

# **Introduktion til VICA – naturlige sprog vs. programmeringsprog.**

rev. 2 v. Anders Bach Nielsen (Python version)

Steffen Brandorff

lektor, ph.d.  
Institut for Informations- og Medievidenskab  
Aarhus Universitet  
[sbrand@imv.au.dk](mailto:sbrand@imv.au.dk)

## 2 Steffen Brandorff

Dette skrift er beregnet som introduktion til kurset "Værktøj i Computer Arbejde", til daglig kaldet VICA. Kurset er et 1. semesters kursus i grunduddannelsen i Informationsvidenskab på Aarhus Universitet, og sigter mod at give en første erfaring med programmering.

Programmering i høj grad en færdighed, på linie med mange håndværk. Programmøren skal kende sine programmeringsomgivelser, skal lære forskellige fejlsituationer, lære at bruge on-line hjælp konstruktivt osv. Der er ingen erstatning for praktisk erfaring, og det er derfor vigtigt i et programmeringskursus at komme i gang med praktisk arbejde så hurtigt som muligt.

Lærebogen på kurset er p.t. Glenn Brookshear, "Computer Science – an overview", 7<sup>th</sup> ed., Addison-Wesley 2002, og den er helt traditionelt bygget op. Først efter kap. 4 er tilvejebragt et grundlag for praktisk programmering. Det er alt for sent for vores kursus, så derfor har jeg forfattet denne lille skrivelse som et introducerende kapitel. Lærebogen har endvidere et meget forenklet syn på sprog, og det har jeg haft et ønske om at udvide.

© Steffen Brandorff, IMV 2003  
skrevet i Microsoft Word under MacOS X  
skabelon lncs.dot fra Springer Verlag  
brødtekst i Times 10 pt  
ca. 40.000 tegn excl. mellemrum

IMV, Aarhus Universitet d. 31.07.2003  
rev. 24.8.2004

Steffen Brandorff  
Anders Bach Nielsen

## Sprog og maskiner

Den følgende tekst prøver at indkredse, hvad der menes, når vi taler om, at computeren har et sprog. Jeg vil i den forbindelse benytte begrebet ”naturlige sprog<sup>1</sup>”, selvom sådanne nok ikke kan ses som naturlige i fx. biologisk forstand. Menneskenes mange sprog er et resultat af deres historie og kultur. Begrebet ”naturlig” kan imidlertid godt bruges til at beskrive forskellen mellem sprog, som mennesker benytter sig af i kommunikation med hinanden, og programmeringssprog, som vi bruger for at få maskiner som computere til at udføre opgaver.

De naturlige sprog har nogle egenskaber, som vi kan se lidt på. Vi har et modersmål, som imidlertid er blevet usynligt for os i langt de fleste situationer. I skoletiden er vi opmærksomme på, at vi skal lære nye ord og begreber, men i de tilfælde er fokus på at forstå og indoptage nye ord. Men sproget som sådan er ikke noget vi lægger mærke til – det er et værktøj som vi har fået med i vores dagligdag, og som vi blot bruger på samme måde som vi bruger hænder og fødder. Men sproget er et særligt værktøj – det kan bl.a. kommunikere om mange ting i vores liv – og så har det en selvreferentiel egenskab: vi kan bruge sproget til at tale om sproget og om, hvad sproget betyder for os. Kan vi på samme måde bruge hænderne til at behandle, hvad vores hænder betyder i vores liv? Det er måske selve begrebet *betydning* som her er på spil. Med det begreb er vi for alvor hoppet over i sprogets gebet.

Med en inspiration fra filosofen Heidegger kan vi tale om, at sproget ligesom andre værktøjer kan ligge parat og blive brugt – det er *ready-at-hand*. Det gælder fx også en hammer. Vi lægger ikke mærke til hammeren – vi slår søm i. Hvis hammeren brækker eller hovedet falder af har vi et sammenbrud af værktøjet : et *breakdown*. Nu får vi pludselig øje på hammeren; den bliver *present-at-hand*<sup>2</sup>. Det er den også, når vi første gang skal lære at bruge den.

