Teknik – Grundskola åk 7-9
Modul: Den digitala världen årskurs 7-9
Del 4: Programmering 1

Programmering 1

Lars Björklund, Linköpings universitet

Inledning

Ett innehåll i kursplanerna för teknik som fått ett nytt och starkare fokus än tidigare är datorn, dess funktion och användning i tekniska lösningar. I det centrala innehållet för 7-9 nämns tekniska lösningar som utnyttjar elektronik och hur den kan programmeras. Vidare hur programmering kan används till styrning och reglering bland annat med hjälp av sensorer. I det centrala innehållet för årskurs 4-6 nämns att några av datorns delar och deras funktioner ska behandlas exempelvis processor och arbetsminne samt hur datorer styrs av program och hur de kan kopplas samman i nätverk.

Den här första programmeringsdelen i modulen ska behandla datorn och programmering och de utmaningar man kommer att möta som lärare, det vi skulle kunna kalla data- eller programmeringsdidaktik:

- Val av dator, programmeringsmiljö och språk.
- Om att lära sig programmera.
- Vilka utmaningar kommer jag att möta som lärare?
- Vilka problem har elever när de börjar programmera?
- Vad är lämpligt att ta upp inom teknikämnets ram?

Programmering som centralt innehåll återfinns även i matematikämnet och många elever har, när de kommer upp i årskurs 4 respektive 7, stött på någon form av programmering. De har kanske programmerat enkla robotar eller tillverkat spel i Scratch. Ingen räknar dock med att eleverna, inte ens i årskurs 9, ska ha blivit duktiga programmerare. Den tid som skulle behöva avsättas för detta finns inte, varken i matematik eller i teknik. Grundläggande förståelse för enkla program och exempel på hur variabler, sekvenser, villkorssatser och repetitionssatser fungerar får vara målet för undervisningen. I teknikämnet har vi ett speciellt fokus på att kommunicera med yttre elektriska och elektroniska komponenter som sensorer, lysdioder, displayer, motorer etc. Dessa komponenter behandlas i del 3 Grundläggande Elektronik 1 och används i del 5 Programmering del 2 - Mäta, detektera, styra och reglera.

Texten i den här och följande delar är tänkt som en hjälp för lärare att arbeta med det centrala innehållet i teknik där elektroniska komponenter och programmering ingår som lösningar på ett antal specifika tekniska problem. Grundläggande programmeringsinstruktioner introduceras när de behövs och med en progression som bygger på datadidaktisk forskning. Modulen är således inte en komplett programmeringskurs utan gör nedslag och visar på exempel där en dator bidrar till moderna

och effektiva lösningar. Texten är inte en nybörjarkurs i programmering utan är till för att ge lärare didaktiska råd och exempel på hur man kan lägga upp sin undervisning. Lärare rekommenderas därför att genomföra Skolverkets webbaserade kurser ”Om programmering” och ”Att programmera” eller andra liknande satsningar.

Val av dator, programmeringsmiljö och språk

I teknikämnet är vi intresserade av dels en dators olika delar och hur de fungerar, dels av hur vi med yttre komponenter, sensorer, motorer med mera kan konstruera olika styr-, mät- och reglerlösningar. Kanske vill vi också pröva olika sätt att kommunicera med vår dator och andra datorer, mobiltelefoner, sensorer och displayer genom till exempel Bluetooth, Wi-Fi, USB eller andra sätt.

De flesta datorer som vi möter i vardagen är enkortsdatorer, ofta inbyggda i andra tekniska system. Om vi arbetar med enkortsdatorer, kanske en Arduino, Microbit eller en Raspberry Pi, kan vi lära oss något om en dators innersta delar, minne, in- och utportar och vi har dessutom möjlighet att ansluta yttre sensorer och styra motorer. Att använda en PC, datorplatta eller en Chromebook och ett program som Scratch är visserligen ett vanligt sätt att prova på programmering men den typ av styrning och reglering man kan pröva på i den miljön uppfyller inte alla de krav vi ställer i teknikämnet. Scratch är ett digitalt verktyg för att hantera bilder och ljud på en skärm och ger mycket lite förståelse för en dators hårdvara och funktion eller hur yttre elektroniska komponenter samverkar med ett program. Här vill vi istället att eleverna först skriver ett program på en PC eller motsvarande dator för att sedan föra över det färdiga programmet till en liten enkortsdator. Därefter exekverar, utför, enkortsdatorn programmet. Man kan koppla loss den programmerade enkortsdatorn, ansluta den till ett batteri, bygga in den i andra mekaniska konstruktioner och i samband med detta diskutera användningen av alla de ”inbäddade” datorer vi har omkring oss i olika styr- och reglerutrustningar. Inte minst intressant är de enkortsdatorer som är uppkopplade mot internet i det vi kallar ”Internet of Things” (IoT). Vi ser nu hur allt fler av hemmets tekniska lösningar; kylskåp, spisar, värmesystem, larm, brödrostar och kaffemaskiner datoriseras och kopplas upp mot internet. IoTdatorerna ska kunna kommunicera med molnbaserad lagring, kanske kunna fjärrstyras från en mobil och inte minst kunna uppgraderas av den som har tillverkat densamma. Naturligtvis finns också en risk att den innehåller felaktig och skadlig kod och kan ”kidnappas” av kriminella element, till exempel för att utföra cyberattacker. Att själv ha fått bygga liknande system kan ge en förståelse för hur lätt det är att göra misstag och införa risker i vår digitala vardag.

