Johansson, Juhlin, Tossavainen & Wedestig

Programmering på gymnasiet

Författarna funderar över hur programmering kan vara ett verktyg för en fördjupad förståelse av matematiska begrepp i gymnasieskolan. Först diskuterar de frågan ur ett teoretiskt perspektiv och därefter rapporterar de om gymnasieelevers erfarenheter av att studera matematik med programmering.

Sedan augusti 2018 är programmering en obligatorisk del av matematikundervisningen i den svenska skolan. Införandet av programmering i skolan började flera decennier före de senaste revideringarna av läroplaner. Redan 1975, i ett av de allra första numren av Nämnaren, skrev Lars-Eric Björk om att skolan måste ge alla elever elementära kunskaper om vad en dator är och kan göra, samt hur datorer kan användas för problemlösning. Intressant nog betonade han också vikten av diskussion kring både fördelarna och nackdelarna med det datoriserade samhället.

Mycket har hänt sedan 1975 men bläddrar man i programmeringsrelaterade artiklar från olika decennier i Nämnarens artikelregister hittar man en viss kontinuitet. Vissa tillämpningsområden och typuppgifter verkar förekomma trots lärar- och datorgenerationsväxlingar. Det är inte underligt eftersom en dator ursprungligen är en sekvenskalkylator, det vill säga en typ av räknare som snabbt kan lösa just aritmetiska uppgifter. Även moderna datorer passar väl till de flesta uppgifter som kan lösas med hjälp av första ordningens logik och numeriska beräkningar. En stor skillnad mellan moderna datorer och äldre apparater är att de förstnämnda kan styras även med hjälp av visuella programmeringsspråk som också är anpassade till symboliska beräkningar.

Programmering i problemlösning

Idag finns det ganska mycket utrymme för att variera innehållet i programmeringsundervisningen. Detta gäller särskilt gymnasieskolan. I praktiken får lärare fria händer eftersom problemlösning är det enda innehållsområdet i gymnasieskolans ämnesplan för matematik där programmering explicit nämns. Programmering ses som ett alternativt verktyg för att lära sig strategier för matematisk problemlösning och modellering av olika situationer. Ämnesplanen tar dock ingen ställning till inom vilket matematiskt innehåll elever ska få lära sig att använda programmering, och i kunskapskraven nämns ingenting om programmering.

Samtidigt ställer ämnesplanen väldigt höga krav på matematiklärare. Skolverket skriver i sitt kommentarmaterial att en problemlösningsuppgift är en uppgift som är utmanande och kräver ansträngning. Det är tydligt att det från början inte ska finnas en för eleven känd lösningsmetod. Det innebär att eleven utöver att anstränga sig måste vara beredd att lägga ned tid. Problemlösningsförmåga innebär att kunna analysera och tolka problem

vilket inkluderar ett medvetet användande av problemlösningsstrategier som att förenkla problemet, införa lämpliga beteckningar eller ändra förutsättningarna. Problemlösning, och än mer problemlösning med programmering som verktyg, förutsätter en process där man kan känna sig trygg att misslyckas och börja om från början.

Ser vi till Polyas problemlösningssteg, så måste eleverna klara av att identifiera ett enklare problem, klara av att lösa det och sedan bygga en modell. Förutom att allt detta är tidskrävande utmanas läraren eftersom hen nästan omedelbart behöver kunna läsa, tolka och analysera olika koder som eleverna producerar. Läraren måste också kunna anpassa sin undervisning i ett klassrum där elevernas erfarenheter och kompetens i programmering ofta skiljer sig mer än deras färdigheter i matematik.

I denna artikel behandlar vi programmering och dess roll i matematikundervisningen ur ett annat och mer sällan diskuterat perspektiv. I stället för att presentera olika tillvägagångssätt till problemlösning med programmering bekantar vi oss med en teori som förklarar varför programmering kan bidra till lärande om matematiska begrepp men också bidra till kollaborativa arbetssätt i matematikundervisningen.

APOS – ett teoretiskt perspektiv

Det finns många olika teorier som beskriver lärande i matematik. En av dem, APOS-teorin, beskriver med hjälp av fyra steg eller nivåer hur elevens förståelse för matematiska begrepp och andra kunskapsobjekt utvecklas.

