Om programmering i matematikundervisning

Ola Helenius, NCM, Morten Misfeldt, Aalborg universitet, Lennart Rolandsson, Uppsala universitet och Ulrika Ryan, Malmö universitet

Programmering har tillkommit som ett innehåll i matematikämnet och den här modulen tar upp frågor som handlar om matematikundervisning och programmering. Vilka didaktiska val behöver göras? Hur kan klassrumsarbetet organiseras? Vilka mål har vi för olika aktiviteter? Hur kan vi säkerställa att matematiken blir uppmärksammad?

Teknik har stor påverkan på vår vardag. Samhällets tekniska, ekonomiska och demokratiska utveckling hänger samman med digital teknik och sådan påverkar också i allt större grad privata sociala relationer. De flesta är därför överens om att skolan ska lära elever att klara ett liv i en digitaliserad värld. Det innebär att eleverna ska få lära sig att använda olika digitala verktyg, med både praktiska och pedagogiska syften. Men de ska också få lära sig att designa, konstruera och manipulera digitala verktyg. De nya skrivningarna om programmering i läroplanen lyfter det senare perspektivet. En tanke bakom detta är att vi genom att förstå hur digitala system fungerar och är uppbyggda, blir bättre rustade att förstå och agera i den värld som omger oss.

I en rad länder har fokus ökat på digital teknik i undervisningen, hur digitala verktyg och system fungerar samt hur man med sådan kan utveckla, utforma och påverka sin omvärld. Detta innefattar också att lära sig om programmering. Motiven för detta är, förutom att eleverna bättre ska förstå sin digitala omvärld, att de ska kunna svara mot framtidens arbetsmarknadsbehov. De länder som inför programmering som ett innehåll i undervisningen gör det på olika sätt, men två huvudsakliga tillvägagångssätt kan urskiljas. Antingen ses programmering som ett eget självständigt ämne i skolan, med särskilda lärare och egna lektionstimmar, eller så knyts programmering till existerande ämnen, i synnerhet till matematik och naturvetenskapliga ämnen.

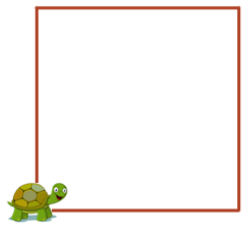
Att vi i Sverige har knutit programmering till undervisningen i matematik och teknik är således ett val, och andra länder gör andra val.

När programmering knyts till matematikundervisningen finns det möjlighet till ämnesmässiga synergieffekter. Det kan exempelvis handla om att utforska matematiska strukturer med hjälp av programmering och om att använda matematiskt kunnande för att utveckla digitala produkter. Båda dessa sidor av programmering kommer att behandlas i modulen.

Programmering i skolan – en kort historisk återblick

Programmering har i varierande grad varit ett innehåll i undervisning sedan slutet av 1960-talet, men inte förrän på 80-talet fick man de första egentliga erfarenheterna av att

undervisa i programmering på grundskolenivå. En central person och pionjär var Seymour Papert från Massachusetts Institute of Technology (MIT) i Boston, USA. Under 70-talet genomförde han olika försök med att låta barn programmera grafiska företeelser med hjälp av procedurer. Man arbetade med en ”sköldpadda” som ritade ett grafiskt spår. Sköldpaddan kunde exempelvis ”gå framåt 100 steg”, ”vänd 90 grader” och göra detsamma ”fyra gånger”. På det sättet ritade sköldpaddan en kvadrat. Se figur 1.



Figur 1. Sköldpadda, programmerad att rita ett grafiskt spår, i detta fall en kvadrat.

Papert utvecklade programspråket LOGO och hans idé handlade inte så mycket om att barn skulle konstruera kvadrater med hjälp av kommandon, utan snarare om att detta visuella och relativt lättillgängliga språk skulle inbjuda barn till att undersöka visuella fenomen. Papert hade en vision om att matematik skulle bli ett ämne där man bygger och skapar till skillnad från att utföra enkla matematikuppgifter (Papert, 1980).

