Programmering i matematik

Ola Helenius, NCM och Morten Misfeldt, Aalborg universitet

Matematik är ett speciellt ämne eftersom det å ena sidan består av sina objekt, som fem, cirklar, rationella tal, polynom och fyrdimensionella kuber, och å andra sidan är ett utövande. När man i kursplanen har valt att skriva fram förmågor som syftet med matematikundervisningen, så är det ett utslag av att lyfta fram matematiken som en handling, det vill säga det som människor gör när de är matematiska; de löser problem, utvecklar och använder olika matematiska begrepp, representationer och rutiner. Ibland säger man att matematiken är *autopoietisk*, självgenererande. Det är genom att vara matematisk som man skapar matematik, och man är matematisk när man sysslar med matematiska begrepp. Programmering kan anknytas till båda sidorna av matematiken.

Vi kan ställa flera frågor som är matematiska till sin natur men som också har en didaktisk sida.

1. Vilka matematiska problem kan man lösa med programmering?
   Vilka är lämpliga att arbeta med ur ett undervisningsperspektiv, och varför?
   Hur ska det i så fall gå till?
2. Vilka matematiska objekt kan programmeras? Eller mer exakt: vilka matematiska objekt kan representeras med eller via programmering?
   Vilka är lämpliga att arbeta med ur ett undervisningsperspektiv, och varför?
   Hur ska det i så fall gå till?
3. Vilka tekniska och matematiska kunskaper och resurser behövs för att arbeta med dessa problem och objekt (punkt 1 och 2)?
4. Har lösningar till matematiska problem som baseras på programmering andra egenskaper än lösningar som inte baseras på programmering, och vad kan i så fall skillnaden vara?

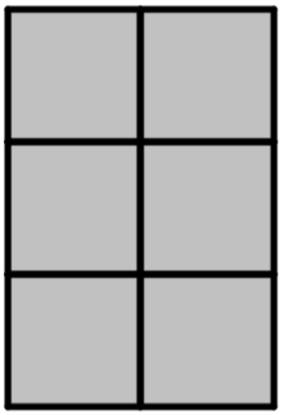
De flesta programmeringsuppgifter så här långt i modulen har handlat om att programmera något som i sig inte är ett matematisk problem, men där matematik på olika sätt används i arbetet eller för att representera till exempel en rörelseriktning eller en position. I denna sista del i modulen ska vi fokusera på att programmera *i matematiken*. Även om det inte finns några generella svar på frågorna ovan, så är det just sådana frågor som hjälper oss att förstå de didaktiska dilemman som uppkommer när man arbetar med programmering i matematiken.

Tre exempel

För att kunna föra en mer generell diskussion om olika användningsområden för programmering i matematiken inleder vi med tre exempel. Alla passar kanske inte att göra i alla årskurser, men syftet är att illustrera olika aspekter av programmering i matematik.

Exempel 1. Primtalstestning

Ett av de viktigaste begreppen inom den multiplikativa sidan av området tal och operationer är primtal. Primtal går att introducera redan när man introducerar multiplikation. Ett mycket användbart sätt att betrakta multiplikation på är som rektanglar med enhetsrutor (figur 1).



Figur 1. En representation av multiplikationen 3 · 2 och 2 · 3.

Denna bild kan då representera 3 · 2 och 2 · 3, den kan ses som 3 tvåor eller 2 treor. Den representerar att 6 kan faktoriseras som 2 · 3. Primtal är sådana heltal (större än 1) som endast har två faktorer, 1 och talet självt. Det betyder att den enda multiplikationsrektangel man kan bygga av ett primtal p, är en 1· p-rektangel. Tal som inte är primtal kan man alltid bygga även på andra sätt, som i fallet med 6 ovan.

Om man vill testa om ett tal n är ett primtal så kan man först testa om det är delbart med 2, sedan med 3, 4, 5 osv till n-1. Egentligen behöver man bara testa med primtal och bara med tal som är mindre än $\sqrt{n}$, men eftersom man inte alltid har tillgång till listor med primtal och eftersom en dator normalt inte knorrar om man ber den att göra lite extrajobb så fungerar den elementära algoritmen som prövar med alla tal upp till n-1. Det är relativt enkelt att programmera en primtalstestare. I figur 2 ser vi ett exempel på hur en sådan skulle kunna se ut, med så kallad pseudokod. Pseudokoden måste sedan skrivas om till kod i ett programmerimgsspråk, till exempel Scratch.

```
Sätt nämnare = 2
Sätt tal = [siffran du vill undersöka]

Medan nämnare är mindre än tal
   Om tal / nämnare = jämt
    Skriv ”Tal är INTE primtal”
   Annars
    Öka nämnare med 1
```

Figur 2. Pseudokod där programmet testar om ett tal är ett primtal.