Pointen er, at vi er nødt til at beskæftige os med selve sproget, når vi programmerer, og det skyldes, at vores kommunikation med maskinen ellers vil bryde sammen. Vi ved, at vi ikke kan sige noget som helst fornuftigt til den; men vi kan lære den – ved tålmodigt arbejde – at udføre handlinger, altså at rette sig efter os. Gevinsten er selvfølgelig, at den kan udføre arbejde i et uhyre tempo, og kan gentage kedsommelige ting masser af gange uden at blive træt. Endnu bedre: vi kan bruge den som medium og tale til andre mennesker igennem den på interessante måder.

---

<sup>1</sup> Denne betegnelse er almindeligt brugt, når dataloger begiver sig ind på diskussion af sprog til maskiner og mennesker.

<sup>2</sup> Heidegger har nok sagt disse ting på tysk, men jeg kender kun de nævnte engelsksprogede versioner.

### Nogle første tanker om sprog og symboler

Har dyr mon sprog? De kan da kommunikere med hinanden, har vi set i naturfilm efter naturfilm! Vi kan måske komme et skridt videre ved at skelne mellem et *signal* – og det kender vi fra dyreverdenen ”Hov, der er fare på færde, her og nu!” og et *symbol*. Måske kan vi osse sige, at den lille fugls sang: ”Jeg er en stor og meget imponerende musvit-han!” er et signal. Signalet er knyttet til en umiddelbar situation – et ”nu” – og når en baby græder, er der her og nu er noget i vejen, som skal rettes. Det er forbeholdt voksne at græde over ting, der skete i går og som nu ikke længere foregår.

Et symbol er groft sagt uafhængigt af tid og sted. Det har en *betydning*, der tillader os at overføre information mellem tider og steder. Vi kan fortælle om, hvad vi oplevede sidste år. Vi kan bruge ordet ”hest” uden at skulle pege på et konkret dyr.

Iflg. Ferdinand de Saussure<sup>3</sup> har et symbol to bestanddele:

- et fysisk udtryk (det stavede eller udtalte – ”lyden selv”)
- et begrebsindhold – en betydning

Der er selvfølgelig forbehold her, fordi det stavede eller udtalte symbol lyder forskelligt og ser forskelligt ud på diverse sprog og dialekter. Der er heller ikke sikkerhed for, at begrebsindholdet er stabilt fra sprog til sprog. Dette forhold bliver behandlet lidt senere i dette skrift.

Russisk-amerikaneren Roman Jakobson<sup>4</sup> har opstillet to punkter vedrørende sprogs udtryksmuligheder:

- der er visse ting, som et sprog *skal* udtrykke
- mange ting, som et sprog *kan* udtrykke

De naturlige sprog skal kunne udtrykke de forhold, genstande og handlinger, som forekommer i alle menneskers liv (se senere). Derudover er der frit spil – sproget kan indeholde lige så mange udtryk, som dets brugere har behov for.

Når vi taler om programmeringssprog, er disse principper stadig gældende. Det kan endda lade sig gøre at specificere den første regel. Programmeringssprog kan reduceres meget mere end naturligt sprog. De skal – udover primitive udsagn – faktisk kun kunne udtrykke tre forskellige strukturer: *sekvens*, *forgrening* og *gentagelse*. Det kan skyldes, at programmeringssprog er beregnet på at udtrykke handling, ikke på at føre dialog med mennesker. Ud over de enkle handlinger kan programmeringssprog indeholde mange andre nyttige konstruktioner, som kan hjælpe programmøren til at lave bedre programmer, mere fejlfrie rutiner eller mere nyttige interfaces. Men kopt ned til det helt basale behøver vi kun de tre nævnte strukturer: sekvens, forgrening og gentagelse.

