Skolverket

Programmering – centrala begrepp

Constanta Olteanu och Lucian Olteanu, Linnéuniversitetet

Att arbeta med programmering i matematik gör det möjligt för eleverna att pröva sig fram hur de kan använda entydiga stegvisa instruktioner för att lösa ett problem snarare än att hitta ”det rätta svaret”. På grundskolenivå handlar det om ett sätt att tänka hos eleverna och att arbeta så att eleverna till exempel får arbeta med algoritmer, tänka logiskt, bryta ned problem i mindre delar, söka och korrigera fel samt tolka resultat.

Programmeringsprocessen

Ett flertal forskare definierar programmering som en process som karakteriseras av att utforma och skriva en uppsättning instruktioner (ett program) för en dator på ett språk som den kan förstå. Programmeringsprocessen kan exempelvis inkludera följande steg:

- analysera och förstå problemet
- dela upp problemet i mindre delar (bryta ned problemet)
- skissa en lösning eller flera lösningar
- välja en lösning
- hitta återkommande mönster och utnyttja dessa
- skapa en algoritm
- förbättra en algoritm
- skriva instruktioner i naturligt språk, en programmeringsmiljö eller i ett programspråk (”koda”)
- göra felsökningar
- tolka resultat

Programmeringsprocessen är inte linjär, utan man ofta får gå tillbaka till tidigare steg flera gånger. Det första steget i programmeringsprocessen är att utveckla elevernas förmåga att konstruera, beskriva och följa entydiga stegvisa instruktioner. Detta kan göras utan att använda en dator. Det är viktigt att höja svårighetsgraden och komplexiteten gradvis.

Centrala begrepp inom programmering

Instruktioner

Instruktioner utgör grunden i ett program och används för att manipulera data, utföra beräkningar, skriva ut data på skärmen, osv. Det finns tre huvudsakliga typer av instruktioner.

Sekvens – en följd av instruktioner som utförs en i taget i den ordning de skrivits

Alternativ eller villkorssats - används för att ge datorn möjlighet att välja mellan olika instruktioner beroende på den situation som gäller för tillfället

Villkorssatserna skrivs på datorn som, *om-så* eller *om-så-annars* (på engelska if och if-else)

Exempel:

> *Om* du har svarta byxor *så* ska du gå två steg framåt, *annars* ska du snurra runt ett varv.

Repetition eller slingor/loopar - upprepar en del i programmet ett givet antal gånger eller tills ett givet villkor har uppfyllts

Exempel:

> *Upprepa* fyra gånger. Gå två steg framåt och hoppa en gång.
>
> *Repetera* tills du har nått fram till målet: Gå två steg framåt.

Algoritmer

En algoritm är en uppsättning instruktioner som skapas för att till exempel lösa ett problem eller förverkliga en idé och beskriver exakt hur man ska gå tillväga för att lösa problemet. Beskrivningen görs steg-för-steg. I årskurs 4-6 ligger fokus på att skapa denna typ av beskrivningar.

Algoritmer finns överallt i vardagslivet, exempelvis ett matlagningsrecept eller en monteringsanvisning, men används även i vardagliga situationer som när man borstar tänderna eller klär på sig. De kan förekomma i klassrummet, till exempel en lektionsplan eller ett schema. I klassrummet kan man öva på stegvisa entydiga instruktioner genom att en elev ger stegvisa instruktioner till en annan elev för att förflytta sig från en punkt till en annan punkt. Dessa stegvisa instruktioner formar tillsammans en algoritm och bygger på ett systematiskt tillvägagångssätt. Det finns algoritmer som upprepas gång på gång, för alltid, eller tills något annat händer.

Det finns likheter och skillnader mellan användningen av algoritmer i algebra och i programmering. En likhet är att en algoritm består av en följd av instruktioner, operationer eller regler, som beskriver hur en viss uppgift ska lösas (t.ex. en additionsalgoritm, hur vi ställer upp räkneoperationer). En annan likhet är att algoritmer är abstrakta och behöver visualiseras eller konkretiseras för att kunna diskuteras. I algebra kan visualiseringen göras genom att använda naturligt språk, grafer eller tabeller och i programmering genom att använda blockdiagram eller ett programspråk i till exempel visuella programmeringsmiljöer.