Till alla de tre tidigare nämnda enkortsdatorerna finns såväl block- som textbaserade programmeringsspråk. De blockbaserade språken har instruktioner för att läsa av och kontrollera portar, de kan läsa såväl digitala signaler som analoga och har i vissa fall färdiga funktioner för att läsa av specifika sensorer och styra motorer av olika slag. Enkortsdatorn, eventuellt kompletterat med en tillsatsmodul, kan fås att kommunicera med hjälp av olika typer av radiobaserade tekniker. Det innebär att det går att nå alla våra mål i teknikämnet med ett blockbaserat språk. I modulen använder vi för såväl år 4-6 som år 7-9 ett

blockbaserat språk. De som vill fördjupa sig i programmeringen kan komplettera med ett textbaserat språk som Javascript och Python. Dels kommer man närmare hårdvaran och förståelsen av vad varje programinstruktion gör, dels är det lättare att skapa, felsöka och dokumentera större program.

I den här och de följande två delarna om programmering diskuterar och använder vi mycket korta och enkla program och de flesta exempel som demonstreras använder Microbitdatorn. Microbit har ett antal för teknikämnet goda egenskaper: den är billig och ganska robust, är liten och kan byggas in i andra konstruktioner. Den har ett antal inbyggda sensorer och det är enkelt att ansluta yttre elektronik. Det finns ett flertal programmeringsspråk varav minst ett blockbaserat språk som dessutom kan automatöversättas till ett textbaserat språk (Javascript), vilket medger en smidig progression. För de mer avancerade projekten finns också språket Python som har blivit vanligt för undervisning i programmering. I webeditorn för Microbit finns också en simulator vilket gör det möjligt att provköra programmen utan att ansluta själva enkortsdatorn. Eleverna kan på valfri datorplattform via en webbläsare både skapa och testa sina program för att sedan ladda ned koden till enkortsdatorn.

Microbit togs fram av engelska BBC och delades ut till skolbarn i England utan kostnad för skolorna. Även Danmark har valt en sådan lösning. Det har vuxit fram en rik flora av resurser som lärare kan använda i sitt arbete och fler tillkommer hela tiden. Gå gärna in och botanisera bland de resurser som finns på nätet:

https://makecode.Microbit.org/

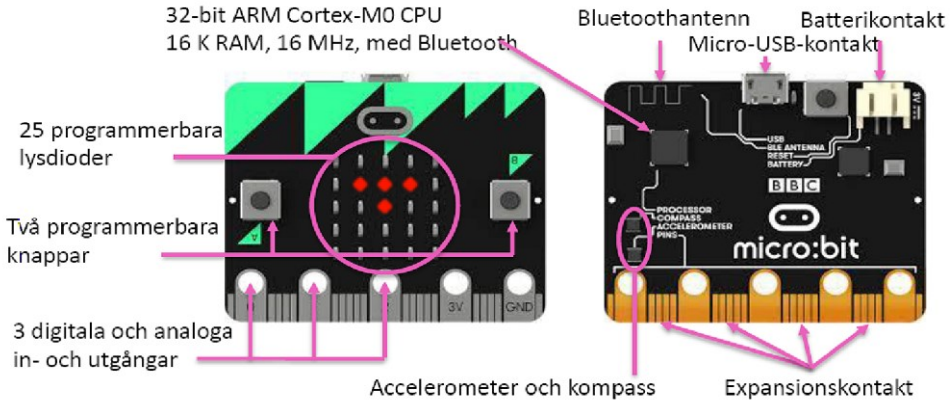
http://www.microbit-i-skolen.dk

http://www.mermicrobit.se

https://www.Microbit.co.uk/app/

https://Microbit-playground.co.uk/howto/

https://sites.google.com/edu.trelleborg.se/programmering/årskurs-7-9/3-resurser



Övningarna i modulens exempel går oftast att provköra i Microbits simulator om man inte har tillgång till just den enkortsdatorn. Detta gör det möjligt för alla att följa den här texten och utföra övningarna även om skolan tidigare har investerat i andra enkortsdatorer, som LEGO, Arduino eller Raspberry pi.

Om att lära sig programmera, datadidaktik

Datalogiskt tänkande

I Sverige har det av Jeanette Wing (2006) introducerade begreppet ”Computational Thinking” översatts till ”Datalogiskt tänkande”. Fredrik Heintz, Linda Mannila, Karin Nygårds, Peter Parnes och Björn Regnell (2015) beskrev begreppet och berättade i en konferensartikel om tidiga försök i Sverige att införa mer datavetenskap och programmering i skolan. De, liksom många med dem, menar att datalogiskt tänkande är en generell förmåga, ett sätt att tänka, som kan utvecklas bland annat genom att man lär sig att programmera och att denna förmåga har stor betydelse och värde för många av skolans ämnen. Idag finner man många olika definitioner av begreppet datalogiskt tänkande men den ursprungliga definieras som förmågor uppdelade i fyra underkategorier:

- Att bryta ned ett problem i mindre delar
- Att hitta mönster
- Att skapa abstraktioner genom att generalisera de identifierade mönstren
- Att skapa algoritmer

Kritik finns dock mot själva idén om att det finns en generell metod för att lösa problem och att då ovanstående beskrivna förmågor skulle kunna övas i en kontext, exempelvis programmering, och att de sedan skulle överföras till att lösa problem inom andra områden. Datalogiskt tänkande utvecklas efter lång tids praktik och erfarenhet inom ett specifikt problemområde menar nämligen ett par av de forskare som har följt programmeringsundervisning (Perkins & Salomon, 1989).