- Det första steget kallas *Aktion*. Det innebär att eleven förstår ett begrepp som en enstaka operation. Ett exempel på aktion är att eleven med hjälp av en formel kan beräkna att derivatan av x^2 är $2x$.
- Det andra steget heter *Process*. En elev som har nått denna nivå kan kombinera flera operationer och redan bemästrade kunskaper om begreppet till relevanta processer. Ett exempel på process är att eleven kan kombinera derivering av monom och deriveringsregler för att hitta derivatan av polynom.
- Det tredje steget kallas *Objekt*. En elev som har nått denna nivå har börjat se begreppet som en helhet med vissa egenskaper. Till exempel att derivata är en funktion som kan kombineras med andra funktioner och i vissa fall deriveras om.
- Det sista steget är *Schema*. På denna nivå bemästrar eleven begreppet, kan handla och generalisera det inom en större ram med anknytningar till olika delar inom matematik. Ett exempel på schema är att eleven förstår sambandet mellan derivering och integrering och vet att man kan beteckna derivata på olika sätt men att det alltid handlar om ett visst gränsvärde.

Programmering enligt APOS

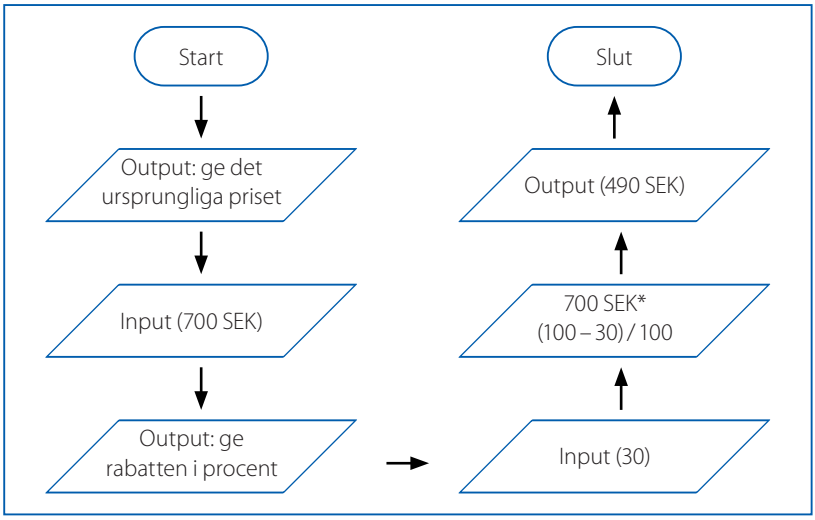
Programmering är väl förenligt med APOS-teorin. Att skriva ett kommando som sätter ett värde på en variabel är en aktion. Att skriva en något längre kommandorad som gör att datorn ber användaren att mata in ett värde på en variabel samt läsa in det givna värdet, är ett exempel på en process. Kombinerar man den med ett par andra rader som räknar ut till exempel kvadratroten ur kvadraten av det inmatade talet och skriver ut svaret får man en kod som i sin helhet representerar allt som behövs för att konstruera objektet absolutbelopp. Kan man använda denna funktion på rätt sätt i olika delar av en större kod har man nått åtminstone en del av schemanivåns förståelse för detta begrepp.

Empirisk forskning har visat att det svåraste steget i lärande om matematiska begrepp är övergången från processnivå till objektnivå. Att ta detta steg innebär att man måste ha samlat ihop en stor mängd exempel på och erfarenheter av olika slags uppgifter kring begreppet och därefter måste man lyckas kondensera allt till ett självständigt matematiskt objekt. Det förutsätter bland annat att man kan urskilja vad som är generaliserbart och gemensamt i olika exempel.

En viktig fördel med programmering är att en kod synliggör var gemensamma och generaliserbara element i olika processer befinner sig. Vi demonstrerar detta med hjälp av ett par flödesscheman så att vi inte fastnar i särskiljande detaljer mellan olika programmeringsspråk. Figur 1 beskriver en lösning på följande problem:

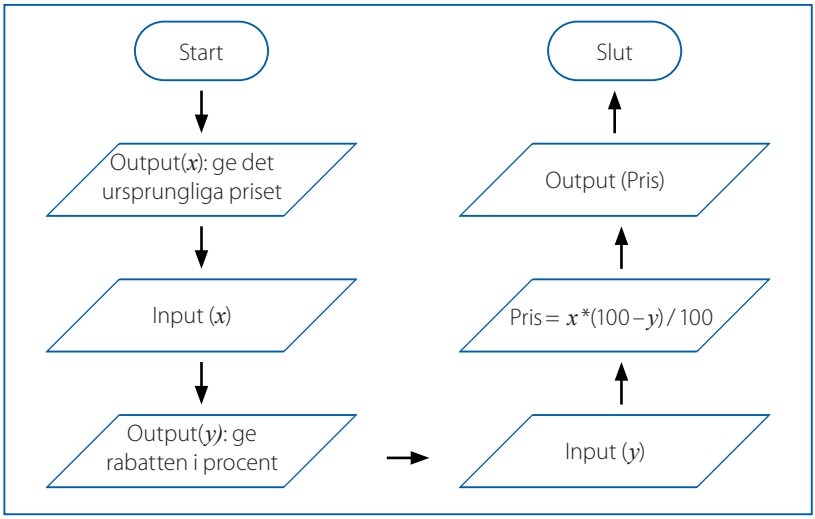
> Ett par hörlurar kostar vanligen 700 SEK men för tillfället får man 30 % rabatt. Vad blir priset då?