Inom matematiken är en algoritm en procedur som utförs i bestämda steg i en viss ordning och används för att lösa ett givet problem eller utföra en beräkning. Algoritmer inom

matematiken kännetecknas av att de består av en beräkningsmetod som alltid utförs på samma sätt oavsett vilka tal som ingår i beräkningen. Skillnaden är att i programmering består en algoritm av stegvisa instruktioner som kan formuleras på olika sätt och till ett visst problem kan finnas flera algoritmer med olika egenskaper. Ett exempel är om man ska gå till ett bestämt mål så kan man gå olika vägar med olika antal steg.

För att kunna stödja eleverna att förstå strukturen i en algoritm är det av betydelse att du som lärare vet att en algoritm måste uppfylla följande villkor:

- ska ha indata – en algoritm har en startpunkt och instruktioner som talar om vad som ska utföras
- ska ge utdata – en algoritm ger ett resultat, men olika algoritmer kan ge samma resultat
- ska vara ändlig – varje algoritm måste avslutas efter ett ändligt antal steg
- ska vara definierad – varje steg i algoritmen ska vara precist definierat

Variabel

Du känner redan till begreppet variabler, som bland annat används i algebra. Då talar man ofta om variabler som x, y, z osv., som används för att representera en kvantitet i ett matematiskt uttryck. Variabler i programmering används lite annorlunda än i algebra. I programmering har variabler ett symboliskt namn som är associerat med ett värde och vars associerade värde kan ändras. I programmering kan vi exempelvis se variabler som olika burkar, där vi kan stoppa in olika värden. Varje burk har ett namn, som är variabelns namn. I vårt exempel är variabelnamnen ”bilar”, ”frukter” och ”spelare”. Lägg märke till att variabelnamnen är valda så att man omedelbart kan förstå vad det står för.

Figur 1. Variabler i programmering

Det är viktigt att förstå att innehållet i varje burk varierar när man kör ett program, vilket inte är fallet med variabler i matematik, där variabeln x i en funktion inte varierar när man genomför olika beräkningar. Beroende på variabelns typ kan man lagra olika värden, såsom till exempel heltal, decimaltal eller text. Låt oss anta att våra variabler *bilar, frukter* och *spelare* ska representera antal och därmed lagrar heltal. Man säger att man *tilldelar* en variabel ett värde. Hur detta ser ut i praktiken beror på vilken programmeringsmiljö man använder eller på vilket programspråk man jobbar i. I många textbaserade språk används likhetstecken som tilldelningsoperator, men denna operator finns inte i exempelvis Scratch som är en programmeringsmiljö. Till skillnad från användningen av likhetstecknet i algebra används detta tecken i programmering med en annan innebörd, nämligen för att föra över innehållet

från högersidan till variabeln på vänstersidan, dvs. att högerledets uttryck beräknas först och därefter får vänsterledets variabel detta värde. Till exempel skulle man skriva ”*frukter* = 5” för att lagra värdet 5 i variabeln *frukter*. Instruktionen ”*frukter* = *frukter* + 4” innebär att man lägger till fyra i burken *frukter*, det vill säga man ersätter det ursprungliga värdet med det nya värdet. Om du ändrar värdet för talet som representerar antal frukter påverkar det inte värdet i variabeln *spelare* eller *bilar*. Med andra ord är variablernas värden oberoende av varandra.

Felsökning

Programmeringsprocessen avslutas med att eleverna tolkar resultatet och säkerställer att indata ger den utdata som var tänkt, det vill säga lösning på problemet. Om så inte är fallet måste en felsökning ske. Felsökning innebär att identifiera och åtgärda eventuella fel. På engelska kallas det *debugging*. För att kunna göra felsökning behöver eleverna uppmärksamma helheten, detaljer i helheten och relationer mellan detaljer, men även egenskaper hos begrepp som avser både algebra och programmering. Felsökning är en viktig del i programmeringsprocessen och eleverna bör därför bli medvetna om detta. Det behövs uthållighet för att hitta fel i programmet och åtgärda dessa.