När vi studerar läromedel om programmering ser vi flera olika försök att konkretisera begreppet datalogiskt tänkande. I kodboken.se förklarar man vikten av att en dator måste få exakta och noggranna instruktioner och exemplifierar med att programmera en robot att följa ett matrecept: (https://www.kodboken.se/start/fattakoden/datalogiskt-tankande/tank-datalogiskt) Eleverna får fundera över vilka instruktioner som behöver beskrivas tydligare: ”grädda pannkakan lagom länge” måste ju t.ex. definieras mer exakt. Det är viktigt att förstå att datorn inte kan tolka in instruktioner som inte står i programmet. Exemplen i detta och andra läromedel utgår ofta från en färdig lösning där uppgiften är att skriva tydliga instruktioner/algoritmer. Det vi då vill att eleven ska förstå är dels att datorn har en mycket begränsad mängd instruktioner och att till synes enkla uppgifter därför måste beskrivas in i minsta detalj. I mänskligt tal har vi mycket fler och fullödigare sätt att beskriva en handling och mycket behöver inte berättas, det är implicit (Du Boulay, 1986).

Den programmeringsstil som presenteras i läromedlen bygger ofta på sekvenser av instruktioner som datorn exekverar en efter en. Den som är nybörjare inom programmering har ofta ingen övergripande plan för hur programmet ska se ut och fungera utan börjar i detaljerna, det som kallas ”Bottom-up”. I exempelvis ett Scratchprogram vill man att en figur, en så kallad sprite ska röra sig framåt, kunna ge ljud från sig och så vidare. Man bygger en kedja av händelser som ska utföras i en följd, eventuellt kompletterad med några villkorliga hopp eller att en del av programmet ska köras flera gånger (en så kallad loop). I ett programspråk som Scratch där programmen ofta placeras i varje enskild sprite leder detta ofta till ett stort antal utspridda ”programskript” och det kan bli svårt att läsa, förstå och felsöka i programmet.

En programmeringsstil som introducerades för nästan 50 år sedan, ”Top-down” försökte skapa struktur och överblickbarhet i ett program. För att klara det krävdes mycket erfarenhet i att se problemet i olika hierarkiska nivåer och att känna igen mönster på olika nivåer. Inte ens experter lyckades alltid med detta utan arbetade ofta först med att lösa delproblem där de redan hade färdiga algoritmer.

Hur utvecklar vi elevernas programmeringskompetens?

I delen om Grundläggande elektronik gick vi närmare in på teorier om problemlösning och hur man bör bygga upp en verktygslåda av lösningar till olika problem. I den här och kommande delar presenteras därför ett stort antal lösningar på små tekniska problem. Som kommer att påpekas senare i texten kan man försöka få eleverna att dela upp sina program i mindre delar med avgränsade uppgifter, så kallade subrutiner (funktioner och/eller procedurer) med relevanta namn och att namnge sina variabler så att man underlättar förståelsen av programmet. Det är vanligen svårt att läsa, förstå och felsöka i någon annans program och det är ju en uppgift som ofta hamnar hos lärare.

I flera av dagens läromedel om programmering framhålls fyra användbara begrepp; Sekvens, Alternativ, Repetition och Abstraktion och dessa förkortas ofta till SARA:

Sekvens handlar om tågordning och tydlighet. Ett program kan ses som en lista av tydliga instruktioner som kommer att utföras i den ordning man har skrivit in dem.

Alternativ handlar om att ett program kan utföra olika uppgifter beroende på vilka villkor och förutsättningar som gäller. Instruktioner kan ha namn som: IF THEN ELSE (Om, Då, Annars).

Repetition handlar om att programmet upprepar instruktioner. Antal upprepningar kan vara förutbestämt eller styras av en villkorssats: REPEAT WHILE LOOP (Repetera, Medans, Slinga).

Abstraktion. Handlar om att vi kan skapa egna generaliserbara lösningar med egna namn som vi senare kan använda i programmet. Dessa kallas för funktioner, metoder, subrutiner eller något annat och kan förekomma med eller utan parametrar.

De här fyra grupperna av instruktioner kan upplevas enkla men forskning kring nybörjares problem och missförstånd vid programmering har visat att det bara är sekvenser som är lätta att lära sig. Såväl repetitionssatser, villkorssatser och förmågan att skapa algoritmer har visat sig vara svårt för nybörjaren. Nu följer därför en översikt av problem och missförstånd som de senaste 50 årens forskning har identifierat hos nybörjare när de försöker programmera. Innehållet är viktigt både när man planerar sin undervisning och när man hjälper elever att felsöka i sina program.

Det digitala verktyget: ”a notional machine”

Elever måste förstå sig på det aktuella digitala verktyg de ska programmera eller hantera. Verktyget brukar kallas ”notional machine” (teoretisk maskin) på engelska och man behöver förstå vad det digitala verktyget kan utföra, vilka funktioner det har. Man behöver förstå hur man styr och kontrollerar dessa funktioner samt naturligtvis behärska själva syntaxen (reglerna för hur ett program ska sättas samman för att vara korrekt) och grammatiken i programmeringsspråket (Sorva, 2013). Vissa författare menar att förståelsen av slutanvändaren och dennes användning av programmet också ingår i begreppet ”notional machine”. Hur man kan stödja en användare att styra och förstå programmet är en viktig del av programmeringen och som kallas ”människa-maskin-gränssnitt”.

Begreppet digitalt verktyg används ofta i våra nya kursplaner. Kalkylprogram, olika rit- och CAD-program, webeditorer med mera är typiska exempel på en ”notional machine”. I ett kalkylprogram matar vi in data, vanligen i form av tal som sedan kan bearbetas av de matematiska och statistiska funktioner som kalkylprogrammet erbjuder. Dessa funktioner kan användas enskilt eller sammanfogas i enkla program. Utsignaler från detta system är tabeller av data eller olika typer av grafiska representationer. Att låta elever använda ett kalkylprogram kan således falla under rubriken programmering. I ett senare beskrivet programexempel samlas mätdata in från en sensor och data bearbetas och presenteras

sedan i ett kalkylprogram. Det digitala maskineri som ligger under ytan och gör att kalkylprogrammet fungerar är det vi vanligen kallar en dator. I teknikämnet är vi intresserade av denna dator, vad den kan och hur den kan styras och vi kommer att använda den för att ”skapa digitala verktyg” som löser olika tekniska problem som att styra, mäta och reglera.