---

<sup>3</sup> fransksproget sprogforsker fra Svejts, 1857-1913

<sup>4</sup> arbejdede også en del år i Brno, Tjekkoslaviet som del af tjekkisk sprogforskning

## Sproglige beskrivelsesniveauer

*Et sprog* kan beskrives som *et filter, vi lægger ned over verden*. I kraft af vores ordforråd har vi så at sige valgt, hvilke ting vi overhovedet kan snakke om. Wittgenstein har i sine unge dage udtrykt det sådan, at ”sproget er vor tankes grænse”. Vi kan – efter denne forståelse – kun tænke så langt som vores sprog tillader os. Denne skarpe pointe modificerede han senere, men den har stadig noget vigtigt at sige: vi er i høj grad afhængige af at verbalisere, når vi tænker sammen. Hvis vi ud over vores eksisterende ordforråd har brug for nye begreber, må vi fremstille dem ved at sammenstille allerede kendte begreber (fx i en definition) og derved danne nye begreber. Vi kan bruge forskellige midler til at fremstille nye begreber: vi kan *generalisere*, vi kan *aggregere* osv. Sådanne nye ord skabes fx inden for videnskaberne; de møder nye fænomener, som skal navngives; de ser på verden med nye øjne og må finde ord for, hvad de ser.

Hvis vi i denne forbindelse ikke tænker mere på dannelse af nye begreber, men blot beskæftiger os med de eksisterende, kan vi udskille forskellige niveauer i sprogbeskrivelsen fra hinanden. Man kan tale om, at sproget har forskellige lag med forskellig organisering, og dem vil vi tillade os at behandle hver for sig.

Beskrivelsen *går ud fra*, at

- et sprog består af *sætninger*;
- sætninger består af *symboler*,
- som sammensættes efter *syntaksregler*,
- der fastsættes i en *grammatik*.

Hertil kommer så indholdet af sætningerne – deres betydning. Altså – kort skitseret:

Sprogbeskrivelsesniveauer:

leksikalsk	leksikon	ordforråd
syntaktisk	syntaks	regler for korrekt rækkefølge mv.
semantisk	semantik	betydningsindhold af ord, sætninger

**Fig. 1.** Tre hovedniveauer i sprogbeskrivelsen

Det er almindeligt inden for beskrivelse af programmeringssprog, at en oversigt over et sprogs syntaks kaldes en *grammatik*.

## Leksikalske forhold

I dette afsnit har vi gang i det niveau, som også beskrives som ”ordforrådet”. Der er fællestræk i menneskers sprog, jvf. Roman Jakobson. Disse kan dreje sig om ting, som alle mennesker har tilfælles uanset nationalitet og kultur.

For eksempel må det være fælles alle steder, at:

- mennesker fødes, bliver unge, senere gamle, og dør:
- mennesker spiser, sover,
- mennesker går oprejst og falder sommetider.
- mennesker udfører handlinger på objekter ( $\approx$  subjekt, verbum, objekt)

Sådanne træk *skal* alle sprog kunne udtrykke. Hvad ord og udtryk betyder, kan beskrives som det semantiske niveau. Mere om det om lidt; først noget om syntaks.

### Syntaktiske forskelle mellem naturlige sprog

Hvis vi kigger ud over verdens ca. 4500 sprog (måske flere) kan vi udlede tre vigtige elementer, som skal være repræsenteret, men hvor rækkefølgen indbyrdes er forskellig.

Overordnet kan man skelne mellem forskellige syntaktiske sprogtyper, alt efter, hvordan placeringen af de tre hovedled i sætningen foretages. De tre er *subjektet* eller *grundledet* (S), *objektet* eller *genstandsleddet* (O), og *verbet* eller *udsagnsleddet* (V). Der findes forskellige typer af syntakskonventioner<sup>5</sup>:

#### SVO-sprog

fx. dansk      Jens har bulet en bil

#### SOV-sprog

fx tyrkisk      Hasan öküzü aldı      Hasan en okse har købt      Hasan har købt en okse

#### VSO-sprog

fx. arabisk      **daraba** 'isa musa      slog Isa Musa      Isa slog Musa

**Fig. 2.** Tre typer syntakskonventioner for naturlige sprog

Detaljer om disse sprog kan vi ikke komme ind på i denne forbindelse, men figuren har da antydnet, at rækkefølgen af de sætningsdannende komponenter altså kan være forskellig fra sprog til sprog. Og dette beskrives altså af syntaksen fx i en "grammatik". Der er også i et eller andet omfang i programmeringssprog tale om, at vi instruerer computeren i at udføre handlinger på "ting". Så begreberne bag "verbum" og "objekt" er i hvert fald til stede i et eller andet omfang. Vi skal senere se på, hvordan programmeringssprog har udtrykt sådanne forhold.