I Sverige utgick man från lärares erfarenheter då programmering på 1970-talet för första gången infördes på en gymnasieskola i Ljungby. Eleverna arbetade med så kallade ”hålremsor” och lärarna fick hyra in sig på någon närliggande industri för att använda deras terminaler när det fanns tid. De konstruerade algoritmer för att exempelvis hitta stora primtal. Lärarna som arbetade med detta på Sunnerboskolan kunde visa på en stark motivationsskapande effekt, speciellt för elever med svårigheter i matematik.

På dåvarande Skolöverstyrelsen insåg man att programmering krävde en ny slags pedagogisk kompetens hos matematiklärarna. Tanken var att återkommande inslag i programmering skulle kunna automatisera delar av matematiken och ge möjlighet till ett nytt sätt att arbeta med matematiken i skolan.

Undervisning i ämnet programmering initierade en ämnesdidaktisk diskussion som fortfarande pågår. Med tanke på att programmering på den tiden associerades till hålremsor och terminaler är det inte osannolikt att många lärare upplevde programmering som irrelevant för elevers lärande i matematik. Man frågade sig säkert, så som många gör idag: Vad tillför programmering ämnet matematik?

Vad är programmering?

För att beskriva vad programmering är ska vi se på vad ett datorprogram är, inte i första hand på vad det utför utan hur det är konstruerat. Att skapa ett datorprogram kallas för att programmera.

Ett datorprogram är ett antal logiskt ordnade stegvisa instruktioner som talar om för datorn vilka operationer den ska utföra när programmet körs. De stegvisa instruktionerna ges inte efter hand, utan de finns färdiga i en sekvens för datorn att följa, det blir som ett självständigt förlopp som datorn utför när programmet körs, *exekveras*. De instruktioner som skrivs i ett programspråk kallas för *kod*. För att datorn ska förstå koden krävs att den följer rätt grammatik, det vill säga vissa regler enligt en bestämd *syntax*.

Vi kan jämföra detta med ett matlagningsrecept, där det finns färdiga stegvisa instruktioner att följa. Då vi läser ett recept finns vissa förgivettaganden inbyggda, instruktioner som förutsätter vår erfarenhet och slutledningsförmåga. Står det i receptet att tre ägg ska vispas så är det underförstått att en skål och visp ska plockas fram, att äggen ska knäckas ner i skålen, att vispen ska föras ner i de knäckta äggen och hastigt föras i cirkulära rörelser. Sådana slutsatser kan en dator inte dra då den läser kod skriven i något viss programspråk. Istället krävs mycket precisa instruktioner anpassade för det programspråk som används och som är noga ordnade i stegvisa logiska *sekvenser* som leder till att utfallet blir det förväntade. Ett enskilt steg i instruktionen som skrivs med kod kallas för en *sats*. En sats kan antingen villkorslöst tala om för datorn vad den ska göra eller låta datorn bestämma utifrån villkorssatser. En villkorssats talar om för datorn att *om* något är på ett visst sätt *då* ska den göra något specifikt. Om vi exempelvis ska laga mat till 12 personer blir kanske konsekvensen att vi behöver 6 ägg. Även andra ingredienser kommer att bero på antalet personer, medan annat inte påverkas av antalet portioner, exempelvis spisplattans temperatur. Villkorssatser bygger på så kallad Boolsk logik som handlar om förutsättningar och konsekvenser.

För att datorn ska kunna utföra det vi vill, måste den få veta vilka förutsättningarna är, den måste få *indata*. I matlagningsexemplet är 12 indata, ”matlagningsdatorn” skulle behöva informationen ”12” för att kunna anpassa receptet. Även annan information skulle kunna vara indata, exempelvis hur stor skål som ska väljas för pannkakssmeten. Ett program som exekveras kan även lämna *utdata*. I matlagningsexemplet skulle det kunna vara hur många pannkakor som receptet visade sig räcka till.

Ett antal satser kan få datorn att utföra en viss *funktion*. I matlagningsexemplet kan en funktion vara att ta fram och knäcka ett ägg ner i skålen. För att slippa ge datorn samma uppsättning instruktioner om och om igen används en följd av koder, kodsekvenser, för att konstruera en viss funktion (t.ex. att knäcka ägg). Sådana funktioner kan sedan kallas på flera gånger i programmet. När man vill upprepa en uppsättning instruktioner flera gånger i rad, till exempel knäcka flera ägg, kan man använda en *loop* där en viss kod upprepas, *itereras*. På så sätt blir programmeringen mer effektiv och koden blir ofta mer lättöverskådlig. En uppsättning kodsekvenser som utför en viss funktion kallas ibland för en *algoritm*. Ett pannkaksrecept skulle kunna vara en algoritm. För att få ihop en pannkakstårta skulle det krävas att flera olika algoritmer kopplades samman. Ett datorprogram utgörs av sådana sammankopplade algoritmer.

