Vad är programmering?

Programmering förknippas ofta med kod, teknik, dyra datorer och specialistkunskap En vanlig uppfattning är att programmerare ofta utgörs av vita män med glasögon (Google, 2015) som sitter isolerade och knackar kod. Och kod, vad är det? Sida upp och sida ner med obegripliga tecken som resulterar i system som man helst vill slippa veta något om. Om man har denna bild av vad programmering handlar om, är det förståeligt om man ställer sig frågan vad begreppet "programmering" gör i den reviderade läroplanen för grundskolan.

Programmering handlar dock om mycket mer än bara kod. I kommentarmaterialet "Få syn på digitaliseringen på grundskolenivå" skriver Skolverket (2017) att programmering inkluderar

"*att skriva kod, vilket har stora likheter med generell problemlösning. Det handlar bland annat om problemformulering, att välja lösning, att pröva och ompröva samt att dokumentera. Men programmering ska ses i ett vidare perspektiv som även omfattar kreativt skapande, styrning och reglering, simulering samt demokratiska dimensioner. Det här vidare perspektivet på programmering är en viktig utgångspunkt i undervisningen och programmering ingår därmed i alla aspekter av digital kompetens.*" (s. 10)

Programmering som en process

Programmering kan alltså ses som något mer än enbart kod eller kodning. Programmering är en process där man utgår från ett problem, en idé eller ett uppdrag som man behöver lösa. Som vilken process som helst går det att identifiera olika steg, till exempel följande:

1) Vad är det vi jobbar med? Analysera problemet, idéen eller uppdraget.
2) Hur kan vi tackla detta? Granska olika lösningsmodeller.
3) Hur väljer vi att göra? Designa en lösning.
4) Hur kan vi implementera denna lösning? Skriv koden.
5) Fungerar det som vi tänkt? Felsök och korrigera programmet.

Processen är inte rätlinjig utan ofta måste vi gå tillbaka till tidigare steg då vi märker att vi bortsett från en detalj eller tänkt fel i något steg. Det viktiga här är dock att se att kodningen enbart utgör *ett* steg i denna process. Programmering handlar om problemlösning och logiskt tänkande. Om att kunna abstrahera och generalisera, samtidigt som man är detaljerad och strukturerad i sitt sätt att arbeta. Det handlar om att vara kreativ, tänka nytt och inte vara rädd för att göra fel – det hör till då man programmerar.

Målsättningen med programmering är att få datorn att göra de saker den är bra på, t.ex. snabba beräkningar och genomgångar av stora mängder data, så att vi människor får göra det vi är bra på, det vill säga komma med idéer, designa bra lösningar och vara kreativa. Detta kommer bli allt viktigare framöver då de problem och situationer vi möter blir allt mer komplexa.

Program, algoritmer och kod?

Något som kan vara förvirrande som nybörjare i programmering, är de många begrepp man möter.. Särskilt förvirrande är det att de ibland förekommer även utanför programmeringsvärlden, men i andra betydelser. Begreppet **program** till exempel. Vi är vana att vi kan titta på program på TV, det finns program på en konferens och vi har olika program till datorn. Inom programmering kallas program även programvara, mjukvara eller datorprogram. Det är en samling instruktioner som en dator kan tolka och

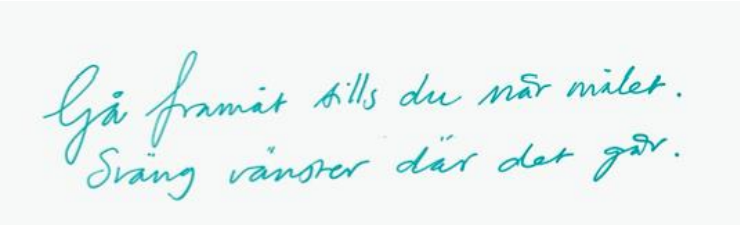
utföra. Program kan vara mycket enkla med några få rader kod eller mycket komplexa med miljontals rader kod.

I grundskolan möter eleverna begreppet **algoritm** oftast som synonymt med hur man gör en uppställning i matematik. En algoritm är en lista väldefinierade instruktioner för att lösa en specifik uppgift med ett begränsat antal steg. Det kan alltså vara en förklaring till hur man vill lösa just en matematikuppgift men också hur ett datorprogram ska sortera information eller bestämma vad som ska visas först på en sida i sociala medier.