### Kommandoer

Vi taler til maskiner - ikke til hinanden - i form af kommandoer. Børn og soldater er måske vant til denne sproglige form: *Ferdinand, gå i seng!* -

<sup>5</sup> jvf. den store danske encyklopædi, opslag "sprog" og "sprogtypologi"

*Pedersen, flyt den kanon!* osv. Her er rækkefølgen selvfølgelig præget af, at vi henvender os til en person, men denne vil ofte være subjekt for handlingen. Så også i kommandosætninger kan vi se SVO-sammenstillinger.

### **Semantiske forhold – ord er ikke alt<sup>6</sup>**

Med det semantiske niveau nærmer vi os så det egentlige indhold i kommunikationen. Vi så i forbindelse med det leksikalske niveau, at der er basale forhold, som et sprog skal kunne udtrykke. Her er betydningsindholdet helt klart. "At sove" betyder det samme, hvad enten du sover i et dansk soveværelse, i en flyvemaskine eller på jorden i Kalahari ørkenen.

Modsat kan vi sige, at der er forhold, som vi ikke har ord for. Vi kan prøve at kigge på tidspunkter, hvor sproget bryder sammen og ikke kan udtrykke bestemte ting.

Journalisten række mikrofonen frem mod Oscarvinderen, etapevinderen i Tour de France, manden der har vundet 10 mio. i Lotto eller kvinden der overlevede skibskatastrofen. Og så spørger han: "*Hvad følte du dengang .....*" Eller en anden situation:

Vi har haft en stor musikalsk oplevelse, og vi vil gerne delagtiggøre nogen i, hvad vi har oplevet. Resultatet bliver ofte bare: "*Det var godt nok fedt/sejt/dejligt*"

Der kommer stort set aldrig noget nuanceret svar, som kan formidle de følelser folk har haft i den slags situationer. Digttere kan måske, men almindelige mennesker kan som regel ikke. Nogle dele af menneskelivet foregår på ikke-verbale planer og det gør, at vi ikke kan få direkte hold på dem, i hvert fald kan vi ikke fastholde dem i ord.

Så ligesom duften af kanel ikke kan formidles i billeder, og en brugsanvisning til en kaffemaskine vanskeligt kan konstrueres med musik, så er der ting, der ikke kan udtrykkes i det talte eller skrevne sprog. Vi har nu en meget lang litterær tradition, som også har forsøgt sig med at beskrive rædsel, triumf, medfølelse osv. Og det er jo gået fint – vi kan opleve meget. Men en medrivende beskrivelse af en musikalsk oplevelse giver os ikke selve musikken, blot den deraf afledte fornemmelses sproglige pendant.

### **Findes der en fælles semantik / betydning?**

Programmeringssprog kan nu heller ikke udtrykke menneskelige følelser, så det er ikke på det punkt de centrale forskelle mellem menneskesprog og computersprog ligger. Programmeringssprog er som regel meget magre og fattige på ord og begreber. De handler som sagt om handlinger. Til gengæld er semantikken meget præcist beskrevet idet kommunikationspartneren er noget så fattesvagt som en maskine.

