

Ett arbetssätt för programmering

Att knyta samman programmering och matematik är en utmaning för matematiklärare. Författarna beskriver en arbetsmetod som hjälper lärare att undervisa programmering tätt kopplat till ett matematiskt innehåll. Det utprovade upplägget ingår i en fortbildningsinsats för matematiklärare i Norge.

I Sverige blev programmering infört som en del av matematikämnet 2018 och i Norge infördes en ny läroplan 2020 där programmering är en del i matematikämnet. Skillnaden mellan svensk och norsk läroplan i matematik är bland annat att de norska kompetensmålen för programmering är något mer explicit knutna till det matematiska innehållet. Norge har specificerade kompetensmål för varje årskurs där ett är direkt kopplat till programmering. En annan skillnad är att i den norska läroplanen nämns inte något om olika programmeringsmiljöer.

Kompetensmål för varje årskurs

I Norge börjar sexåringarna i 1. trinn, vilket gör att 2. trinn–10. trinn motsvarar årskurs 1–9 i Sverige. I den norska läroplanen kan vi läsa:

Mål for opplæringen er at eleven skal kunne:

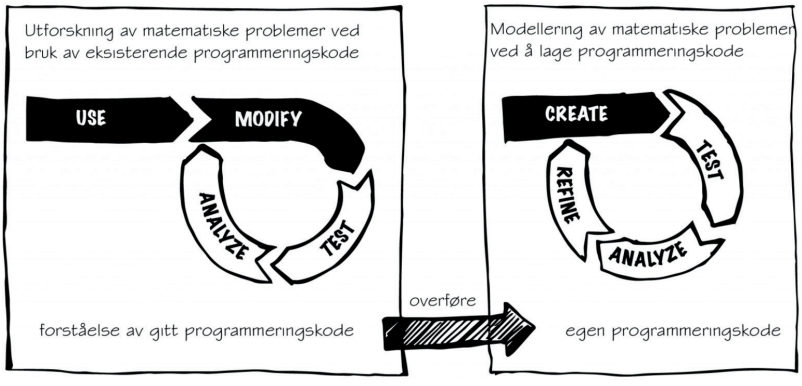
2. trinn:	lage og følge regler og trinnvise instruksjoner i lek og spill
3. trinn:	lage og følge regler og trinnvise instruksjoner i lek og spill knyttet til koordinatsystemet
4. trinn:	lage algoritmer og uttrykke dem ved bruk av variabler, vilkår og løkker
5. trinn:	lage og programmere algoritmer med bruk av variabler, vilkår og løkker
6. trinn:	bruke variabler, løkker, vilkår og funksjoner i programmering til å utforske geometriske figurer og mønstre
7. trinn:	bruke programmering til å utforske data i tabeller og datasett
8. trinn:	utforske hvordan algoritmer kan skapes, testes og forbedres ved hjelp av programmering
9. trinn:	simulere utfall i tilfeldige forsøk og beregne sannsynligheten for at noe skal inntreffe, ved å bruke programmering
10. trinn:	utforske matematiske egenskaper og sammenhenger ved å bruke programmering

Vårt utvecklingsprojekt

År 2020 genomförde vi ett forskningsprojekt vid Høgskolen i Østfold, där vi undersökte vilka resurser som behövs för att stötta lärare med liten eller ingen tidigare programmeringskunskap att implementera den nya läroplanen i matematik. Projektet genomfördes av oss tre, två från högskolans lärarutbildning och en från IT-avdelningen. Vi planerade och genomförde fem heldagsworkshops under ett års tid med nio högstadielärare från tre olika norska skolor. Vårt övergripande mål var att undersöka hur programmering kan stötta matematikämnet och det inbegriper också att undersöka hur man kan stötta matematiklärare med ringa erfarenhet av programmering. Vi tog utgångspunkt i kompetensmålen och gjorde vår tolkning av vad de skulle kunna innebära. Vår övergripande tanke var att ge de deltagande lärarna en överblick över vilka verktyg som skulle kunna användas men också att arbeta med uppgifter som vi ansåg användbara tillsammans med elever på (motsvarande) högstadiet. Vi utgick från målen i de tidigare årskurserna och arbetade först utan digitala hjälpmedel, för att sedan använda en programmerbar robot (mBot). Därefter fick lärarna arbeta med Excel och Scratch och avslutningsvis med Python. Under varje workshop fick lärarna input från oss, de fick själva pröva ut uppgifter och slutligen gavs tid för planering och förberedelse för utprövning i eget klassrum. Varje kommande workshop startade med en gruppintervju för att dela erfarenheter kring utprövningarna.