En möjlig utveckling av programmet är att få det att skriva ut faktorerna för sådana tal som inte är primtal eller till och med att rita upp alla möjliga faktoriseringar som rektangelfigurer. De yngre eleverna kanske inte själva kan konstruera ett sådant program, men genom att arbeta med rutmönstermodeller för multiplikation och resonera om vad som behöver göras för att konstatera om ett tal är ett primtal eller ej, går det att närma sig en algoritmisk/schematisk beskrivning av ett program som det ovan.

Om man inte tidigare har arbetat med rutmönstermodeller för multiplikation eller med primtal, så kan man börja konkret och arbeta med något laborativt material, till exempel multilink. Man provar hur många olika rektangelformer man kan bygga utifrån ett givet antal (*n*) kuber. Därefter kan man ställa frågan hur man kan skapa en systematisk regel, en algoritm bestående av en sekvens av tester, som gör det möjligt att säkert avgöra om ett tal är ett primtal eller ej. Att visa, använda eller utveckla ett program som representerar denna algoritm kan vara ett sista steg.

Exempel 2. Många dubbleringar

Betrakta följande problem:

> *Hur många vikningar måste man gör av ett papper för att det vikta papperet ska vara mer än 5 gånger så tjockt som ett enstaka papper? Mer än 100 gånger så tjockt? Mer än 1000000 gånger så tjockt? Du behöver inte bry dig om att det rent praktiskt inte går att göra hur många gånger vikningar som helst.*

Genom att experimentera med papper och resonera kan man enkelt komma fram till att papperet dubblas i tjocklek för varje vikning. Så en vikning = dubbelt så tjockt. Två vikningar blir 4 gånger så tjockt, tre vikningar blir 8 gånger så tjockt och så vidare. För att det ska bli mer än 5 gånger så tjockt måste man alltså göra 3 vikningar.

Men orkar eleverna räkna ut hur många vikningar som behövs för att det ska bli 100 gånger tjockare? Kanske. Men 1000000 orkar de nog inte. Det går naturligtvis att slå x 2 många gånger på en miniräknare och titta efter när fönstret visar ett tillräckligt stort tal och hålla ordning på antalet gånger man dubblerade. Men det går ju också att systematisera denna algoritm i ett program.

Man kan exempelvis använda kommandot *while* som finns implementerat i alla blockprogrammeringsmiljöer. Då kan vi be programmet att så länge som tjocklek<1000 (till exempel) repetera instruktionerna

```
multiplicera tjocklek med 2 om produkten är mindre än önskat
värde,
 addera 1 till antal vikningar.
```

Eftersom programmet stannar precis innan ”tjockleken” blir det eftersträvade värdet i variabeln *antal vikningar* +1 det tal vi söker.

Exempel 3. Skapa en geometrisk figur

Detta exempel på programmering handlar om att utforska matematiska begrepp i geometri. I tidigare exempel i modulen har vi tittat på hur man i programmeringsmiljöer som Scratch kan flytta figurer på en skärm. Då har utgångspunkten varit figuren och hur man med hjälp av programkod skapar vissa konkreta former eller rörelser, där figuren styrs genom en bana. Men samma program kan också ses som ett sätt att utforska en matematisk idé, till exempel geometriska figurer. Om vi ska styra en figur med hjälp av förflyttningar längs en linje en bestämd sträcka (t.ex. 100 pixlar), och av vridningar (t.ex. 90 grader) så kan vi ställa frågan: Går det, genom att upprepa sekvensen ”*förflyttning+vridning*”, att skapa en kurva som slutar på samma ställe som den började?

Den här frågan är ekvivalent med att fråga efter hur man kan konstruera regelbundna *n*-hörningar. Elever som har viss erfarenhet kanske inser att fyra upprepningar av *”förflyttning 100 pixlar + vridning 90 grader*” skapar en kvadrat. Men går det att välja vridningsvinkeln och antalet upprepningar på andra sätt och skapa en annan kurva som också slutar där den börjar? Om eleverna inte är vana att programmera i miljöer som Scratch eller Blockly kan läraren orkestrera lektionen så att de inledningsvis arbetar med problemen i helklass, där läraren sköter programmeringen och resultatet visas på en skärm som alla ser.