Der er nu også helt dagligdags forhold, som udtrykkes på nogle sprog, men ikke på andre. Den danske semiotiker Louis Hjelmslev har givet udgangs-

---

<sup>6</sup> "Wovon man nicht sprechen kann, darüber muss man schweigen" – Wittgenstein, Tractatus.

punkt for følgende eksempel, hvor jeg sammenligner tre sprog ud fra, hvilke ord de bruger om træ i forskellige roller:

begreb	enkelt individ	lille gruppe	stor gruppe	materiale
Dansk	<i>træ</i>	<i>lund</i>	<i>skov</i>	<i>træ</i>
Engelsk	<i>tree</i>	<i>wood</i>	<i>forest</i>	<i>wood</i>
Fransk	<i>arbre</i>	<i>bois</i>	<i>forêt</i>	<i>bois</i>

Fig. 3. Overlappende betegnelser på tre sprog (efter Hjelmslev)

Sprogene er altså ikke enige om, hvornår vi skal "genbruge" et ord. På dansk er materialet "træ" samme ord som individet "et træ", mens engelsk og fransk bruger et og samme ord for både en lille gruppe træer og selve materialet. Hvis vi havde lyst til at lave et lille, finurligt dansk digt, der spillede på de to betydninger af "træ", vil vi ikke kunne oversætte det direkte til engelsk eller fransk.

Næste opstilling sammenligner nogle begreber for familierelationer på engelsk og dansk:

Dansk	<i>farbror</i>	<i>morbros</i>	<i>onkel</i>	<i>faster</i>	<i>moster</i>	<i>tante</i>
Engelsk	<i>uncle</i>	<i>uncle</i>	<i>uncle</i>	<i>aunt</i>	<i>aunt</i>	<i>aunt</i>
Fransk	<i>oncle</i>	<i>oncle</i>	<i>oncle</i>	<i>tante</i>	<i>tante</i>	<i>tante</i>

Fig. 4. Familierelationer og deres navngivning på tre sprog

Vi kunne forsætte med "fætter – kusine", som i hvert fald engelsk sprog har vanskeligt ved at præcisere i ét ord. Det er klart, at selv om et givet ord mangler i et sprog, kan man udtrykke en relation, fx. familierelation ved et lidt længere udtryk: "aunt Mary – my mother's sister".

Til forklaring af ords betydning bruges begreberne konnotation og denotation. En *denotation* er et ords grundbetydning; den vi kan se i et leksikon eller en ordbog. En *konnotation* er et ords bibetydning, en nuance med samme denotation, og den kan være positivt eller negativt ladet. Som eksempel kan nævnes: *kvinde*, *dame*, *kone*, *madamme*, *pige*, *tøs*, *sild* og *hejre*. De har hver sin nuance og bruges i lidt forskellige sammenhænge – de har med andre ord forskellige konnotationer, mens grundbegrebet - denotationen - er "kvinde".<sup>7</sup>

Som et sidste eksempel har jeg valgt begrebet "videnskab". "Science" siger de på engelsk; men dækker begrebet det samme? Online-ordbogen fra Merriam-Webster skriver en liste på 22 ord, som har noget med "science" at gøre. De første 10 er: *science*, *behavioral science*, *Christian Science*, *cognitive*

<sup>7</sup> Sådanne konnotationer ses ofte tydeligst af skønlitteraturen. Det ses fx. at far og farbror ofte står for noget nært og velkendt, ligesom mor og moster. Disse person-par er ikke tvillinger, men dog meget tæt forbundne. "Kønskrydset" i faster og morbror afbildes ofte som mere tvivlsomme relationer. En faster ses ofte fremstillet som en skrap person (ligesom en tante). På dette punkt kan indbygget sproglig værdiladning vise sig, og hvis litteraturen har en overvægt af sådanne stereotyper, ligger en sådan ladning altså i vores sprog. Personligt kan vi kende en kærlig tante, og det vil naturligt nok farve vores individuelle konnotation til begrebet.

*science, creation science, domestic science, earth science, exact science, information science, og library science.*<sup>8</sup>

Det er en alfabetisk liste, og der grupperer sig i 3 typer: naturvidenskaber, nye videnskaber (psykologi, info, bibliotek, husholdning) samt to kristne grupper (Christian Science og kreationister). Der er hverken sprogvidenskab, historie, medier eller lignende humanistiske videnskabsfag. Encyclopedia Britannica indeholder følgende definition:

**Science:** any system of knowledge that is concerned with the physical world and its phenomena and that entails unbiased observations and systematic experimentation.

