Programmering som språk

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.

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.

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.

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 åk 1-3 ligger fokus på att skapa denna typ av beskrivning genom att använda entydiga instruktioner.

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

Felsökning

Programmeringsprocessen avslutas med att eleverna tolkar resultatet och säkerställer att indata ger den utdata som var tänkt, d.v.s. 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 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 de exempelvis:

- förklara instruktionerna för någon annan
- kontrollera instruktionerna kritiskt
- göra en lista med möjliga fel
- anteckna vad de 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 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.

Dessa aktiviteter ger eleverna möjlighet att fundera över hur man ska göra för att få en instruktion att fungera. Dessutom finns det förslag på aktiviteter som fokuserar på att ge eleverna en inblick i användningen av olika typer av instruktioner (sekvens, alternativ, repetition) samt strukturen i en algoritm. Aktiviteterna kan göras muntligt eller med papper och penna.

Aktivitet 1 - felsökning

I denna aktivitet arbetar ni med instruktioner för att förflytta sig i rummet och äta en kaka. Till att börja med kan det vara fördelaktigt att eleverna får styra dig att utföra instruktionerna. Lägg en kaka på ett bord längst fram i klassrummet och ställ dig själv längst bak.

Uppgift A

Utrustning: Kaka, instruktionskort (gå framåt, sväng vänster, sväng höger, gå bakåt)

Förberedelser: Gör klart instruktionskorten (gå framåt, sväng vänster, sväng höger, gå bakåt). Beroende på elevernas förkunskaper och erfarenheter kan du även välja att låta eleverna formulera egna instruktioner antingen muntligt eller skriftligt.

Syfte: Att utveckla elevernas förståelse för i vilken ordning olika instruktioner ska ges samt att instruktionerna behöver vara tydliga.

Matematiskt innehåll: Förflyttning mellan olika punkter i planet (din position i klassrummet vid olika tillfällen).

Genomförande: Elevernas uppgift är att styra dig att gå fram till bordet. Dela in dem i mindre grupper och ge varje grupp instruktionskort med enkla instruktioner som till exempel ”gå framåt”, ”sväng vänster”, ”sväng höger”. Eleverna i varje grupp ska välja de instruktionskort som de behöver för att du ska kunna komma fram till bordet. Eleverna arbetar inledningsvis i olika grupper. Därefter låter du en grupp i taget ge dig

instruktionerna som gruppen har kommit överens om. Du genomför instruktionerna och därefter felsöker du, tillsammans med eleverna, de instruktioner som inte fungerade. Diskutera vad som ska ändras och varför. Låt därefter eleverna ge nya instruktioner, men nu med olika startpunkt – du startar där varje grupp är placerad. Målet är fortfarande detsamma, det vill säga att gå fram till kakan på bordet. På så vis blir instruktionerna för de olika grupperna olika och det är ett bra tillfälle att lyfta fram att olika program kan leda till samma mål.

Alternativ: En alternativ aktivitet kan vara att du har olika startpunkter, det vill säga att du befinner dig på olika ställen i klassrummet, och följer samma instruktioner. Resonera med eleverna om instruktionerna kommer att fungera eller om det blir något fel.

Uppgift B

Utrustning: Kaka, instruktionskort (gå framåt, sväng vänster, sväng höger, gå bakåt)

Förberedelser: Gör klart instruktionskorten (gå framåt, sväng vänster, sväng höger, gå bakåt). Beroende på elevernas förkunskaper och erfarenheter kan du även välja att låta eleverna formulera egna instruktioner antingen muntligt eller skriftligt.

Syfte: Att utveckla elevernas förståelse för i vilken ordning olika instruktioner ska ges samt att instruktionerna behöver vara tydliga.

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

Genomförande: Denna gång sitter du framför bordet med kakan på bordet. Elevernas uppgift är att formulera instruktioner, muntliga eller skriftliga, som är så tydliga att du ska kunna följa dem för att du ska kunna ta upp kakan, föra den till munnen och äta en bit. Även denna gång arbetar eleverna i grupper. Därefter väljer du några olika instruktioner som:

a) gör det möjligt för dig att äta en bit av kakan.

b) inte gör det möjligt för dig att äta en bit av kakan.

Du genomför instruktionerna och därefter felsöker du tillsammans med eleverna instruktionerna som inte fungerade. Diskutera även vad som ska ändras och varför. I detta sammanhang kan du även lyfta fram att en algoritm ger ett resultat och att olika algoritmer kan ge samma resultat.

Aktivitet 2 - felsökning

Syfte: Att utveckla elevernas förståelse för i vilken ordning olika instruktioner ska ges samt att instruktionerna behöver vara tydliga. Syftet är även att skapa en förståelse för att inte ge en för stor mängd information på en gång samt felsökningens betydelse eftersom instruktioner behöver korrigeras om vissa steg saknas eller är felaktiga.

Matematiskt innehåll: Förflyttningar.