För att kunna felsöka en instruktion behöver eleverna i första hand uppmärksamma olika fel. Det finns flera olika typer av fel, men vi koncentrerar oss på de slag fel som kan uppträda när eleverna ger instruktioner till varandra utan att använda dator. Några möjliga fel kan vara:

- i vilken ordning olika instruktioner ges eller genomförs
- instruktioner är inte tillräckligt tydliga
- instruktioner saknas i vissa steg
- instruktioner används inte stegvis, det vill säga en för stor mängd information på en gång

Dessutom det kan finnas modellfel när en modell av verkligheten inte stämmer överens med verkligheten. Modellfelet kan bero på att man har gjort orimliga instruktioner eller att man missförstått själva problemet.

Om man använder en dator, så kan det uppträda även andra fel. De vanligaste felen kan vara syntaxfel (ett grammatiskt fel som bryter mot de lagar som ett programmeringsspråk har), exekveringsfel (uppkommer under körning och medför att programmet inte kan utföras), logiskt fel (genererar fel svar eller svar som blir fel bara för en viss sorts indata).

Ett problem och därmed även den algoritm som löser det kan brytas ned i olika delar. I nedbrytningsprocessen kan eleverna förstå, lösa, utveckla och felsöka delarna separat. Beroende på vilket slags fel det är fråga om kan du använda olika metoder för att leta efter och upptäcka fel. Om eleverna ger eller genomför instruktioner utan att använda en dator kan du exempelvis:

- förklara instruktionerna för någon annan
- kontrollera instruktionerna kritiskt
- göra en lista med möjliga fel
- anteckna vad du redan har testat

Om eleverna använder en dator kan du eller eleverna testa programmet systematiskt med olika indata för att se om resultatet blir rätt. Felsökning är ett utmärkt tillfälle för eleverna att lära sig av sina misstag och att bli bättre på programmering.

Andra begrepp

Andra begrepp som används inom programmering är exempelvis

- kod – en sekvens av instruktioner i ett programspråk som datorn kan tolka och utföra
- programspråk – de språk som man använder för att skriva koden (ex. JavaScript, Python)
- satser – instruktioner till datorn för att utföra något (ibland används ordet kommando)
- script – en sekvens av satser
- syntax – språkets grammatik, anger hur instruktioner i språket får bildas
- köra/exekvera – att utföra instruktionerna i ett program

Tinkering

I programmeringsprocessen är det viktigt att förklara och klargöra idéer med exempel och visualiseringar samt att skapa en lärandemiljö som gör det möjligt för eleverna att lära sig programmering (Tohata, 2014). Ett flertal forskare hävdar att en lärandemiljö som tillåter eleverna att testa saker och idéer i en ostrukturerad process, *tinkering,* gynnar lärandet av programmering.

Inom didaktisk forskning presenteras två olika sidor av tinkeringprocessen, dels den fria aktiviteten där elever skapar eller ändrar ett befintligt datorprogram och dels aktiviteter där eleverna får testa, göra fel, samt ge och få feedback från andra elever, lärare eller dator. Det finns forskare som gör en distinktion mellan *tinkerers* och *planerare* (Blikstein, m.fl., 2014). Tinkerers testar och gör en serie stegvisa förändringar i sina program för att skapa en färdig lösning. Planerare däremot är mer systematiska, de tar fram och genomför en handlingsplan för att kunna göra slutliga ändringar i skapandet av en algoritm eller ett program. Att låta eleverna utvecklas som både tinkerers och planerare skapar möjligheter för dem att förstå olika faser i programmeringsprocessen. Användningen av visuella programmeringsmiljöer kan stödja både tinkerers och planerare eftersom miljön erbjuder elever möjligheter att både

testa sig fram och gå systematiskt tillväga. Detta spelar en central roll i vad eleverna har möjlighet att förstå när programmering används i matematikundervisningen.