Kod är ungefär samma sak som program, men används inte alltid helt synonymt. Till exempel skulle man kunna säga "Programmet fungerar inte som det ska, det måste vara något fel i koden." Program används mer övergripande, medan kod ofta avser den bakomliggande programmeringen. Något som också kan förvirra i början är att ordet kod inte används i plural. Det heter "mycket kod", inte "många koder". Vi talar om fem rader kod, inte fem koder. Program, algoritmer och kod är begrepp som alla kan beskrivas som en lista med instruktioner för hur något ska utföras. Ett stort program är ofta uppdelat i mindre delar som utför olika saker. Var och en av dessa delar motsvarar då en egen algoritm som gör att datorn utför det vi vill att den ska göra. Känns det förvirrande? Det ÄR lite snårigt men du kommer att möta dessa begrepp flera gånger i den här kursen så förhoppningsvis klarnar det så småningom.

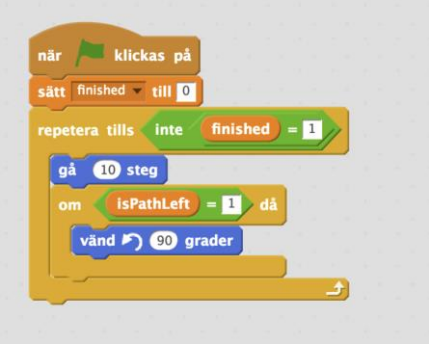
Olika språknivåer

Det språk som vi använder till vardags kallas i programmeringssammanhang för naturligt språk, till skillnad från ett programspråk som är konstruerat för att datorn ska kunna tolka det. De flesta programspråken består av text, men idag finns även blockprogrammering, som är grafiska element som representerar skriven kod. Programspråket blir i sin tur översatt av kompilatorn, till maskinkod som består av ettor och nollor.



Naturligt språk

Symboler



Blockprogrammering med grafiska element

```
while (notFinished()) {
  moveForward();
  if (isPathLeft()) {
    turnLeft();
  }
}
```
Programspråk (JavaScript)

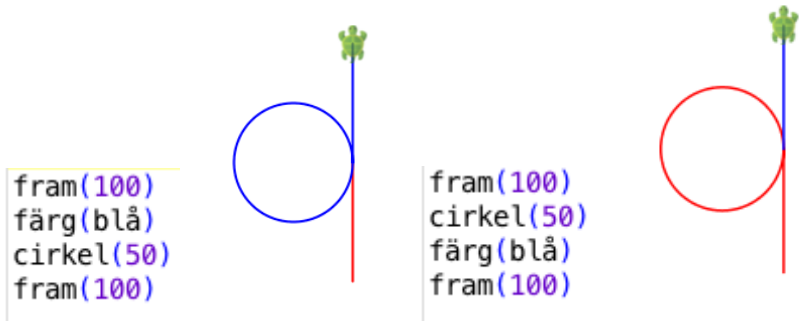
```
00110101001101010001011011001001010110010101010100101010
10111100010101111000110111011100010101001010010110101010
00101010001001001011010100010100101110001100101010010011
00011010101011110101101111010010010001011010101010000010
00110101001101010001011011001001010110010101010100101010
```
Maskinkod

Som vi konstaterade tidigare handlar programmering, förenklat, om att ge instruktioner till datorn så att den utför det vi vill att den ska göra. En viktig skillnad mellan människor och datorer är att datorer inte kan läsa mellan raderna eller kasta om ordningen på instruktionerna. En kommentar som "vad kvavt det är här inne" kan en människa uppfatta som en uppmaning om att öppna ett fönster. Vi kan ta instruktioner i fel ordning och förstå utelämnad information som i uttrycket "Var god stäng toalettlocket". Förhoppningsvis förstår läsaren att locket ska stängas efter toalettbesöket och inte före. Vi är alltså inte känsliga för om instruktioner är strikt sekventiella, eller inte. Vi kan också klara oss utan att veta exakt hur många repetitioner som ska göras innan tänderna är färdigborstade och om vi läser att vi ska vispa grädden, förstår vi att den ska vispas tills grädden bli tjock, utan att det uttryckligen står i receptet. En dator skulle behöva varje moment uttryckligen förklarat och definierat.