Ett digitalt verktyg som ofta används i skolan är programmeringsmiljön Scratch. Det kan hantera bilder, så kallade sprites, som kan flyttas runt på bildskärmen, det kan spela upp ljud och utföra en del enklare matematik och logik. Det kan avläsa tangentbordet och musen, lyssna efter ljud via en mikrofon och kan känna av när en sprite krockar med ett annan sprite eller ett annat objekt som exempelvis bildskärmens kant. Verktyget kan användas för att skapa enklare spel och grafiska berättelser. I programmeringsspråket finns instruktioner för att styra och kontrollera allt detta och dessutom ett antal matematiska och logiska funktioner. Klassiska programmeringsinstruktioner finns naturligtvis för att styra programmets flöde; Sekvenser, Alternativ, Repetitioner och Abstraktioner (SARA). Programmeringen sker genom att man väljer instruktioner i form av block ur en meny. Blockens utformning gör att de bara kan placeras på ett sätt i programmet och att man undviker felstavningar eller andra syntaktiska fel. Däremot kan man fortfarande placera block i fel ordning eller glömma viktiga delar vilket medför att programmet inte utför det man vill göra. Instruktionsuppsättningen är väl anpassad till användningsområdet vilket gör det lätt att komma igång och att snabbt producera intressanta resultat på datorns skärm. Scratch har en låg ingångströskel vilket förklarar dess popularitet. Det är dessutom möjligt att skapa mer komplicerade program, det har djup och bredd, även om några svagheter då märks mer tydligt. Det är svårt att felsöka eller ”avlusa” ett program och då programmen ofta splittras upp i många sprite är det ibland svårt att läsa och följa vad som händer i ett program.

När forskare (Aivaloglou & Hermans, 2016; Fields, Giang, & Kafai, 2014) studerade program som elever skapat i Scratch, och det finns miljontals exempel på nätet, fann man att programmen vanligtvis var mycket korta och att eleverna främst hade använt instruktioner för att flytta objekt och mer sällan använt de instruktioner (SARA) man hoppades att de skulle öva på. Programmen innehöll ofta ”död”, oanvänd, kod och eleverna verkade ofta bara ha kopierat andras program och modifierat några parametrar. De skapade program utan att ha en övergripande plan och när man frågade varför de tycker om programmet visade det sig att det ofta var möjligheten att rita egna kostymer på figurerna och att spela in egna ljud som var roligt. Sammantaget verkar Scratch passa väl som ett digitalt verktyg i bild eller svenskämnet men det räcker inte till för att uppnå teknikämnets alla mål. Det finns dock projekt där enskilda programmerare har utvidgat programspråket Scratch för att kunna styra anslutna enkortsdatorer som Arduino (http://s4a.cat) med flera. Dessa lösningar kräver dock lite mer avancerade kunskaper hos användaren. De fördelar som blockprogrammering ändå har kommer vi att utnyttja i modulens olika exempel men då med Microbits blockprogrammeringsmiljö. Systemet Microbit, med dess hårdvara och

olika programmeringsmiljöer, bildar den ”notional machine” som vi ska arbeta med för att nå flera av teknikämnets mål och centrala innehåll.

Novisens problem med att lära sig programmera

De problem och misstag som nybörjaren gör när de programmerar uppmärksammades redan för flera decennier sedan då man utvärderade tidigare satsningar på programmering i skolan. En grupp av fel som är vanligt förekommande kallas ”Hidden Minds Superbug” (Pea, Soloway, & Spohrer, 1987) och beror enligt författarna på att eleven tillerkänner datorn intelligenta och tolkande egenskaper. Att datorn vet vad som har skett tidigare och vad som kommer att ske och inte bara den instruktion som just nu utförs. Eleverna påstår själva att de inte alls tror att datorn är så ”smart” men empirin verkar tyda på att de när de skriver sina program ofta gör fel av den typen. Två huvudgrupper av dessa fel har identifierats: samtidighet och tron på att datorn vet vad vi egentligen menar med vårt program.

Samtidighet

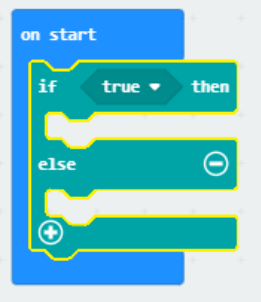
Novisen uppfattar ibland, felaktigt, att datorn exekverar (utför) alla instruktioner samtidigt, och därmed att det räcker att införa exempelvis en villkorssats en gång för att den ska utföras hela tiden (Pea, et al., 1987). Troligen beror det på en sammanblandning mellan datorspråkens ”IF” och hur man i vanligt tal använder en villkorssats. När exempelvis läraren säger ”OM du fuskar så blir du underkänd på prov” behöver detta inte upprepas fler gånger för att gälla senare under dagen. Datorprogrammets ”if (Ljusstyrka > 100)” testas dock bara en gång och måste upprepas, till exempel läggas inuti en repetitionssats, för att variabeln ska testas kontinuerligt. Det här är ett fel som även äldre elever gör sig skyldiga till och beror på att man i dagligt tal inte är så strikt sekventiell. En del av en mening kan ha verkan även senare under kommunikationen och behöver inte upprepas. Ett motexempel är ett matrecept där man ju utför receptets instruktioner i en strikt ordning. De komiska filmklipp på nätet där klasser instruerar sin lärare att bre en smörgås är tankeväckande, men kom ihåg att påpeka allt det som människor gör utan explicita instruktioner och hur svårt det är att tänka på allt som är självklart, när man utför instruktionen.