Några centrala begrepp och definitioner

Algoritm: En algoritm är en uppsättning instruktioner som är stegvis ordnade med tydliga kategorier av indata och utdata som löser en väldefinierad uppgift. Algoritmer är centrala i programmering då de beskriver handlingar på ett sätt som kan översättas till en kod som datorn förstår.

Programspråk: Programspråk är språk som man använder för att skriva kod. Det finns både textbaserade programspråk som Pascal, Java, C++ och Python och visuella programspråk som Scratch och Microbit. Visuella programmeringsmiljöer där fördefinierade grafiska element, bilder, sätts samman kallas för blockprogrammering.

Kod: En beskrivning (av datorprogrammet) i ett programspråk som kan tolkas av en dator kallas för kod. För att en programkod ska fungera måste den skrivas korrekt och med rätt grammatik, med rätt *syntax*. Syntax kan variera mellan olika programspråk.

Pseudokod: När man gör ett utkast, en skiss, till sitt program börjar man med att skriva en pseudokod. Den beskriver algoritmen med hjälp av vanligt språk, eventuellt kompletterat med programspråk och matematiska symboler. Pseudokod är lättare att skriva (och läsa) än kod i ett specifikt programspråk och ställer inga krav på exakthet och syntax. Att skriva pseudokod hjälper eleverna att tänka igenom problemet och strukturera sin programmering.

Sats: Ett program byggs upp av instruktioner. Instruktionerna uttrycks i ett programspråk och kallas kod. Ett annat ord för instruktioner är satser.

Villkorssats: En sats som gör det möjligt att välja mellan ett eller flera alternativa satser beroende på om ett villkor är uppfyllt eller inte; *om* något är på ett visst sätt gör *då* något specifikt.

Loop: En loop är en instruktion som itereras, det vill säga repeteras. Loopar används för att vi ska slippa skriva samma del av ett program flera gånger.

Exekvera: Att låta en dator genomföra instruktionerna i ett program, det vill säga att köra programmet, kallas för att exekvera. Betydelsemässigt är det ingen skillnad mellan exekvera och köra.

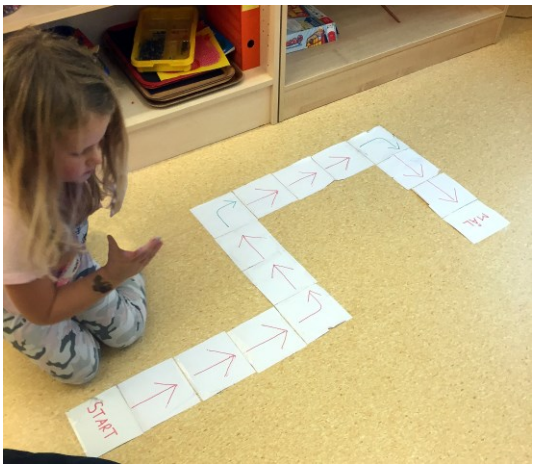
Felsökning: Om programmet inte fungerar som det var tänkt måste man leta efter felet, buggen, och åtgärda det. Felsökning och ”avlusning” är en väsentlig del av programmeringen.