Genomförande: Eleverna arbetar i par med att testa och ändra instruktioner. Eleverna ger instruktioner till varandra för att exempelvis resa på sig, gå fram till tavlan, gå fram till dörren, och så vidare.

Låt eleverna diskutera:

- Kommer instruktionerna att fungera? Om inte, vad var det som blev fel?
- Hur kan en fungerande instruktion 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?

Aktivitet 3 – felsökning

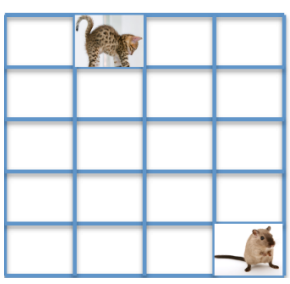
Utrustning: Kopior av bilden som presenteras i denna aktivitet, rutade papper, katten och musen.

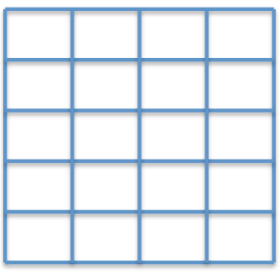
Förberedelser: Gör klart kopior efter bilden som presenteras i denna aktivitet, rutade papper, katten och musen. Repetition av höger och vänster bör göras innan aktiviteten.

Syfte: Att skapa förståelse för hur eleverna kan identifiera, ta bort och åtgärda fel.

Matematiskt innehåll: Förflyttningar och relationer mellan olika föremål i plan.

Genomförande: Låt eleverna undersöka om katten kan få tag i musen genom att följa exempelvis följande instruktioner:





Gå framåt ett steg.

Sväng höger.

Gå framåt två steg.

Sväng vänster.

Gå framåt två steg.

Diskutera med eleverna om instruktionerna kommer att fungera eller om det blir något fel samt hur en fungerande instruktion kan se ut istället. Beroende på elevernas förkunskaper och erfarenheter kan du även låta eleverna placera katten och musen på andra ställen, än vad som föreslås i aktiviteten, på det rutade papper. Kan instruktionen ges på ett annat sätt? Ska eleven räkna den ruta katten står i? Går katten automatiskt ett steg framåt när katten svänger eller står katten kvar i samma ruta och svänger?

Alternativ: Du kan göra en liknande uppgift rent fysiskt i klassrummet eller som en utomhusaktivitet. Du kan även ändra instruktionerna eller placera katten och musen på andra ställen på det rutade papper.

Aktivitet 4 – repetition

Utrustning: Instruktioner som eleverna ska testa.

Förberedelser: Skriv ut instruktionerna. Dela eleverna i par och ge till varje par instruktionerna som ska testas.

Syfte: Syftet med denna aktivitet är att eleverna ska få syn på hur repetition, det vill säga slingor/loopar, kan användas i programmering samt att varje algoritm måste avslutas efter ett antal ändliga steg.

Matematiskt innehåll: Enkla mönster.

Genomförande: Låt eleverna arbeta i par för att testa följande instruktioner:

- Upprepa fyra gånger: gå ett steg framåt och hoppa en gång.
- Upprepa tre gånger: säg ett tal.
- Upprepa två gånger: rita en triangel.

Låt eleverna diskutera vad som händer om man inte anger antal upprepningar och vilket mönster de urskiljer. Välj några mönster och diskutera med eleverna vad som upprepas i varje mönster. Låt därefter eleverna skapa olika mönster genom att använda upprepningar. Exempel på en uppsättning instruktioner kan vara:

Upprepa fyra gånger: gå tre steg

vänd 90^0 åt höger

Motsvarande mönster är en kvadrat.

Aktivitet 5 – villkorssats

Utrustning: Instruktioner som eleverna ska testa.

Förberedelser: Skriv ut instruktionerna. Dela eleverna i par och ge till varje par instruktionerna som ska testas.

Syfte: Att skapa en förståelse för användningen av alternativ/villkorssatsen samt vikten av att varje instruktion i en algoritm är precist definierad.

Matematiskt innehåll: förflyttning, jämna och udda tal

Genomförande: Låt eleverna arbeta i par och genomföra följande instruktioner:

- *Om* du har svarta byxor *så* ska du gå två steg framåt, *annars* ska du snurra runt ett varv.
- *Om* du fyller år på ett jämnt datum så ska du räcka upp handen *annars* ska du klappa händerna en gång.

Alternativ: Beroende på elevernas erfarenheter och förkunskaper kan du variera det matematiska innehållet som tas upp i varje instruktion.

Aktivitet 6 - skapa egna felsökningsövningar

Syfte: Att skapa egna instruktioner.

Matematiskt innehåll: kan variera

Genomförande: Låt eleverna arbeta i par och skapa egna instruktioner för att utföra till exempel en uppgift som de ofta gör i sin vardag. De kan själva välja vilken uppgift de vill programmera. Då parens program är klara, får de testa dem på klasskamraterna för att se om programmen fungerar eller vad de behöver ändra på.

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.

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.