Datorn tolkar vad vi egentligen menar

En annan effekt av vår vana att kommunicera med andra människor är att vi antar att den som lyssnar inte tolkar oss bokstavligt utan försöker förstå meningen/intentionen av det vi säger. Elever uppvisar ofta, inte bara när de programmerar, en tro på att deras eget perspektiv, deras intentioner är tydliga för andra och därför även för datorn (Sleeman, Putnam, Baxter, & Kuspa, 1986). De glömmer därför, när de programmerar, att skriva instruktioner, namn och ingångsvärden på variabler och annat viktigt i programmet och förutsätter att datorn kan fylla i på samma sätt som en annan människa skulle kunna göra. De kan använda samma variabelnamn på olika variabler så att X i en del av programmet representerar en sak och senare något annat. När de försöker hitta fel i programmet verkar de i tanken ”fylla i” delar som inte står där och har därför svårt att hitta felet.

Missförstånd som bygger på hur vi i vanligt tal använder ord och begrepp

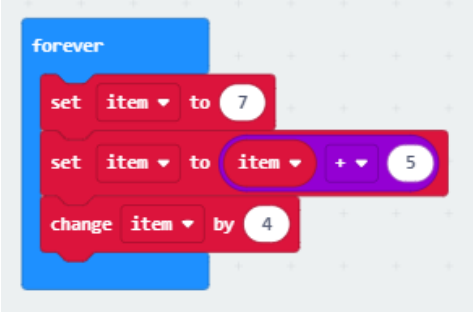
När man i dagligt tal använder konstruktionen ”IF” är ofta ”ELSE” delen underförstådd medan man i de flesta datorspråk måste vara tydlig med vad som ska hända om villkoret inte är uppfyllt. Blockspråkens sätt att införa dessa instruktioner innehåller en automatisk avslutning och explicit ELSE-del. Det underlättar förstås för eleven, men de får inte heller chans att lära sig att det är ett krav vid programmering i ett textbaserat språk. Även ordet THEN kan misstolkas, i vardagligt tal betyder det ju bara att något händer efter något annat (Du Boulay, 1986).



Problem att förstå vad en variabel är

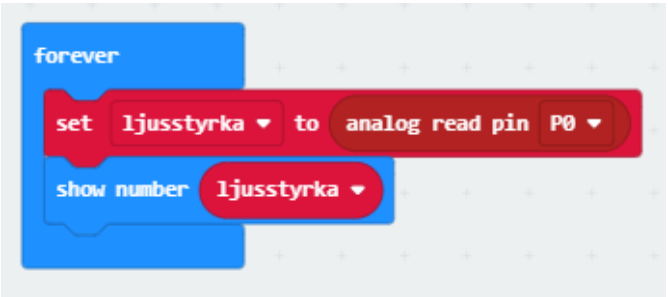
Variabler är, kanske lite förvånande, något som kan ge upphov till problem för nybörjaren (Du Boulay, 1986). Analogin med lådor där tal kan förvaras som ofta används i läromedel kan leda till att eleven tror att mer än ett tal kan läggas i samma låda. Lämpligare analogi kan vara att se en variabel som en suddbar skrivplatta där datorn kan skriva ett tal eller en text.

```
item = 7
item = item + 5
item += 4
```



Två exempel som tilldelar och bearbetar variabler, det första i språket Javascript det andra i blockmiljön för Microbit. I exemplen ovan läggs först värdet 7 i variabeln item, sedan adderas talet i item med 5 och resultatet läggs sedan tillbaka i variabel item. Det tredje exemplet använder en speciell funktion i Javascript som adderar ett tal till variabelns tidigare värde. På svenska heter det tredje blockkommandot ”ändra item med talet 4”. Det är möjligen enklare att förstå på svenska men i exemplen i modulen använder vi den engelska versionen eftersom det då blir lättare att gå över till att programmera i textbaserade språk.

Det verkar enligt några studier som om programexempel där man läser in värden från en sensor via en ingång (port eller pin) ger mer förståelse för hur man använder en variabel. Värdet som kommer att läggas i variabeln är ju okänt vid själva programmeringstillfället. I



programmet till höger läses ett värde in från ingången P0 som blir värdet på variabeln ljusstyrka. Det värdet ska sedan visas via microbitens lysdioder.

Programstruktur och planering

Experten med sin mycket större erfarenhet av att programmera kan arbeta ”Top-down” och lägger ofta mycket tid på att formulera och planera sitt program innan han/hon börjar koda. Experten tänker inte i enstaka instruktioner utan i större funktionella block och kan därför strukturera ett program mycket bättre. De kan känna igen mönster i ett problem och bryta ned problemet i mindre delar som kan lösas med kända algoritmer. Att begära att noviser skulle kunna lära sig ”Top-down” programmering utan att ha expertens stora erfarenhet är inte möjligt. Vi kan dock som lärare hjälpa eleverna genom att låta dem skapa tydliga och förklarande namn på sina variabler och låta dem skapa subrutiner eller funktioner som kan testas var för sig och sedan anropas från ett huvudprogram.