En av de övningar som finns på code.org, se figur 3, liknar den frågeställning som beskrivs ovan. Redan i steg fyra i denna övningssekvens ritar eleverna kvadrater med hjälp av kommandot *loop*, det vill säga upprepar en förflyttning och en vridning fyra gånger.

Figur 3. Övning från https://studio.code.org/s/frozen/stage/1/puzzle/4

De tre exemplens olika karaktär

Det tre exemplen ovan involverar inte bara olika matematiskt innehåll utan är också av olika karaktär.

Skapa ett matematisk verktyg

I primtalsexemplet skapas ett *matematiskt verktyg*. Det handlar ju inte bara om ett speciellt problem som man löser med programmet. Att konstruera programmet är inte heller bara en aktivitet för att lära sig ett specifikt matematiskt innehåll. Det program som man har skapat går att använda även i helt andra fall när man vill veta om ett tal är ett primtal.

Primtalsproblemet är relativt enkelt att förstå konceptuellt, men arbetssamt att hantera med papper och penna så snart talen blir någorlunda stora. När man i sådana fall vill göra ett stort antal likartade beräkningar är det lämpligt att programmera en dator som får sköta beräkningarna. Den här situationen är generell. Om man har en viss klass av problem som man kan skapa en algoritm för att lösa, så kan man också omvandla algoritmen till kod och därigenom skapa ett matematisk verktyg. Oftast är det mer avancerat att skapa koden än att lösa problemet i fråga, men fördelen med koden är att när programmet väl är klart går det snabbt att lösa *alla* problem i den givna klassen. I primtalsexemplet används dessutom datorns egenskap att mycket snabbt kunna genomföra många beräkningar. Att se om 16 eller 17 är primtal är ju enklare att göra för hand än genom att skapa ett programbaserat verktyg. Men om vi vill ta reda på om 65537 är ett primtal så är programmet överlägset.

Lösa ett matematiskt problem

Problemet i dubbleringsexemplet är endast en enkel övning i dubblering och programmet som löser problemet går inte direkt att använda till något annat än att lösa just detta problem. Men det är ändå intressant som exempel på att programmering ibland kan vara ett möjligt och kanske till och med ett effektivt sätt att lösa vissa matematiska problem. För att det ska förhålla sig så behöver ju problemet vara relativt svårt att lösa manuellt, men relativt lätt att förstå algoritmiskt. Man måste också ha tillräcklig vana att programmera för att kunna skapa, testa och förbättra den algoritm för lösning som man har tagit fram.

Att undersöka en matematisk idé

I månghörningsexemplet är poängen att programmering används för att utforska en viss matematisk idé – den om månghörningar och vinkelsummor, för att vara exakt. I både primtalsexemplet och dubbleringsexemplet skapar vi program som ger oss svaret på en fråga. Men i månghörningsexemplet skapar programmet bara en bild som är mer eller mindre korrekt och vi får själva experimentera med olika variabler för att hitta olika lösningar. Programmet som vi har skapat ger bara en generell ram som gör det möjligt att testa många varianter på ett snabbt sätt och med många upprepningar. I det här fallet är arbetet tinkeringbaserat då eleven modifierar redan existerande program. Det går naturligtvis att se det som att programmet bara löser ett intressant problem, men uppgiften är upplagd så att egenskaper hos vissa matematiska objekt/begrepp (månghörningar) sätts i relation till andra begrepp (vinklar, sträckor). De färdiga programmen är inte i fokus utan det är de matematiska idéer som man urskiljer, uttrycker, undersöker och använder när man skapar och använder programmet som är i fokus.