Tinkering tillåter att idéer kolliderar (dvs. att en eller flera kritiska aspekter medverkar samtidigt) och det är just i dessa kollisionspunkter som kreativitet uppstår. Lärare ger eleverna självtillit genom att de tillåts experimentera, ta risker och utforska egna idéer (Martinez & Stager, 2013). Detta i sin tur leder till att de börjar se sig själva som elever som har bra idéer och kan förvandla sina idéer till verklighet. Detta kan bidra till att eleverna utvecklar färdigheter som kan användas senare i programmering. Dessa färdigheter är centrala dimensioner för att lära sig programmera och inkluderar:

- Vana vid att hantera komplexitet.
- Uthållighet vid arbete med svåra eller stora problem.
- Tolerans för tvetydighet eller osäkerhet.
- Förmåga att hantera öppna problemuppgifter.
- Förmåga att kommunicera och samarbeta med andra för att komma fram till en gemensam lösning.

Förslag på aktiviteter

Nedan finns förslag på aktiviteter som hjälper dig komma igång med grundläggande programmering med eleverna. Inled med att berätta om att en del av programmering är att lära sig läsa och felsöka sin egen och andras kod samt att skapa entydiga instruktioner. Precis som i olika matematikaktiviteter kan eleverna, även i programmering, behöva möta flera liknande övningar för att de ska ha möjlighet att förstå själva huvuduppgiften.

Inled en lektion med en gemensam övning. Lägg en penna och ett papper på ett bord. Eleverna ska nu ge dig instruktioner att ta upp pennan från bordet och lägga tillbaka den på bordet igen. Med denna övning kan du göra eleverna medvetna om vikten av tydlighet och vad eleverna bör tänka på när de ger instruktioner för att skapa en algoritm. När du har skapat en gemensam förståelse för instruktionernas betydelse vid programmering kan du genomföra en eller flera av nedanstående aktiviteter.

Aktivitet 1 - Algoritm – felsökning

I denna aktivitet kan du välja att genomföra antingen variant 1 eller 2. Skillnaden mellan variant 1 och variant 2 är att i variant 1 ger eleverna instruktioner medan i variant 2 följer eleverna de instruktioner som ges.

Variant 1

Utrustning: Valfritt föremål, papper och penna.

Syfte: Att utveckla elevernas förståelse för ordningen i vilken olika instruktioner ges, att instruktionerna behöver vara tydliga, att man inte kan ge en för stor mängd information på en gång samt behovet av att korrigera instruktionerna om vissa steg saknas eller är felaktiga.

Matematiskt innehåll: Förflyttning mellan olika punkter i rymden (ett föremåls läge vid olika tillfällen).

Genomförande: Ställ ett föremål på golvet. Dela in eleverna i par och låt dem skriva ner en algoritm som ska vara så tydlig att du ska kunna följa den för att ta upp föremålet från golvet och sätta det på ett bord. Be eleverna ange hur säkra de tror de är på att deras algoritm fungerar och välj några algoritmer som:

a) gör det möjligt för dig att ta upp föremålet från golvet och sätta det på ett bord;

b) inte gör det möjligt att ta upp föremålet från golvet och sätta det på ett bord.

Utför instruktionerna och felsök därefter tillsammans med eleverna de instruktioner som inte fungerade. Diskutera även vad som ska ändras och varför. I detta sammanhang kan du även lyfta fram att en given algoritm ger ett givet resultat, men att olika algoritmer kan ge samma resultat.

Variant 2

Utrustning: Valfritt föremål.

Förberedelser: Gör klart instruktionerna som eleverna ska följa. Kopiera och dela instruktionerna till varje par.

Syfte: Att utveckla elevernas förståelse för ordningen i vilken olika instruktioner ges, att instruktionerna behöver vara tydliga, att man inte kan ge en för stor mängd information på en gång samt behovet av att korrigera instruktionerna om vissa steg saknas eller är felaktiga.

Matematiskt innehåll: Förflyttning mellan olika punkter i rymden (ett föremåls läge vid olika tillfällen).

Genomförande: Ställ ett föremål på golvet. Låt eleverna arbeta i par och undersöka vad det är som händer när man använder följande algoritm för att ta upp föremålet från golvet.

- Greppa tag om föremålet.
- Lyft föremålet.
- Ställ föremålet på bordet.