Ett välfungerade arbetssätt

Vi vill framförallt berätta om arbetssättet *use–modify–create*, som finns beskrivet i artikeln *Computational thinking for youth in practice*, och som vi upplevde fungerade bra och är stöttande för både lärare och elever. Det ger också utrymme för ett undersökande och anpassat arbetssätt. Modellen går ut på att eleverna i första fasen, *use*, får använda ett fungerande program, undersöka vilka funktioner det har och fundera på hur det kan tänkas vara uppbyggt. Nästa fas, *modify*, handlar om att eleverna går in och läser programkoden för att därefter ändra delar av den. Beroende på vad eleverna vill förändra eller hur mycket eleverna förstår så kan de ändra olika stora delar i koden. Denna fas ger också utrymme för att pröva sig fram. I sista delen, *create*, ska eleverna använda sin kunskap och programmera ett liknande program. Alla delar i denna modell är iterativa och samarbete uppmuntras. Det finns heller inte alltid någon helt tydlig gräns mellan de olika faserna. Vi har samlat information kring projektet i form av förslag till upplägg och en film där två av oss samtalar om arbetsmetoden, se länk i slutet av artikeln.



Modell över arbetssättet use–modify–create.

Tärningskast: ett tema, två programspråk

Vi fokuserade på kompetensmålet ”simulera resultat i slumpmässiga försök och beräkna sannolikheten att något kommer att inträffa, genom användning av programmering” (vår översättning).

Eftersom lärarna hade tidigare erfarenhet av att använda tärningar i sin undervisning om sannolikhet ville vi knyta an till den erfarenheten när vi planerade. Tärningar är visuella och vi tänkte att tröskeln skulle bli lägre om vi gjorde samma sak med programmering som man gör med fysiska tärningar. Det var dock viktigt för oss att lyfta fram vilka fördelar som finns med att arbeta med programmering. Till exempel att man kan ändra antalet sidor på tärningarna till vilket antal som helst och att man kan kasta väldigt många kast utan att det tar någon längre tid. Vi gjorde detta upplägg i Excel också men vi vill framför allt beskriva det vi gjorde i Scratch och Python.

Uppläggen med Scratch och Python finns i länk i slutet av artikeln.

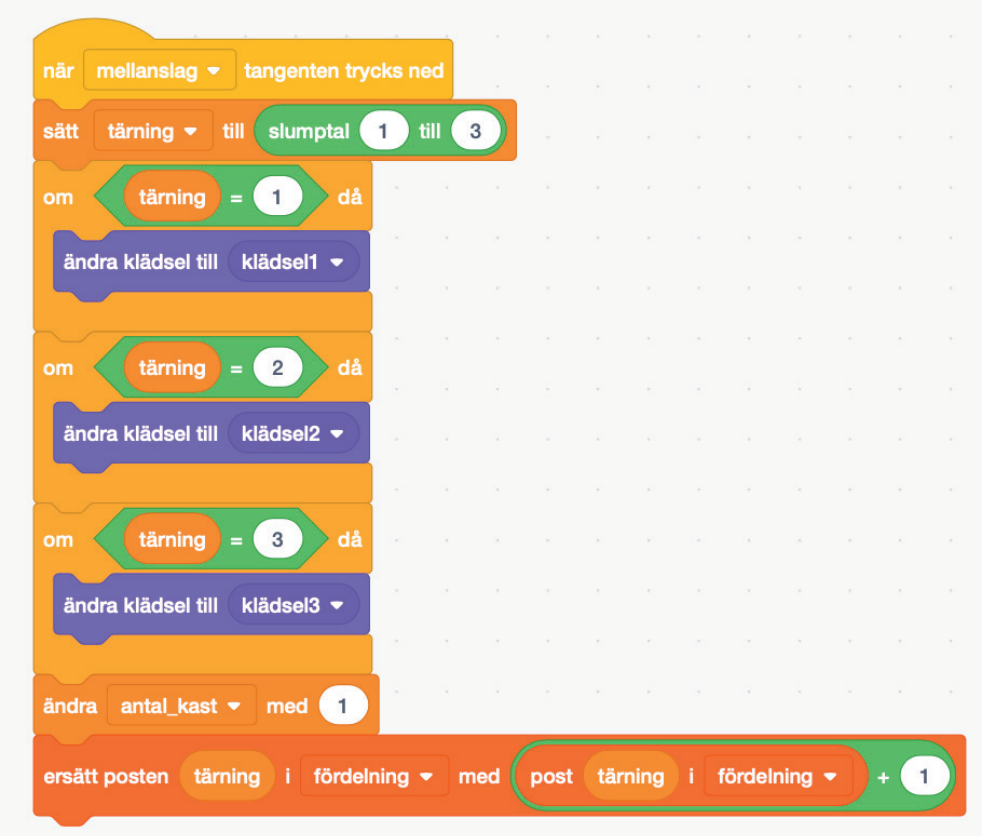
Scratch

Vårt exempel i Scratch handlar om tärningskast och eleverna får som första uppgift använda, *use*, programmet. Det de kan se är att antal ögon fördelar sig (troligtvis) ganska lika, särskilt ju fler kast som görs. Detta kan ge underlag till diskussion om de stora talens lag och sannolikhetsfördelning. Kanske kan man komma in på om det är möjligt med en fysisk tärning som har tre sidor. Förhoppningsvis leder detta till att eleverna blir nyfikna på programkoden bakom programmet och går in på ”se inuti”.