Se också Skolverkets begreppsordlista som finns i webbkursen ”Om programmering”: https://www.skolverket.se/skolutveckling/kompetensutveckling/om-programmering-webbkurs-till-alla-som-arbetar-i-skolan-forskolan-och-vuxenutbildningen

Att programmera

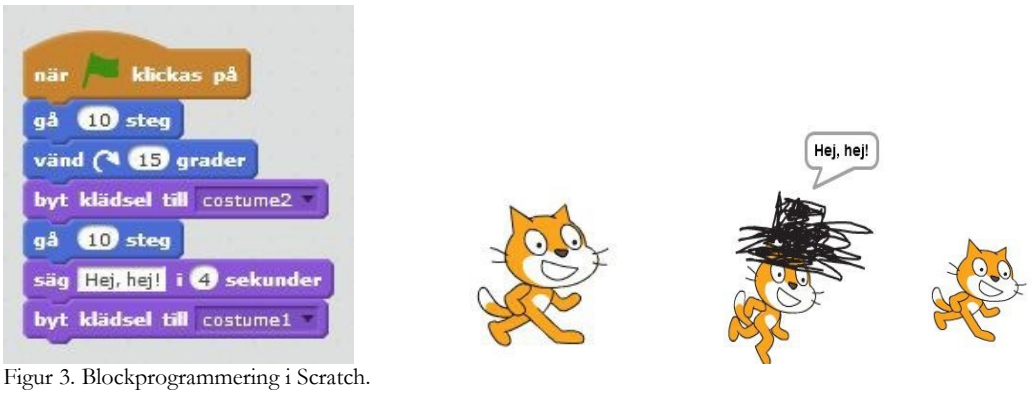
I praktiskt arbete med programmering kompletteras den analytiska och systematiska sidan av programmering med en kreativ och skapande sida, där man inte alltid konstruerar program från början utan också modifierar och kombinerar ihop redan färdiga program.

En bra ingång till programmering är att arbeta utan dator med stegvisa instruktioner. De stegvisa instruktionerna kan ges med hjälp av till exempel bilder, pilar, bokstäver eller ord som läggs eller skrivs i en viss ordning. För att göra övergången från stegvisa instruktioner utan dator till programmering på en digital enhet tydlig, kan man fokusera på vikten av att instruktioner ges i rätt ordning. Allteftersom ska eleverna sedan få möjlighet att arbeta i olika miljöer och med olika programspråk och de ska få möjlighet att både skapa program från början och att undersöka och modifiera andras program.



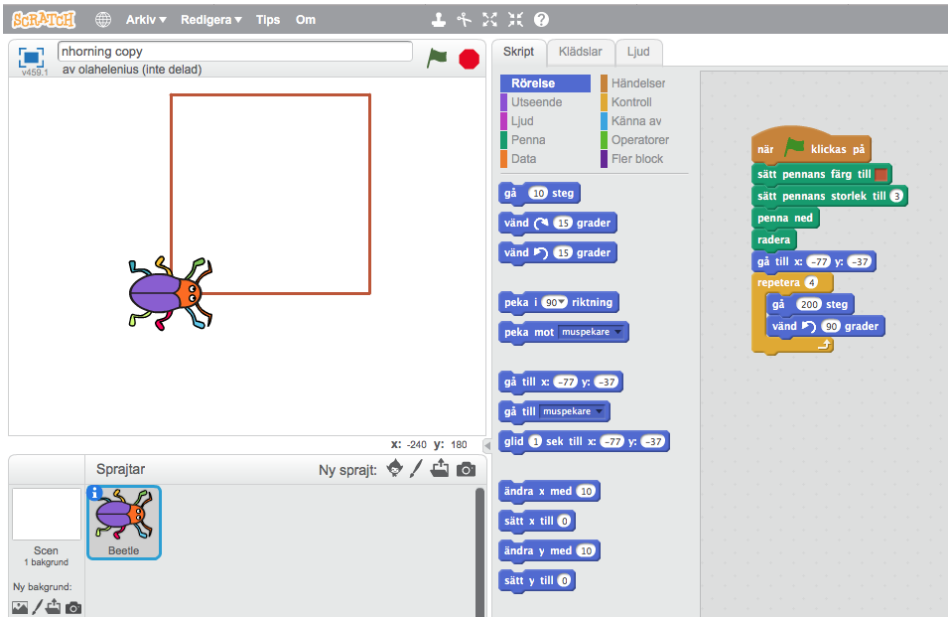
Figur 2. En förövning till programmering där en sekvens av symboler kombineras till en algoritm.

