Varför programmering?

Världen vi lever i är full med prylar som är programmerade. De kallas för "intelligenta". Man pratar om *artificiell intelligens*. Men prylarna kan inte tänka själva. Någon har programmerat dem, närmare bestämt de elektroniska komponenter – små datorer – i dem. Det är de som styr all funktionalitet.

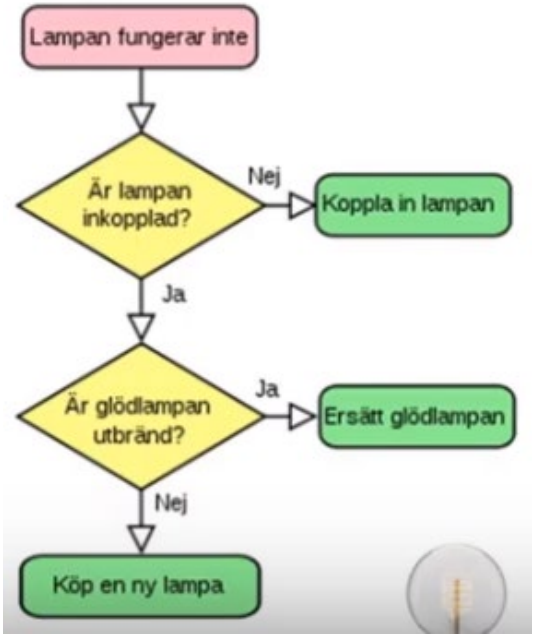
Programmering är ett av de mest spännande kapitlen i teknologihistorien. Inte bara därför att den har lagt grunden till den moderna IT-industrin och på gott och ont revolutionerat världen. Den har också bidragit till att förverkliga den urgamla mänskliga drömmen att förenkla mödosamma arbeten. Istället för att plåga sig instruerar man en maskin med idéer. Programmering realiserar önskemålet att låta datorn göra jobbet för att ha mer tid över för annat i livet.

När man tröttnat på att använda program som andra skrivit – maila, surfa eller lyssna på musik – är det dags att börja programmera själv. Det är roligare att köra en bil än att bara åka med. Det är kreativiteten och det fria skapandet som lockar. Med programmering kan du testa helt nya egna idéer.

"Everyone in this country should learn how to program a computer. Because it teaches you how to *think*."

(Alla borde lära sig programmera för det lär oss att *tänka*.) *Steve Jobs*

Men hur programmerar man?



Egentligen gör vi det varje dag utan att vara medvetna om det. Är en lampa trasig följer vi ungefär det som t.ex. kan beskrivas med bilden till höger, en s.k. *flödesplan*. I praktiken löser vi problemet att ersätta en trasig lampa genom att tänka och göra så utan att någonsin rita en flödesplan. Flödesplanen illustrerar *algoritmen* (tillvägagångssättet) för problemets lösning. När den en gång är ritad skulle den kunna användas av vem som helst som vill byta en trasig lampa. Den blir en slags manual.

Ett annat vardagligt exempel är matlagning. Vare sig vi använder ett recept ur en kokbok eller lagar efter känsla, följer vi en algoritm som dessutom – till skillnad från lampalgoritmen – även har en *input*, råvaror och en *output*, maträtten. Hårdvaran som hjälper oss är köket med alla sina instrument. Matreceptet är mjukvaran dvs programmet. Det är precis samma struktur när vi kör ett program på datorn, matar in indata och får ut utdata som resultat. Programmet vi använder är avgörande för resultatet, precis som matreceptet samt dess förverkligande är avgörande för om vi lyckas med maträtten.

Algoritmiskt tänkande

Båda exemplen visar: Det är algoritmer som medvetet eller omedvetet styr *hur* vi gör – ett sätt att tänka vars gemensamma drag kan generaliseras så här:

1. Att formulera problemet och definiera målet. Hur når vi målet – problemets lösning?

2. Genom att bryta ner problemet i mindre, överskådliga och enklare delar, s.k. *moduler*. Varje modul ska i princip kunna utföras av vem som helst. Detta kallas för *modularisering*.

3. Genom att ge *instruktioner* som leder till problemets lösning. De måste formuleras på ett entydigt sätt så att de inte kan tolkas på olika sätt. För datorer gör exakt som vi säger. Det har visat sig att det vanliga språket inte lämpar sig för detta ändamål, för det är tolkbart. Skönlitteraturen är ett praktexempel för olika tolkningar av språket. Det vore synd om det inte vore så. Därför har man i programmering hittat på andra, speciella programmeringsspråk vars vokabulär och syntax följer strikta regler som är entydiga. Datorn kan tolka dessa regler endast mekaniskt.

4. I denna process uppstår situationer där vi måste träffa ett *val* – samma sak som att besvara en *fråga*. Den första frågan i algoritmen "Att byta lampa" är "Är lampan inkopplad?" (ovan). Valet mellan "Ja" och "Nej" avgör hur algoritmen fortsätter. Ytterligare val följer.

Det är avgörande att skilja mellan *instruktion* och *val*. En instruktion måste *utföras* medan ett val är en fråga som måste *besvaras*. Därför är de markerade i flödesplanen med olika färger.

Algoritmers andra ingredienser

Instruktion och *val* är endast två av algoritmers viktigaste strukturelement medan *modularisering* är en allmän princip i all problemlösning och programmering. Det finns ytterligare strukturelement i konstruktion av algoritmer som *sekvens* och *repetition* som kommer att tas upp i senare artiklar.

Avgörande i en algoritm är strukturelementens inbördes *ordning*. Tar man in i en kokande gryta potatisen först och köttet sedan – istället för tvärtom – blir det mos istället för maträtt. Till detta hör även algoritmens korrekta avslutning. Utan ett exakt formulerat *avslutningskriterium* som uppnås i ändlig tid uppstår evighetsloopar. När datorn "hänger sig" är detta ofta orsaken.

Ytterligare en ingrediens av algoritmer är *logik*. Datorer kan ingen logik. Människan måste föra över logiken in i datorn. Det är det som kallas för *artificiell intelligens*. Speciellt formuleringen av val i algoritmer kräver logiskt tänkande.

Att upptäcka *mönster* är också en förmåga som ofta behövs i konstruktion av algoritmer, vilket vi kommer att se i kommande artiklar.

I valet av instruktioner som ska tas med i en algoritm är det en självklarhet att man sorterar bort allt som är mindre relevant och tar in endast det som är relevant. Även att avgöra *relevansen* av saker och ting för att uppnå det definierade målet (punkt 1) hör till programmerarens uppgifter.