Låt eleverna diskutera vilka delar av dessa instruktioner som en dator inte skulle förstå. Exempel: vilket föremål, vad det innebär att greppa tag om, hur man lyfter, hur högt, vilket bord. Låt eleverna diskutera:

- Kommer algoritmen att fungera? Om inte, vad var det som blev fel?
- Hur kan en fungerande algoritm se ut istället?

Gå slutligen igenom ett eller flera av felen tillsammans. Du kan ställa följande frågor:

- Vad var problemet?
- Hur hittade ni det?
- Hur löste ni det?
- Har någon annan löst problemet på ett annat sätt?
- Var det något som var förvirrande?
- Vilka fel hittade ni och vilka åtgärder har ni tagit?

Variant: Utifrån elevernas förkunskaper och erfarenheter kan du variera instruktionerna så att det ska finnas flera steg eller innehålla flera ”fel” (t.ex. att instruktionerna kommer i fel ordning, någon instruktion är otydlig, något moment saknas).

Aktivitet 2 - Programmering med papper och penna

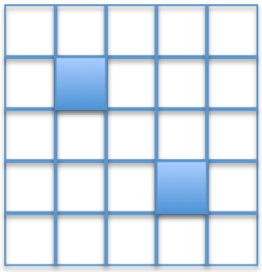
Utrustning: Penna, papper med ritade kvadrater som består av 5 x 5 rutor.

Förberedelser: Ta ett A4-papper och rita flera kvadrater som består av 5 x 5 rutor. Gör kopior efter mönster som anges i denna aktivitet eller skapa egna mönster. Dela olika mönster till elev A och till elev B. Tala om för eleverna att inte visa för varandra vilka mönster de fick.

Syfte: Att utveckla elevens förmåga att ge stegvisa instruktioner.

Matematiskt innehåll: mönster, förhållandet mellan olika punkter, koordinatsystem

Genomförande: Dela in eleverna i par där elev A ska vara programmerare och ge sin kamrat instruktioner. Elev B ska genomföra instruktionerna som programmeraren ger. Gör uppgiften två gånger så att båda elever får prova på att vara programmerare. Elev A markerar två rutor utan att visa elev B. Varje ruta betraktas som en pixel. Dessa rutor kan exempelvis placeras på följande sätt:

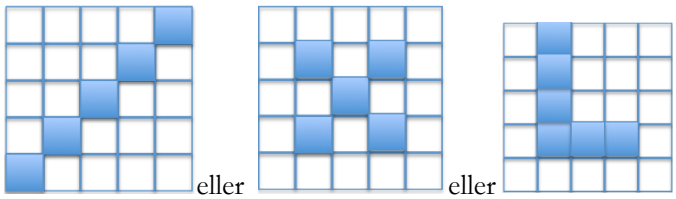


Elev B ska markera samma rutor utifrån As instruktioner. Uppmuntra eleverna att beskriva rutornas positioner med egna ord även om de kanske inte har det exakta vokabuläret. När båda är överens jämför eleverna sina inlagda rutor. Byt sedan roller.

Den här övningen fungerar mycket väl för alla åldrar och färdighetsnivåer och kan varieras i det oändliga. Man kan även göra den tredimensionellt med hjälp av exempelvis lego eller multilink. Det blir en riktig utmaning när eleverna får abstrakta bilder att jobba med. Då sätts förmågan att beskriva och ge stegvisa instruktioner verkligen på prov. Övningen kan enkelt varieras och ges i stigande svårighetsgrad. Nedan presenteras tre alternativ.

Alternativ 1

Elev A markerar flera rutor som bildar ett mönster. Exempelvis:



A ger instruktioner om var rutorna ligger utan att se på vilka rutor B markerar. B får ställa frågor och fråga fler gånger. A repeterar informationen tills B är nöjd.

Alternativ 2

Elev A lägger in flera rutor som bildar ett mönster. A ger information men endast då B ställt en fråga. B måste ta initiativ och vara aktiv annars får han eller hon ingen information.

Alternativ 3

Elev A markerar flera rutor som bildar ett mönster. A agerar sändare och ger instruktioner utan att interagera med B. I det här alternativet har B ingen möjlighet att ställa frågor eller reagera med kroppsspråk eller ljud för att visa att han eller hon inte förstått.