För grundskolans yngre elever används ofta så kallad blockprogrammering, programmering i en visuell programmeringsmiljö. Scratch är ett exempel på en sådan. Olika färdiga block som representerar instruktioner sätts samman till ett program. I figur 3 syns ett exempel på blockprogrammering i Scratch tillsammans med klipp från utdata, datorns utsignaler som i det här fallet visas på skärmen i form av animeringar då programmet körs. I de vita fälten anger programmeraren indata som i det här fallet talar om förutsättningarna för hur långt katten ska gå ,”10” steg, hur den skan vända sig, ”15” grader, vad katten ska säga, ”Hej hej” samt till vilken klädsel han ska byta ”costume 2” och ”costume 1”.



Figur 3. Blockprogrammering i Scratch.

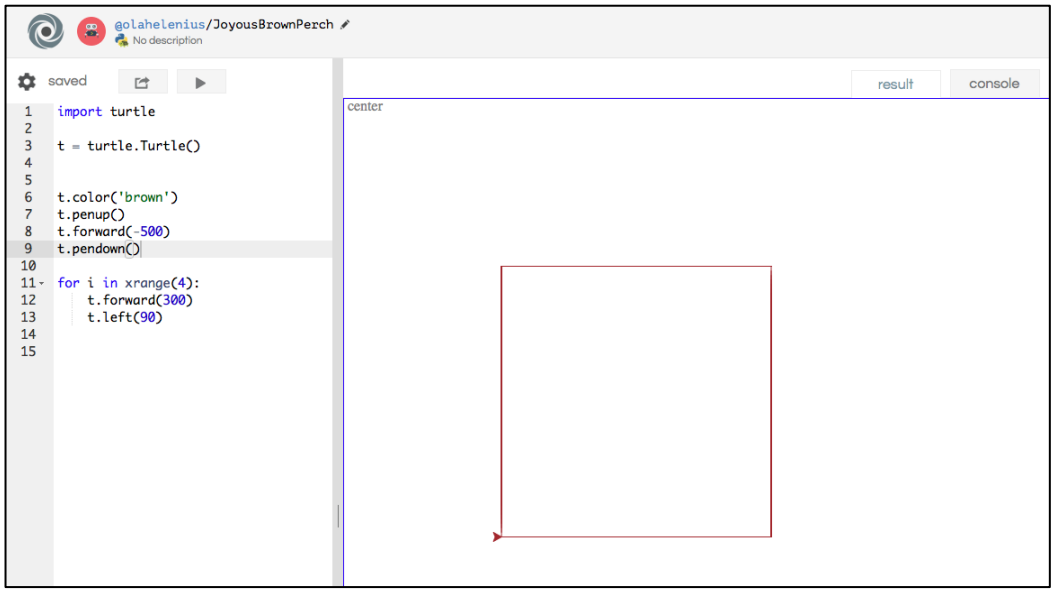
I blockprogrammering har man ofta en lista på kommandon som man drar till ett arbetsbord och sätter samman på det sätt man vill. Sedan kan man köra programmet och i vid sidan se resultatet. I Scratch ser programmeringsmiljön ut som i figur 4.



Figur 4. Programmeringsmiljön Scratch.

När eleverna senare i årskurs 7-9 ska möta programmering i olika programmeringsmiljöer ska de förutom blockprogrammering använda *textbaserad programmering*.

Även i textbaserad programmering bygger man sina lösningar med hjälp av steg-för-steg instruktioner för att skapa kod, men istället för att dra färdiga block skriver man in instruktionerna i en *texteditor*. I figur 5 syns textfönstret till vänster och resultatet när programmet har körts till höger.



Figur 5. Programmeringsmiljön repl.it, med programspråk Python 3 och turtle graphics.

Det vi ser är återigen en kvadrat, precis som vi tidigare såg i Scratch. Här används det som kallas ”sköldpaddsgrafik” i Python gjord i en texteditor på webplatsen repl.it. Referensen till sköldpaddor hänger ihop med Seymor Paperts arbete med LOGO. Liksom i Scratchexemplet i figur 4 definieras först en pennfärg, man går sen till en startpunkt och därefter genomförs med hjälp av en loop fyra förflyttningar och vridningar. Skillnaden är att vi nu har skrivit instruktioner med text istället för att dra och släppa kodblock.