Att läsa och avlusa ett program

Alla forskare är överens om att det är svårare att läsa och förstå ett program än att skriva det. Det innebär att speciellt nybörjare har stora svårigheter att hitta fel i sina program (Rodrigo et al., 2013). Även som lärare kommer man ibland hamna i svårigheter när man felsöker en elevs program. Att lämna sin egen förståelse av vad man ville att programmet skulle göra och att sätta sig in i hur det aktuella datorsystemet tolkar programmet är svårt. Experter verkar använda sin erfarenhet av hur program brukar se ut, var och hur variabler deklareras, i vilken ordning funktioner presenteras etc. De har en mönsterigenkännande förmåga som hjälper dem att fokusera på delar av ett program där ett fel kan finnas. Novisen försöker däremot att steg för steg följa programmets flöde vilket nästan är omöjligt i exempelvis ett programspråk som Scratch. Man ser också hur elever vanligen raderar och skriver om hellre än att de modifierar sina program. Att tillsammans läsa och diskutera färdiga program kan vara ett sätt att träna upp den här förmågan. För läraren är det av värde att ha kunskap om vilka svårigheter en nybörjare brukar ha.

Progression

Många studier har försökt finna i vilken ordning man ska införa nya begrepp och programmeringsinstruktioner. Det är mycket viktigt att man ger eleverna god tid och mycket stöd i början för att de inte ska köra fast. När forskare som exempelvis Robins (2010) försökte förklara varför så många studenter misslyckades i sin första programmeringskurs antog man först att de saknade talang, vissa kunde bli programmerare andra inte. Nu förklarar man detta med att programmering är så komplext och att man måste behärska så många nya detaljer att en del studenter ger upp om det går för fort, stödet från lärare är för litet och ibland att relevanta, motiverande exempel saknas (Robins, 2010; Rodrigo, et al., 2013). Forskningen visar som tidigare nämnts att loopar, repetitionssatser och villkorssatser är svåra att förstå men även att elever har svårt att hantera variabler. Det verkar således lämpligast att börja med det som är lättast att lära sig, enkla sekvenser, utmatning och styrning. Det finns studier (Bagcı, Kamasak, & Ince, 2017; Lavonen, Meisalo, Lattu, & Sutinen, 2003; Pásztor, Pap-Szigeti, & Lakatos Török, 2010;

Rubio, Romero-Zaliz, Mañoso, & de Madrid, 2015) som pekar på att det underlättar när eleverna får programmera något extern fysiskt till exempel en robot med sin enkortsdator (physical computing). Hur de när de får se och uppleva hur datorn konkret reagerar på ändringar i yttre signaler från en kontakt eller sensor och när datorn kan styra externa komponenter som motorer och lampor lättare tar till sig och förstår programmeringsinstruktioner. Möjligheten att läsa in ett värde från en strömbrytare eller en sensor och placera den i en variabel underlättar förståelse vad en variabel är. Att i en villkorssats vänta på att en signal från en ljuskänslig detektor ändrar sitt värde innan man slår till eller från en lampa verkar underlätta förståelse av repetitionssatser och villkorssatser. I bilden finns ett exempel för Microbit där den inbyggda ljussensorn avläses i en villkorssats. Den interna ljussensorn (light level) avläses och värdet jämförs med talet 128 varefter programmet tar en av två vägar och matar ut antingen 1 (+3V) eller 0 (0V) på utgång P0 där en lysdiod är inkopplad så att en operatör kan se om det är ljus på sensorn. I simulatorn kan man ändra infallande ljus och se vad som händer på utgång P0. Genom att klicka och dra i halvmånesymbolen ändrar man ljusstyrkan i simulatorn.

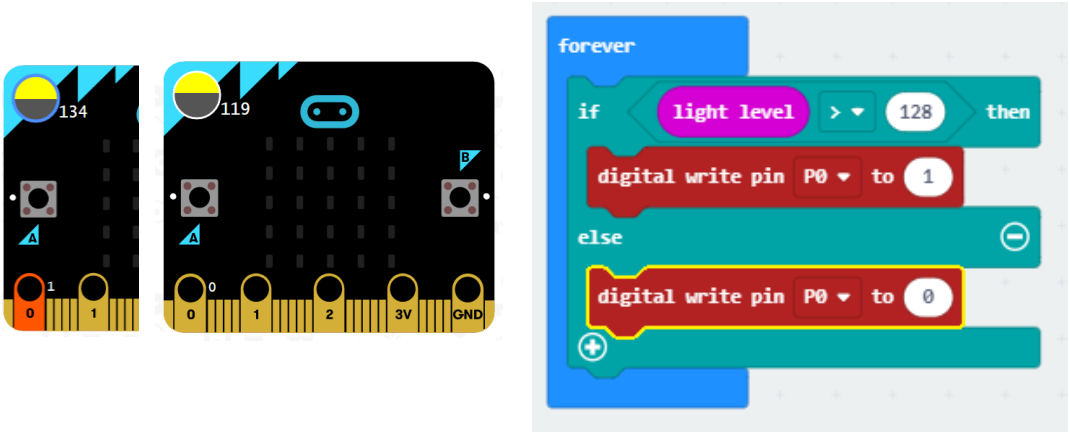


Bild på simulatorn, med lågljusintensitet (119) respektive hög (134) samt blockprogrammet.

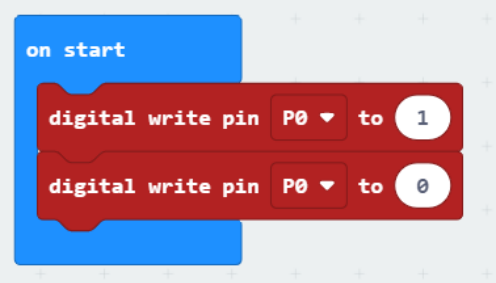
En progression i undervisningen som prövats vid arbete med enkortsdatorer och styrning av robotar beskrevs av Sparkes (1995). Forskaren arbetade tillsammans med en kvinnlig lärare och 12-åringar i en flickskola. Den valda progressionen verkade fungera bra och när man genomförde eftertester fann man att de flesta elever kunde klara av att förstå de första stegen i progressionen men att ingen klarade testerna på den högsta nivån. Eleverna hade svårt, trots mycket stöd, att planera sina program: de arbetade ”Bottom-up” och ganska ostrukturerat. När de fick sitta vid datorn och fick direkt ”återkoppling” av datorn på sina fel klarade de av svårare uppgifter än när de arbetade med papper och penna. Författaren rekommenderade användning av grafiska och ikonbaserade språk för att minska syntaxfelen i programmen.