[Britannica.com , opslag "science"]

Der er discipliner inden for sprogvidenskab, der prøver at gå systematisk eksperimenterende til værks, men de er ikke med i denne definition. På dansk bruger vi begrebet *humanvidenskaber* eller *humaniora*, og hermed omfatter vi sprogvidenskaberne, arkæologi, filosofi, drama, musik, litteratur, kunst, semiotik mv. Engelsksprogede universiteter bruger ofte begrebet:

**Humanities, plural :** the branches of learning (as philosophy, arts, or languages) that investigate human constructs and concerns as opposed to natural processes (as in physics or chemistry) and social relations (as in anthropology or economics)

Men det beskrives altså ikke som en *science*. En humanvidenskab er ofte et *study* eller en *branch of learning*, men ordet science anvendes ikke i definitionen. Så status er, at humaniora er videnskab på dansk, men de er ikke *science* på engelsk. I tysk sprog dækker begrebet *Wissenschaft* temmelig præcist det samme vi kender i Danmark, men vi har historisk også vores videnskabsbegreb fra tysk. I disse år er vi under stærk indflydelse fra engelsksprogede områder, så vi bliver ofte konfronteret med forskelle mellem den angelsaksiske og den germanske universitetstradition. Det kan endda ske, at selv en dansk forskningsminister defineret "videnskab" som lig med "naturvidenskab" og helt glemmer humaniora. Disse forskelle kan være nyttige, når man fx skal forklare, hvad "Informationsvidenskab" er.

Alt dette var en runddans omkring det semantiske niveau. Semantikken defineres altså sprog for sprog og kan kun i begrænset omfang uddestillere fælles indhold. Ofte må vi ty til konteksten for at fastslå betydningen af en ytring. Konteksten er derfor afgørende vigtig at få med, når vi vil beskæftige os med naturlige sprog. For programmeringssprog er det derimod sjældent, at konteksten får lov at betyde meget. Her arbejdes som regel med *kontekst-fri grammatikker*. Det betyder, at et reserveret ord i et programmeringssprog altid oversættes til den samme kombination af handlinger i computeren, uafhængigt af konteksten.

---

<sup>8</sup> Jeg skulle betale en afgift for at få de sidste 12 eksempler med, og det sparer jeg stadig sammen til.

## Programmeringssprog

Lingvisten Noam Chomsky har brugt en definition på sprog, som afspejler en opfattelse af, hvordan et kontekst-frit sprog kan beskrives. Den har jeg fx fundet sammenskrevet således<sup>9</sup>:

”Chomsky’s language/grammar definition:

- A **language** is a (infinite) set of sentences, each finite in length and constructed out of a finite set of elements.
- A **grammar** is a device that separates the grammatical sequences from the ungrammatical sentences and generates the structure of the grammatical ones.
- A grammar is a reconstruction of the native speaker’s competence, his ability to generate (produce and understand) an infinite number of sentences
- A grammar is a theory of a language. It must comply with the empiristic axioms:
  - *adequacy*: It must generate all sentences of L. And only the sentences of L. It must describe (explain) the structures of the sentences.
  - *simplicity*: It must be as simple as possible (e.g. in terms of “length”, number of rules ...)

The axioms replace the “discovery procedures” of classic structuralism (e.g. the commutation test)”. (Tom Brøndsted, AUC, hjemmeside)

Bemærk altså:

- her tales slet ikke om betydning
- et sprog forudsættes at bestå af sætninger, der kan skabes ud af et begrænset ordforråd (elementer)
- Langt hen ad vejen er det en god definition, men der er nogle usikkerheder mht. hvordan man opdager, om en sætning er en del af et sprog
- det er vigtigt for denne type definition, at kommutationsprøven<sup>10</sup> bliver sat ud af kraft, da den forudsætter menneskelig bedømmelse. I stedet forsynes maskinen med en simpel ordbog, som indeholder oversættelse af reserverede ord til maskinens primitive operationskommandosprog.