Analysera, strukturera, designa en lösning och skriv koden

Programmeringsarbetet följer olika faser och innan programmeraren börjar skriva programkod krävs mycket arbete. Inledningsvis görs en problemanalys som mycket väl kan liknas vid problemlösning i matematik. Det rör sig om att identifiera delproblem vars lösningar sammantaget utgör lösningen på hela problemet. Problemet behöver sedan sekvenseras logiskt så att en dator kan förstå det.

När problemet är strukturerat och uppdelat i mindre delar som går att lösa skriver man koden som utgör det slutliga programmet. Sådan kod kan skrivas i olika programmeringsmiljöer eller programspråk. Att tänka steg för steg i logiska sekvenser på ett sådant sätt att en dator kan lösa problem kallas ibland för datalogiskt tänkande. Maskinen gör nämligen inte det som vi tänker, utan utför de instruktioner som koden ger, den kod som vi har skrivit med symboler.

Testa, hitta felen och avlusa programmet

Oavsett om man är expert eller nybörjare består programmeringsarbetet till stor del av att hitta buggar eller fel och åtgärda dessa, så kallad felsökning och avlusning (från den engelska termen ”debugging”). Även om vi ger noggrant genomtänkta stegvisa instruktioner finns det mycket som kan gå fel i själva konstruktionen av algoritmen, så att

maskinen (datorn) inte utför det som vi har tänkt. Fel som uppstår kan bero på många olika saker. Det kan till exempel vara att man glömt ett tecken (syntaxfel), att man stavat eller skrivit fel (semantiska fel), att man har skrivit instruktionerna i fel ordning (logiska fel) eller att man har tolkat uppgiften fel. Eftersom minsta lilla fel bidrar till att programmet inte fungerar blir felsökningen ett stort arbete i programmeringsprocessen. Därför består arbetet med programmering oftast av en upprepad process av problemsekvensering, kodskrivning, exekvering och felsökning innan det slutliga datorprogrammet är klart.

Ibland beskrivs programmeringsarbetet som ett givet arbetsschema med logiska arbetssteg, vilket kan misstolkas som att det skulle vara ett linjärt arbete. Arbetet är dock mycket mera komplext, då förståelsen för problemet växer i interaktion med programspråket och miljön under arbetets gång. Nedanstående sexpunktslista kan följas på olika sätt, du kan till exempel arbeta med stegen 1, 2, 3, 1, 2, 3, 1, 2, 3, 4, 5, 4, 5, 6, 5, 6 och så vidare innan resultatet blir som du har tänkt.

1: Analysera problemet

2: Utvärdera olika lösningsmodeller

3: Designa en lösning

4: Skriv programkoden

5: Testa programmet

6: Felsök och avlusa programmet

Undervisning i programmering

När man ska undervisa med och i programmering måste flera olika aspekter beaktas. Vi kan fokusera på att bygga upp elevernas kunskaper om programmering, men vi kan också inrikta oss på att göra det möjligt för eleverna att utforma och utveckla egna program och att få saker att ske med hjälp av programmering. Vi kan också använda programmering för att undersöka exempelvis matematiska begrepp.

Om vi ser på hur undervisningen i programmering brukar se ut, kan vi tänka på den som *projektorienterad* eller som *instruktionsorienterad.* Olika angreppssätt betonar olika aspekter när man arbetar med programmering

En projektorienterad programmeringsundervisning är experimenterande och lekande. Den kan exempelvis ta sin utgångspunkt i egna eller andras halvfärdiga programmeringsprojekt som justeras och ändras genom det som brukar kallas ”*tinkering*”. Begreppet ”tinkering” betyder att syssla med, pröva, rätta till, meka och handlar oftast om ett experimenterande skapande. Sådana processer är centrala när man arbetar med komplexa digitala konstruktioner. Vi återkommer till begreppet ”tinkering” i del 2. Instruktionsorienterad undervisning har en arbetsgång där de mest grundläggande begreppen presenteras i en genomtänkt ordning och läraren har färdiga exempel på hur dessa används.

Projektorienterad och instruktionsorienterad undervisning behöver inte ses som en fråga om antingen eller. För läraren handlar det om att göra ett medvetet val av arbetssätt i sin undervisning, med hänsyn till olika aspekter.

Referenser

Papert, S. (1980). *Mindstorms: children, computers and powerful ideas.* Brighton: Harvester Press.