Den plan de använde sig av ser, efter författarens översättning, ut så här:

- Att slå på och av digitala utgångar
- Skapa en tidsfördröjning
- Tillverka en procedur eller funktion, som sen anropas
- Repetera en händelse för att exempelvis blinka en lampa
- Använda flera funktioner för att strukturera sitt program
- Använda repetitionsfunktioner av olika slag
- Läsa av digitala ingångar och lagra i variabler
- Använda villkorssatser IF THEN ELSE, med digitala ingångars signaler

Nu följer några exempel på hur man kan arbeta utefter denna plan med en Microbit. De flesta exemplen går att testa i simulatorn så man behöver ingen ansluten hårdvara. Vill man koppla in lysdioder till enkortsdatorn finner ni konkreta kopplingar i övningsbilagan.

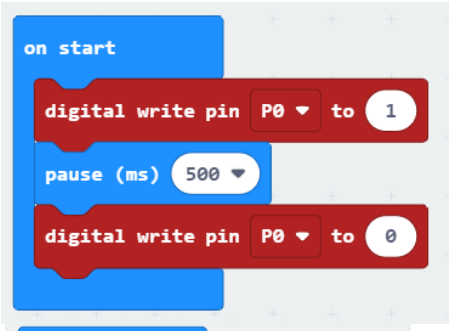
Att slå på och av digitala utgångar

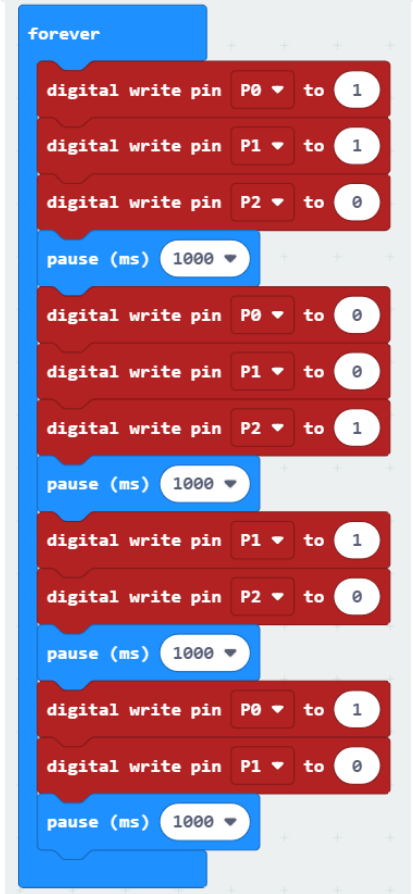


I många styr och reglertillämpningar behöver man slå till eller från något, det kan vara en lampa eller en motor som driver en fläkt eller pump. I Microbit finns ett kommando ”digital write pin” som lägger ut det binära talet 0 (ca 0V) eller 1 (ca 3,3V) på någon av de 21 utgångarna (P0-P20). P0, P1 och P2 är de utgångar som kan anslutas med krokodilkontakter. Strömmen på utgången (5mA) räcker för att tända en lysdiod men däremot inte för att driva en motor. Då behövs någon form av förstärkare. För att åstadkomma en blinkning behöver man skriva först talet 1 följt av talet 0 på en utgång. Microbit är mycket snabb och lysdioden lyser i ovanstående exempel bara ca 10 mikrosekunder och för att få en användbar blinkning och hinna se något behöver man därför lägga in en paus i programmet innan man släcker lysdioden.

Pauser eller fördröjningar

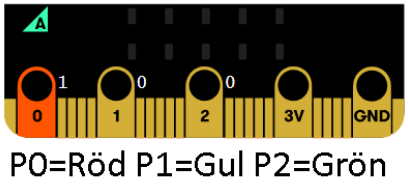
I Microbit finns en pausfunktion, programmet ”stannar ” under en tid som anges som ett tal i instruktionsblocket. Längden på pausen anges i millisekunder. Eleverna behöver få testa och pröva dessa instruktioner, styra olika utgångar och variera paustiderna. En lysdiod kan styras för att signalera ett SOS meddelande (· · · — — — · · ·) eller generera andra typer av sekventiella signaler. Eleverna kan få skapa en sekvens av flera olika utsignaler som till exempel för att styra ett enkelt trafikljus.





Tabellen nedan visar hur signalerna till tre olikfärgade lysdioder anslutna till P0=Röd, P1=Gul och P2=Grön ska se ut och hur ett programexempel för Microbit kan se ut.

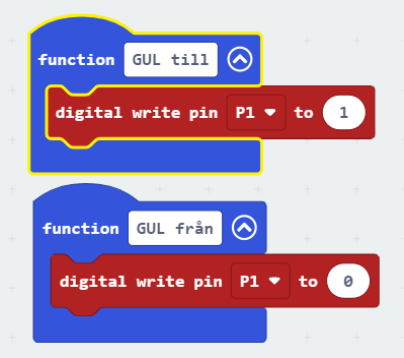
P0	P1	P2
1	1	0
0	0	1
0	1	0
1	0	0



P0=Röd P1=Gul P2=Grön

Programmet går att provköra i simulatorn där man kan se hur signalen på de tre utgångarna ändras.