---

<sup>9</sup> her i en formulering af Tom Brøndsted, AUC.

<sup>10</sup> Det klassiske eksempel: “hvad er forskellen på en kat og en hat?” – “ét bogstav!” Eller rettere – eet enkelt bogstav kan helt ændre ordets betydning, så det ene bogstav er afgørende vigtigt. Denne test kræver menneskelig indsigt i, hvad ord betyder. – Og “khat” er iøvrigt et stimulerende planteprodukt fra mellemøsten. Man kunne sige, at fx NuDansk Ordbog er en oversigt over kommutationsprøvens resultater for dansk sprog.

Programmeringssprog kan ligesom naturlige sprog beskrives på tre niveauer: leksikalsk, syntaktisk og semantisk niveau.

Men her er vi i det sprogligt set simple tilfælde, der svarer til, at vi taler om enkle handlinger som at sove, spise, danse osv. Vi kan på enkel måde oversætte elementerne i et programmeringssprog til computerens interne koder, både de enkelte dele af ordforrådet og deres syntaktiske sammenstilling.

Alle maskiner som vi p.t. bruger i dagligdagen er lavet efter v. Neumann principperne, og dem kommer vi tilbage til senere. Den store fordel herved er, at de i et eller andet omfang arbejder ens. Vi behøver ikke til sætte os ind i forskellige begrebssystemer på grund af forskellige maskintyper. Dog skal vi regne med nogle varianter, idet der findes mere end én familie af programmeringssprog.

Det er først, når vi taler om forskellige programtyper – tekstbehandling, databaser, design/layout programmer osv. at vi skal sætte os ind i andre begrebssystemer. Og i mange tilfælde er disse begreber overtaget fra praktisk arbejde, sådan som det foregik før maskinerne kom til. Denne lighed er i hvert fald vigtig for brugere, som skal skifte arbejdsmåde i forbindelse med overgangen til computerarbejde, at de fortsat kan gøre brug af deres faglige færdigheder og erfaring.

Et *tekstbehandlingsprogram* er lavet ud fra en metafor med en skrivemaskine og et stykke papir. Det var fordelagtigt mhp. at få overført administrative arbejderes kompetencer fra skrivemaskinen til det ny værktøj, tekstbehandlingsanlægget. En *database* tager udgangspunkt i at efterligne en kartotekskasse fuld af kartotekskort. Sådanne metaforer kan være blevet overflødige med nye generationer som Jeres (– hvor mange af Jer har skrevet mon på gammeldags elektriske skrivemaskiner? eller manuelle skrivemaskiner?) Og alligevel er kartotekskort-metaforen en nem måde at forstå et databaseprogram på, især første gang man støder på det.

Efter hvilken metafor er så selve programmeringssprogene lavet? Efterligner de så menneskesprog eller hvad? Nej, kun i begrænset omfang – men vi kan alligevel ikke se bort fra de naturlige sprog.

Programmeringssprog henvender sig som nævnt til en meget fattesvag indretning. Computeren kan i princippet kun udføre meget primitive handlinger, og situationen kræver, at vi sætter os ned på dens niveau, før vi kan instruere den. Dog er der den dobbelthed, at vi ikke bare taler med selve maskinen, men ofte faktisk taler mere eller mindre direkte til en kommende bruger af computeren, når vi programmerer, så vores programmering skal have et dobbelt fokus: sprogformen skal computeren kunne forstå – indholdet (det kommunikerede) skal bruger kunne relatere sig til. Brugeren taler til gengæld et naturligt sprog. Også i omgangen med maskinen er vi nødt til at relatere os til sproglig (billedmæssig, lydlig osv) kommunikation med andre mennesker, vores slutbrugere.

### Lidt om variable og tilstand

En afgørende forskel på maskiner og mennesker er, at vi meningsfuldt kan tale om, at en maskine er i en *tilstand*. På det allermest grundlæggende niveau kan enhver del af den fx være *tændt* eller *slukket*. Men til vores formål her kan vi sige, at computeren er i stand til at gemme en eller flere talværdier i såkaldte *variable*. En variabel er altså et sted, hvor vi gemmer talværdier.