Proceduren i sig är ganska enkel. Det ursprungliga priset och rabatt i procent läses in och därefter räknar datorn, med hjälp av en formel, ut det nya priset och skriver ut det. Allt detta kan beskrivas med sex steg.



Figur 1. Räkna ut ett nytt pris för ett par hörlurar à 700 SEK efter rabatt på 30 %.

En elev som redan ligger på processnivå med avseende på begreppet, borde till exempel se att de sex stegen i processen inte kan tas i vilken ordning som helst för att få rätt lösning av den givna uppgiften. Troligen förstår hen också att om uppgiften hade varit att räkna ut rabatt i procent då borde man ha använt en annan formel i det femte steget samt reviderat innehållet av det tredje steget. Men samtidigt finns det några detaljer i denna process som kan variera utan att man behöver ändra något i processens genomförande. Det är helt irrelevant att det handlar om hörlurar. Och om det gamla priset och procenttalet var något annat skulle det räcka att justera formeln i det femte steget och processen skulle ge det nya priset för vilken vara som helst. Figur 2 visar samma process som i figur 1 med den skillnaden att de inmatade talen har ersatts med variablerna x och y.



Figur 2. Räkna ut ett nytt pris för en vara à x SEK efter rabatt på y %.

Att ersätta ett tal med en variabel i en kod är alltså ett konkret sätt att generalisera processen. Å andra sidan framgår det klart, både i flödesschemana och motsvarande koder, att processerna är strukturellt lika. Dessutom syns processernas interna samband väl i båda representationerna. I steg 5 visar formeln hur det ursprungliga priset och rabatten är relaterade (figur 1) eller vilket sambandet det finns mellan x och y (figur 2). På detta sätt sker en utveckling från en lösning av ett enstaka konkret problem till ett objekt som kan användas som en allmän modell av procentuell förändring.

Att programmering visualiserar matematiskt tänkande är något som kan användas också för att stärka kollaborativa arbetssätt i matematikundervisningen. En kod skulle kunna fungera i matematiska diskussioner på samma sätt som en bild eller en annan visuell representation av matematiska objekt som effektiv förmedlare av matematiska idéer. En motivering för detta är att visualisering minskar belastning av arbetsminnet och underlättar kommunikation om de matematiska idéerna. Det är någonting som vi har börjat undersöka i ett samarbete mellan Luleå gymnasieskola och Luleå tekniska universitet.

Elevers erfarenheter

I kursen Matematikspecialisering genomfördes ett moment med programmering. I kursen ingick elever från gymnasiets årskurs 1 och 2. Förkunskaperna i programmering varierade, någon hade hållit på mycket med programmering men de flesta inte alls eller väldigt lite. Därför började kursmomentet med fem lektioner där grunderna i Python samt grundläggande programmering avhandlades. Utvecklingsmiljön som användes var Spyder som är en del av Anaconda och kurslitteratur var *Programmering i matematiken*. Lektionerna leddes av en ämneslärare i programmering och matematik och utgick från bokens upplägg. På grund av pandemin genomfördes en del av undervisningen på distans. Exempel på matematiska problem som ingick i kursen är primtalsfaktorisering och att beräkna sannolikheter för olika utfall vid tärningskast.

Tärningskast

En uppgift var att testa sannolikheten för jämn respektive udda summa vid kast med två sexsidiga tärningar. I uppgiften fanns också möjliga vidareutvecklingar av programmet så att användaren skulle kunna välja antal tärningar och antal sidor på tärningarna. Eleverna jobbade på distans i grupper om 2–3 elever och använde sig av så kallade breakout rooms i Microsoft Teams.

De flesta grupper löste problemet för kast med två tärningar, men att sedan generalisera antal tärningar och sidor blev svårare. Några grupper fastnade på att de i första delen använt *tärning1* och *tärning2*, det vill säga olika variabler för varje tärningskast, och lyckades inte gå vidare till att använda en loop som summerade x antal tärningskast.