Oavsett vilket alternativ du väljer eller om du skapar nya mönster än vad som föreslås i denna text är det viktigt att samtala med eleverna:

- Vad är det som händer om man ändrar de blåa rutornas placering (t.ex. ett steg åt höger, vänster, upp eller ner)? Hur ändras instruktionerna? Finns instruktioner som inte fungerar?
- Hur kan ändringarna som eleverna föreslår uttryckas?
- Vad ser eleverna för mönster när de blåa rutornas förflyttas?

Aktivitet 3 - Algoritm – största talet

Utrustning: Papper och penna eller dator

Förberedelser: Välj en programmeringsmiljö om du använder dator. Se till att eleverna har samma program installerat på sina datorer.

Syfte: Att ge eleverna möjlighet att skapa en förståelse för dels vilka villkor en algoritm måste uppfylla, dels begreppet variabel.

Matematiskt innehåll: jämförelse av tal

Genomförande: Låt eleverna arbeta i par med att skapa en algoritm som frågar efter två olika tal och därefter anger det största talet. Varje elev skriver var sin algoritm. När båda

eleverna är klara med sina algoritmer byter de och skriver in koden i en programmeringsmiljö. Sedan diskuterar eleverna med varandra om algoritmerna fungerade. Eleverna ska rätta till algoritmerna om något har blivit fel. Gå slutligen igenom ett eller flera av felen tillsammans. Du kan ställa följande frågor:

- Vad var problemet?
- Hur hittade ni det?
- Hur löste ni det?
- Har någon annan löst problemet på ett annat sätt?
- Var det något som var förvirrande?
- Vilka fel hittade ni och vilka åtgärder har ni tagit?

Alternativ: Du kan variera uppgiften genom att låta eleverna skriva en algoritm som anger det största talet bland tre eller fyra olika tal.

Aktivitet 4 - Algoritm – jämna och udda tal

Utrustning: Papper, penna, dator

Förberedelser: Gör instruktionerna som eleverna ska använda. Kopiera och dela ut instruktionerna till varje par. Välj en programmeringsmiljö. Se till att eleverna har samma program installerat på sina datorer.

Syfte: Att ge eleverna möjlighet att skapa en förståelse för dels vilka villkor en algoritm måste uppfylla, dels begreppet variabel.

Matematiskt innehåll: jämna och udda tal samt aritmetiska uttryck

Genomförande: Låt eleverna arbeta i par med att skriva en algoritm som frågar efter ett godtyckligt tal och undersöker om talet är jämnt eller udda. Om talet är jämt ska datorn beräkna ett aritmetiskt uttryck. Om talet är udda ska datorn beräkna ett annat aritmetiskt uttryck. När eleverna är klara diskuterar de med varandra om algoritmen fungerade. Eleverna ska justera algoritmen om något har blivit fel.

Gå slutligen igenom ett eller flera av felen tillsammans. Du kan ställa följande frågor:

- Vad var problemet?
- Hur hittade ni det?
- Hur löste ni det?
- Har någon annan löst problemet på ett annat sätt?
- Var det något som var förvirrande?
- Vilka fel hittade ni och vilka åtgärder har ni tagit?

Diskutera med eleverna att enligt kraven på en algoritm bör den ha väldefinierade instruktioner, avslutas inom ändlig tid och ge ett resultat. Uppmuntra eleverna till att använda bra variabelnamn. Du kan även utmana dem till att försöka klara sig med endast x variabler.

Referenser

Blikstein, P., Worsley, M., Piech, C., Sahami, M., Cooper, S., & Koller, D. (2014). Programming pluralism: Using learning analytics to detect patterns in the learning of computer programming. *Journal of the Learning Sciences*, 23(4), 561-599.

Martinez, S. L. & G. Stager. (2013). *Invent to learn: Making, tinkering, and engineering in the classroom*. Torrance, CA: Constructing Modern Knowledge Press.

Thota, N. (2014). Programming course design: Phenomenographic approach to learning and teaching. In: *Proc. 2nd International Conference on Learning and Teaching in Computing and Engineering* (pp. 125-132). Los Alamitos, CA: IEEE Computer Society.