Exempel på use-fasen. Kast med tärning med tre sidor i Scratch.

I nästa fas, *modify*, får eleverna i uppgift att läsa, försöka förstå koden och förändra den så att, till exempel, tärningen får sex sidor. Här ges möjlighet för eleverna att pröva sig fram och se vilket resultat en viss förändring ger. Kanske sker något helt annat än det som de hade tänkt sig eller så fungerar ingenting längre. Vi uppmanar alla att vara systematiska i sin utprövning men det finns alltid möjlighet att börja om från början eller att ångra en förändring. Det viktiga är att eleverna inte ska känna en oro att ”förstöra” något utan känna utforskarglädje.

Som sista steg, *create*, ska eleverna programmera ett eget, liknande program. Här är det viktigt med elevernas egna förslag och önskemål. Vi har gett några förslag på program som eleverna kan programmera och för oss har det varit viktigt att inte tappa det matematiska innehållet i programmet. Vi föreslår att eleverna kan göra program som kastar väldigt många kast samtidigt eller att de tittar på summan av antalet ögon vid två kast och hur dessa fördelar sig.



Exempelkod i Scratch att förändra.

Lärarens roll är mycket viktig för att knyta samman programmering med det matematiska innehållet. På samma sätt som med konkret material i matematikundervisningen står inte programmet för en inneboende kunskap som eleverna kan tillägna sig. Detta exempel i Scratch kan fungera både som underlag för utforskning och diskussion, men läraren måste stötta eleverna i att finna sammanhang mellan programmering och matematik, till exempel genom att ha en diskussion om vilka slutsatser eleverna har dragit, hur stor sannolikheten är att få tre ettor i rad, eller något annat som läraren har som mål i sin undervisning.

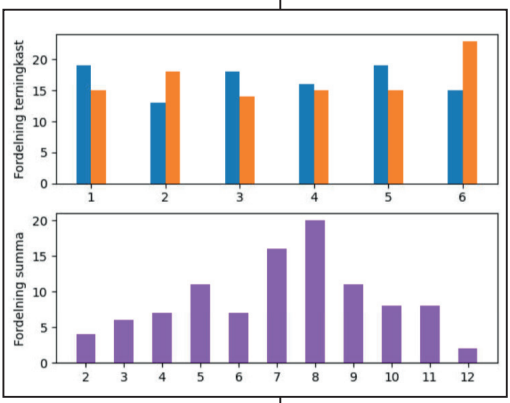
Python

Även exemplet i Python handlar om tärningskast och på samma sätt så uppmuntras först eleverna att använda, *use*, programmet för att sedan förändra, *modify*. Detta exempel är lite mer avancerat än det föregående i Scratch, dels för att programmet skriver ut fördelningen för varje tärning och fördelningen av de olika summorna vid kast med två tärningar, dels för att textprogrammering också kräver en korrekt syntax.

Som första steg är tanken att eleverna får se att fördelningen med många kast för varje tärning är ganska jämn. Det blir ungefär lika många ettor som tvåor. Detta gäller inte för summan av två tärningar. Hur kan det komma sig?

Vi önskar att detta ska väcka nyfikenhet och leda till matematiska diskussioner som i sin tur gör att man kan komma in på den teoretiska modellen.

```
import os
os.environ['MPLCONFIGDIR'] = os.getcwd() + "/configs/"

#importera allt som behövs
import random as rand
import time as time
import numpy as np
import matplotlib
#matplotlib.use('Agg')

import matplotlib.pyplot as plt

##### start programkod #####
antal = int(input('Hvor mange kast? '))

# antal terningsidor
sidor = 6

freqtabell_terning1 = {}
freqtabell_terning2 = {}
freqtabell_summa = {}

#lista över möjliga terningsidor {1,2,....,sidor}
terningsidor = range(1, sidor + 1)
#lista över möjliga summor för to terningar {2,3,4,....,(2*sidor)}
summor = range(2, sidor * 2 + 1)

#init frekvenstabell summa till 2:0, 3:0, 4:0 ...., 12:0
for k in summor:
    freqtabell_summa[k] = 0

#init frekevenstabell terningar till 1:0, 2:0 ...6:0
for k in terningsidor:
    freqtabell_terning1[k] = 0
    freqtabell_terning2[k] = 0

#kasta terninger
```



Exempelkod i Pyhton för kast med två tärningar som har sex sidor.