Grundläggande byggstenar

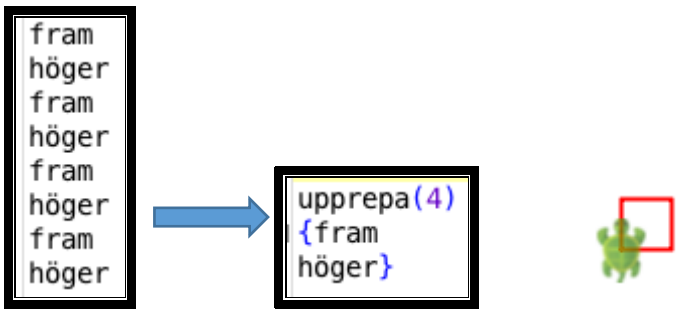
Det finns en mängd programspråk (Javascript, Ruby, Python, Java, C++, Scratch, ...) som alla har sina egenskaper som gör att de passar bättre i vissa sammanhang och sämre i andra. Vissa språk är anpassade för att användas på webben, medan andra lämpar sig mer till att bygga komplexa databaser. Vi skiljer ofta mellan textbaserade programspråk och blockbaserade programspråk. De blockbaserade programspråken är framför allt framtagna för att underlätta inlärningen för nybörjare inom programmering. De vanliga programspråken innehåller dock alla likartade byggstenar. Tre av dessa byggstenar är instruktionstyperna **sekvens, repetition** och **villkor**, som gör det möjligt att bygga ett flexibelt program.

Sekvens handlar om ordning. Sekventiella instruktioner utförs rakt av, en gång i den ordning de står. Om ordningen ändras, blir resultatet annorlunda. I programmen nedan har instruktionen för färg, ändrat plats och därför ritas figuren upp på olika sätt.



Med **repetition**, som även kallas loop eller slinga, kan vi få datorn att automatiskt upprepa en eller flera instruktioner, vilket gör programkoden enklare att både läsa och underhålla. Här gäller det att identifiera mönster som ska upprepas flera gånger. I denna text har vi valt att använda orden repetition och slinga.

Vad innebär det till exempel att göra en kvadrat? Jo, vi behöver rita fyra lika långa linjer med 90 graders vinkel emellan. Ett sätt att göra det är att rita en linje, svänga 90 grader, rita en linje till, svänga 90 grader igen, och så vidare tills vi har vår kvadrat. Men det vi gör är ju då att upprepa samma sak flera gånger om – vårt mönster är "rita linjen, sväng 90 grader". Då kan vi istället använda en slinga som upprepar detta mönster 4 gånger. Och eftersom datorn är effektiv och snabb kan vi lika väl be den upprepa mönstret 1000 eller 1 miljon gånger.



Programmen ger samma resultat, men det kräver mindre kod med en repetition inlagd. Klamrarna (kallas även måsvingar eller krullparenteser) anger att båda instruktionerna (fram och höger) ska upprepas fyra gånger.

Om vi inte hade tillgång till slingor skulle vi behöva skriva upp alla instruktionerna efter varandra. Att upprepa två instruktioner 1000 gånger skulle innebära 2000 rader kod. Och för att få de två instruktionerna att köras 1 miljon gånger skulle vi behöva skriva 2 miljoner rader kod. Det blir kortare och mer effektiv kod om vi använder oss av slingor istället.
Det finns flera olika typer av slingor: sådana som upprepar en eller flera instruktioner ett *givet antal gånger* (t.ex. repetera 10 gånger), *tills ett givet villkorsuttryck uppfylls* (t.ex. repetera tills poängantalet överstiger 100) eller *så länge som ett villkorsuttryck är uppfyllt* (t.ex. repetera så länge som poängantalet är mindre än 100). Du ser exempel på de två första typerna i figuren nedan.

Den tredje instruktionstypen är **villkor** (alternativ, if-sats), och med den kan vi få programmet att bete sig olika i olika situationer, beroende på om ett villkorsuttryck är sant eller inte. Vad är då ett **villkorsuttryck**? Jo, det är ett påstående som antingen är sant eller falskt, t.ex. "Det regnar", "Gretas ålder < 20" och "Jag heter Bo". Antingen regnar det eller så regnar det inte. Greta är antingen yngre än 20 år eller så är hon det inte. Och personen som säger "Jag heter Bo" talar antingen sanning eller så är hans påstående falskt.

Om denna möjlighet inte fanns skulle ett program alltid bete sig på samma sätt oberoende av till exempel hur många poäng vi har i ett spel, om ett trafikljus visar grönt eller rött eller hur hög temperatur en sensor inne i ett kärnkraftverk registrerat. Med hjälp av villkor kan vi skriva ut olika meddelanden beroende på hur många poäng spelaren har (se exempelprogrammen i figuren nedan), få ett självkörande fordon att köra vidare om trafikljuset är grönt och att stanna om det visar rött och få kontrollsystemet för ett kärnkraftverk att skicka ut ett alarm om sensorn registrerar en temperatur som är högre än tillåtet. För att program ska vara användbara behöver de kunna reagera på saker som sker både i programmet självt och i dess omgivning. Figuren nedan visar hur du kan använda olika villkor i ett spel för att programmet ska bete sig olika beroende på hur många poäng spelaren har fått.