Ur ett problemlösningsperspektiv hade det varit bättre om eleverna i stället hade börjat med den generaliserade uppgiften, då hade de haft lättare för att komma på hur den skulle lösas. Om de hade börjat med att rita, eller i alla fall tänka, ett flödesschema så hade de mycket enklare och snabbare löst det här och andra problem. Att eleverna börjar koda först och tänka sedan är ett vanligt problem i programmeringskurser.

Ur ett begreppsmässigt matematiskt perspektiv hade det kanske varit bättre att stanna vid att beräkna sannolikheten för jämn och udda summa vid kast med två tärningar så att sannolikheten hamnat mer i fokus. Den uppgiften löste de flesta och man hade då kunnat jämföra och diskutera den prövade sannolikheten med den teoretiska.

Att grupparbeta i Teams

Att jobba med de beskrivna uppgifterna i Teams fungerade bra, på många sätt till och med bättre än om eleverna hade suttit på plats. När var och en har sin egen skärm att titta på kan de enkelt dela skärm med gruppmedlemmarna så att alla kan vara delaktiga i problemlösandet. Ganska ofta tog en av eleverna en ledande roll i diskussionen kring en strategi som krävs för att lösa uppgiften och en annan elev fokuserade på att kommentera och utveckla strategin genom att fokusera mer på detaljer. Det som var särskilt intressant att upptäcka var att elever ganska lätt hittade sina roller men att de också kunde byta roller mellan olika uppgifter.

APOS-teorin

På en intervju med elever kom flera detaljer upp som är förenliga med APOS-teorin. När de fick frågan om möjliga fördelar med programmering i matematik betonade eleverna vikten av tydliga strukturer och förståelse för i vilken ordning saker bör göras. Detta passar väl ihop med hur lärandet utvecklas från förståelse för enstaka operationer till bemästrande av processer. Eleverna tyckte också att de hade blivit bättre på att hitta fel och otydligheter vid mer komplicerade problem och bevis, vilket visar att de också utvecklat förmågan att dela upp processer till en rad operationer. Ett exempel på att en del elever kunde generalisera sina idéer till matematiska begrepp på objektnivå var att de nämnde programmering som en metod att kontrollera antaganden innan de försökte hitta algebraiska metoder.

När elever tillfrågades om möjliga nackdelar med programmering i matematik, fanns det elever på helt skilda kunskapsnivåer som sade att programmering tar tid från den ”riktiga” matematiken. De yttrade sig också om svårigheter med att hitta problem där programmering blir ett effektivt sätt att lösa dem. Detta skapar frågor om varför man ska använda programmering då det finns effektivare metoder för att lösa samma problem. De här kommentarerna demonstrerar att det inte är enkelt att flytta innehållet av matematiska uppgifter från en kontext till en annan, vilket enligt APOS-teorin förutsätter förståelse för centrala begrepp minst på objektnivå.

Sammanfattning

Som vi ser det kan programmering i matematik bidra till både en stärkt problemlösningsförmåga och förståelse för matematikens struktur och begrepp. Precis som eleverna upplever vi att det tar tid från den ”riktiga matematiken”. Kanske en konsekvens av att programmering saknas i kunskapskraven. Med kompetenta lärare och ett större utrymme i ämnesplanerna är vi ändå övertygade om att programmering har sin naturliga plats i matematikundervisningen. Att lära sig algoritmiskt tänkande ger eleverna viktiga färdigheter för högre studier och de kan tryggare förhålla sig till och påverka den digitala utvecklingen i samhället.

LITTERATUR

Arnon, I., Cottrill, J., Dubinsky, E., Oktaç, A., Roa Fuentes, S., Trigueros, M. & Weller, K. (2014). *APOS Theory. A framework for Research and Curriculum Development in Mathematics Education*. Springer.

Björk, L.-E. (1975). *Datorn, individen och samhället,* Nämnaren 2:2.

Tossavainen, T. (2007). Proceduralising conceptual knowledge of mathematics – or the other way around. I O. Eskilsson & A. Redfors (red), *Ämnesdidaktik ur nationellt och internationellt perspektiv* (s 285–292). Kristianstad University Press.

Tossavainen T. (2009). Who can solve 2x = 1? An analysis of cognitive load related to learning linear equation solving. *Montana Mathematics Enthusiast*, 6(3), 435–48.

Trangius, K. & Hall, E. (2018). *Programmering i matematiken*. Thelin Förlag.