Ett sätt att tänka om orkestrering av lektioner

Som vi har beskrivit tidigare i modulen ställer undervisning om och med programmering i matematikundervisningen speciella krav eftersom det både handlar om programmering och matematik. Eleverna ska få möjlighet att bekanta sig med programmeringsmiljön och att experimentera med egen programmering, genom tinkering eller modifikation av kod som andra har gjort. När programmering lyfts in i matematikundervisningen är det rimligt att det är större fokus på det matematiska innehållet. Det matematiska innehållet avgör vilken lektionsdesign man väljer att använda. I modulens tidigare delar har lärarledda diskussioner av det matematiska innehållet med fördel kunnat placeras i slutet av lektionen. Eleverna kan ju mycket väl ha använt olika matematiska idéer eller principer för att lösa samma programmeringsuppgift, eller arbetat med något olika programmeringsuppgifter. Läraren kan då lyfta det matematiska innehållet i efterhand. Men när programmering ska användas för matematiska syften, är det rimligt att anta att det matematiska innehåll som eleverna ska möta kräver en introduktion av läraren. Kanske behöver även relevanta programmeringskonstruktioner och koncept introduceras. Det kan också vara lämpligt att hålla klassen mer samlad runt ett gemensamt matematiskt innehåll eller ett gemensamt problem. De lektionsförslag som finns i den här delen baseras alla på sådana orkestreringsmönster.

Förslag på orkestrering

Det är bra att före lektionen tänka igenom vilka matematiska begrepp och procedurer som aktiviteten kommer att innehålla och att fundera igenom vilka av dessa som betraktas som förkunskaper och vilka som du vill att eleverna ska utforska eller öva på.

Fundera också igenom vilka programmeringsrelaterade konstruktioner och koncept som kommer att behövas och vilka av dessa du betraktar som förkunskaper och vilka som du snarare vill att eleverna ska utforska eller öva på. Lämna ut instruktioner för det som du anser är förkunskaper inom detta område, så att delar av koden kan användas på ett instrumentellt sätt av eleverna. På så sätt hindras de inte i onödan, utan kan arbeta med att utforska relationerna mellan matematiska representationer och kodrepresentationer.

Planera hur du ska summera lektionen eller lektionsserien. Lyft i denna avslutning fram:

a: vilka matematiska begrepp och procedurer vi har utnyttjat, och

b: vilka programmeringstekniker vi har utnyttjat.

Var noga med terminologi och precision, genom att själv omformulera elevernas kommunikation med korrekta begrepp om det behövs.

Sammanfattning av modulen

Den här modulen har handlat om programmering i matematikundervisningen. Kursplanens skrivningar innebär inte att matematikundervisningen ska användas för att göra eleverna till kompetenta programmerare i allmänhet. Formuleringarna som berör programmering är alltid knutna till centralt innehåll eller matematiska förmågor. Detta har påverkat modulens budskap så att den i högre grad fokuserar på att undersöka och modifiera existerande program än att helt skapa egna. Den behandlar också i högre grad vilken matematik eller vilka matematiska förmågor som relaterar till programmering än olika programmeringstekniska begrepp. Men, med ledning från forskningen om digitala verktyg från bland annat Trouche (2004), så är det viktigt att uppmärksamma att eleverna får möjlighet att lära sig verktyget. Språk och miljöer byts ut och förändras över tid, men med kunskaper om grundläggande principer och byggstenar får eleverna verktyg för att lösa problem och förverkliga idéer, med hjälp av programmering.

På det didaktiska planet har vi utgått från att många lärare är nybörjare när det gäller att lyfta in programmering i matematikundervisningen. Programmeringsinslagen i lektionsförslagen är därför oftast av utforskande karaktär, där läraren inte alltid förväntas ha alla svar färdiga. Men även en sådan undervisning måste vara välplanerad, kanske mer välplanerad än undervisning som bygger på att läraren ger eleverna explicita instruktioner och förklaringar. Därför behandlar modulen hur lektionen orkestreras och olika möjligheter att orkestrera lektioner i olika sammanhang. En lektion som innehåller programmering är ofta mer organisatoriskt komplicerad än andra lektioner eftersom själva tekniken också måste beredas lämplig plats, men också observeras så att den inte tar över. Detta är en del av orkestreringen. Eftersom programmering är nytt för många lärare, så kan det med fördel byggas in i en existerande matematikundervisning, där läraren har en upparbetad trygghet. Det är av det skälet som modulen har varit upplagd så att den startade med ett större fokus på programmeringen för att i slutet handla om programmeringens roll i själva matematiken.

Referenser

Trouche, L. (2004). Managing the complexity of human/machine interactions in computerized learning environments: Guiding students' command process through instrumental orchestrations. *International Journal of Computers for Mathematical Learning*, *9*, 281-307.