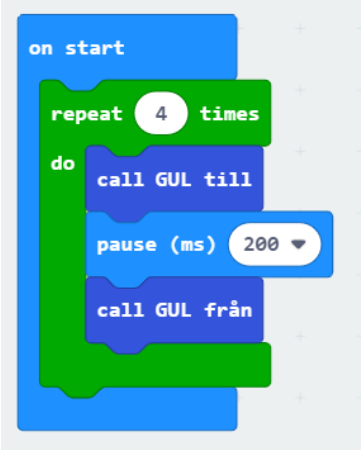
Om sekvenserna och därmed programmen blir långa kan de bli svåra att läsa, förstå och felsöka och det är viktigt att redan på ett tidigt stadium introducera eleven för ett mer strukturerat sätt att programmera.



Hjälp dem att använda funktioner (function) och att använda relevanta namn på variabler och andra beteckningar i programmet.

I vidstående program skapas först två funktioner som styr och kontrollerar värdet på utgång P0: GUL_Till och GUL_Från.

I huvudprogrammet anropas dessa funktioner fyra gånger så att en lysdiod inkopplad till P0 blinkar. Även om programmet blir lite längre när man skapar funktioner på detta sätt blir det mer lättläst.



Då blockprogrammeringen gör det enkelt att utföra repetitionssatser så kan man redan här börja använda dessa.

Det är viktigt att låta eleverna träna på dessa till synes enkla instruktioner så att de bygger upp sin egen ”verktygslåda” av lösningar till olika delproblem. Då finns det möjlighet att de börjar se mönster och kan identifiera lösningar/algoritmer för att lösa delproblem. Som lärare går vi ofta alldeles för fort fram för att vi vill att eleven ska känna att de lyckas göra avancerade program. Eleverna kopierar då eller följer ett ”recept” för hur och vad de ska göra. Det blir naturligtvis ofta en färdig fungerande produkt, ett roligt spel eller liknande, men vi lärare är mer intresserade att de lär sig att använda datorn till att lösa olika tekniska problem. Låt dem pröva och experimentera med så många små program som möjligt, korta och ej så komplicerade så de inte kan förstå dem. Låt dem ändra värdet på konstanter och ändra delar av ett program. På https://makecode.Microbit.org/projects finns många små projekt och användbara exempel på hur Microbit kan användas till att lösa olika problem.

Se dessa små exempel på algoritmer som ett digitalt alfabet, liknande det Polhemska mekaniska alfabet som tidigare beskrivits. I de följande moduldelarna om programmering ska vi konstruera digitala system för att mäta, detektera, styra och reglera. Vi kommer att ansluta sensorer, motorer och annan elektronik till Microbit-datorn.

Referenser

Aivaloglou, E., & Hermans, F. (2016). How Kids Code and How We Know: An Exploratory Study on the Scratch Repository. Paper presented at the ICER '16, Melbourne.

Bagcı, B. B., Kamasak, M., & Ince, G. (2017). The Effect of the Programming Interfaces of Robots in Teaching Computer Languages. Paper presented at the RiE 2017.

Du Boulay, B. (1986). Some Difficulties of Learning to Program. Journal of Educational Computing Research, 2(1), 75-93.

Fields, D. A., Giang, M., & Kafai, Y. (2014). Programming in the Wild: Trends in Youth Computational Participation in the Online Scratch Community. Paper presented at the WiPSCE 2014, Berlin, Germany.

Heintz, F., Mannila, L., Nygårds, K., Parnes, P., & Regnell, B. (2015). Computing at School in Sweden – Experiences from Introducing Computer Science within Existing Subjectsz. Paper presented at the ISSEP 2015, Ljubljana, Slovenia.

Lavonen, J. M., Meisalo, V. P., Lattu, M., & Sutinen, E. (2003). Concretising the programming task: a case study in a secondary school. Computers & Education, 40, 115-135.

Pásztor, A., Pap-Szigeti, R., & Lakatos Török, E. (2010). Effects of Using Model Robots in the Education of Programming. Informatics in Education, 9(1), 133-140.

Pea, R. D., Soloway, E., & Spohrer, J. C. (1987). The Buggy Path to The Development of Programming Expertise. Focus on Learning Problems in Mathematics, 9(1), 5-30.

Perkins, D., & Salomon, G. (1989). Are cognitive skills context-bound? Educational Researcher, 18(1), 16-25.

Robins, A. (2010). Learning edge momentum: A new account of outcomes in CS1. Computer Science Education, 20(1), 37-71.

Rodrigo, M., Andallaza, T., Castro, F., Armenta, M., Dy, T., & Jadud, M. (2013). An analysis of java programming behaviors, affect, perceptions, and syntax errors among low-achieving, average, and high-achieving novice programmers*. Journal Educational Computing Research, 49(3), 293-325.

Rubio, M. A., Romero-Zaliz, R., Mañoso, C., & de Madrid, A. P. (2015). Closing the gender gap in an introductory programming course. Computers & Education, 82, 409-420.

Sleeman, D., Putnam, R. T., Baxter, J., & Kuspa, L. (1986). Pascal and High School Students: A Study of Errors. Journal of Educational Computing Research, 2(1), 5-23.

Sorva, J. (2013). Notional Machines and Introductory Programming Education. ACM Transactions on Computing Education, 13(2).

Sparkes, R. A. (1995). An investigation of year 7 pupils learning Control Logo. Journal of Computer Assisted Learning, 11, 182-191.

Wing, J. M. (2006). Computational Thinking. Communication of the ACM, 49(3), 33-35.