I nästa steg, *modify*, utmanade vi deltagarna att förändra koden till att antingen ha flera tärningar, eller ha fler antal sidor på tärningarna och se hur fördelningen av summan blir. Med textprogrammering ser vi både en del där eleverna lär sig syntax – det finns säkert ”ord” som eleverna inte vet vad de betyder och får söka upp – och som en del att börja förändra och se hur detta påverkar programmet. Här upplevde vi att det var svårare att ha fokus på det matematiska innehållet utan fokus hamnade i större utsträckning, jämfört med Scratch, på koden.

Våra erfarenheter

I våra workshops hade vi en liten introduktion till temat och det program som lärarna skulle jobba med. Vi uppmanade alla att sätta sig in i programmet, se vilka funktioner som finns och därefter diskutera och skriva ner en plan för vad de skulle kunna tänka sig att förändra, hur de tror att programmet är uppbyggt och därefter gå in och se på koden. När lärarna sedan satte igång så började nästan alla, efter att mycket kort sett på programmet, att gå in i koden och pröva sig fram och därigenom förstå programmets uppbyggnad.

Vi hade tänkt att få till diskussion och reflektion kring programkoden innan de gick in i fasen *modify*, men lärarna upplevde att de ville se koden först och ha det som diskussionsunderlag. Vi ställer oss genuint frågande om vi tänkte fel med att en djupare reflektion skulle komma innan man ser på koden eller om koden ska fungera som stöd till reflektion.

Lärarna rapporterar att denna metod hjälpte dem att få igång matematiska samtal i sina klasser. Eleverna diskuterade vad de olika kodavsnitten betydde och varför programmet fungerade som det gjorde. Till exempel så sa en lärare att eleverna diskuterade vad i koden som måste ändras för att variera antalet sidor på tärningen. Hon berättade också om hur de kom in på de stora talens lag efter att ha jobbat med programmering med tärningskast och att programmeringsuppgifterna låg till grund för att ”höja kvaliteten på samtalen”. Som med mycket annat i matematikundervisning står inte heller ett program för en inneboende förmåga att hjälp eleverna att tillägna sig kunskap utan lärarens roll är otroligt viktig. Denna typ av programmering kan stödja läraren att lyfta elevernas samtal och koppla samman koden med det matematiska innehållet.

Andra reflektioner vi har fått från lärarna efter att de har prövat ut metoden i klassrummet är att den var särskilt viktig när de jobbade med textprogrammering. Att eleverna skulle läsa och förändra koden sänkte tröskeln betydligt för att använda relativt avancerade program som var kopplade till ett matematiskt innehåll, till exempel sannolikhetslära. Någon nämnde att de aldrig skulle kunna jobba med programmering av denna typ om de inte hade fått programkoden på förhand av oss för att sedan ta med in i sin undervisning.

Flera tyckte att metoden hjälpte dem att dels individanpassa stödet till enskilda elever, dels att de själva kunde anpassa materialet till elevgruppen. Vi hade till exempel en uppgift med att rita ut geometriska figurer med Python. Denna uppgift kunde lärarna anpassa genom att välja svårighetsgrad på figurerna, hur de upprepades och antalet figurer. Mängden kod var en viktig faktor för om vissa elever ens skulle börja att sätta sig in i den. När eleverna skulle arbeta med koden kunde de välja att börja jobba med de delarna som de förstod bäst.

Det som inte gick som vi hade planerat från början var att både vi och lärarna bara tog de två första stegen i modellen *use–modify–create*. Vi prioriterade att arbeta med de två första stegen eftersom många av lärarna inte hade så stor erfarenhet av programmering. Även lärarna rapporterade att detta fungerade bäst för dem då en del kände sig stressade av att få med programmering i sin undervisning. Vi argumenterar nu för att *use* och *modify* kan vara tillräckligt i början då vi inte ska utbilda programmerare utan demokratiska medborgare som ska ha en förståelse för hur den digitala världen är uppbyggd och vi är medvetna om att *create* är den mest utmanande fasen.

Det finns fortfarande mycket kvar att lära hur man på bästa sätt knyter ihop matematik och programmering, men vi hoppas att våra exempel med *use–modify–create* kan vara ett litet bidrag.

LITTERATUR OCH LÄNKAR

Lee, I., Martin, F., Denner, J., Coulter, B., Allan, W., Erickson, J., Malyn-Smith, J. & Werner, L. (2011). Computational thinking for youth in practice. *Acm Inroads*, 2(1), 32–37.

blogg.hiof.no/maprog
scratch.mit.edu/studios/25998548/projects
repl.it/@SusanneStigberg/maprogterning#main.py