Et simpelt eksempel kunne være et tælleapparat. Dets tilstand udgøres af et tal, der fortæller, hvor mange mennesker er gået igennem tælleapparatet siden i morges. Hver gang en ny person går igennem, tælles variabelen op med een. Vi navngiver selv variabelen; i næste eksempel hedder variabelen ”tælleren”. Hvis vi leger, at programmet sidder i selve tælleapparatet, kunne det se ud som følger:

```
når (en_person_drejer_tællehjulet)
    udfør(tæl_antal_én_op)
    vis(på_display, tælleren)
slut_når
```

lidt mere detaljeret set, kan vi definere ”tæl\_antal\_én\_op” på følgende måde:

```
tildel(tælleren) værdien (tælleren + 1)
```

Maskinen skal altså hente værdien fra den celle, der hedder ”tælleren”, lægge tallet 1 til og lægge resultatet tilbage i variabelen. Den kan derefter vise resultatet af operationen fx på skærmen. Hvordan ligner det måden mennesker tæller på?:

- *Kan du tælle til 10?*
- *ja, selvfølgelig kan jeg det! 1, 2, 3, 4, 5, 6, 7, 8, 9, 10!*

Dette virker på overfladen anderledes end når vi bruger maskinen til at tælle, men faktisk minder de to fremgangsmåder om hinanden. Mennesker er også nødt til at fastholde, hvor langt vi er kommet, mens vi tæller. Små børn gentager ofte et af tallene, fordi de glemmer hvor langt de er kommet i talrækken. Man er faktisk nødt til at ”gemme mellemresultatet”, når man tæller – også selv om man ikke er en maskine. Det er en del af tælleriets natur.

### Det leksikalske niveau

I eksemplet med tælleren er nogle ord fremhævet med halvfed type. Det er de ord, som maskinen skal kunne genkende, og som kun må bruges til de specificerede formål. Ordbogen – *leksikonet* i et programmeringssprog – består som regel af ikke ret mange ord. De omtales ofte som programmeringssprogets *reserverede ord* eller *keywords*. De er forskellige fra sprog til sprog, om end mange af dem udviser famlieligheder . En kort oversigt viser, at nogle dog går igen. Følgende er eksempler fra forskellige programmeringssprog:

**assignment**

alder = 17	fx. C, C++, Java, Python
alder := 17	fx. Pascal
set the value of alder to 17	fx. HyperTalk, Lingo
alder <- 17	fx. SmallTalk
17 -> alder	fx. Beta

Assignment er den engelske betegnelse for, at vi gemmer en talværdi i en variabel.

Vi kan også opstille betingelser for, hvornår en operation skal udføres:

<b><u>betingelse</u></b>	(opstil en ramme for udførelsen)	
if alder < 18 then put "Adgang forbudt" endif		Pascal
if alder < 18: print "Adgang forbudt"		Python
if (alder < 18) fprintf("Adgang forbudt");		C
if (alder < 18) system.output.writeln("Adgang forbudt");		Java
if (ALDER < 18) THEN LPWRITE("Adgang forbudt");		COBOL

For at opnå at få et fælles meta-sprog, som kan bruges til samtale om programmering, uden at programmører og andet godtfolk behøver at arbejde i det samme sprog, har man opfundet et antal **pseudosprog**. De indeholder fællesmængden fra en række konkrete sprog, og kan siges at være programmeringssprogenes kunstsprog ligesom esperanto, ido og volapük er det naturlige sprogs kunstige paralleller.

Et pseudosprog prøver – ved at standardisere. dvs. at udligne forskellene mellem programmeringssprog, så man kan koncentrere sig om det vigtigste – at tale om løsning af særlige opgaver på computer. Det generaliserer et ordforråd til at programmere med

**Lærebogens pseudokode**

Vores bog bruger pseudokoder i sin beskrivelse af, hvordan et computerprogram kan løse sine opgaver. Pseudosprogets leksikon er følgende:

<b>assign</b>	tildel værdi
<b>do</b>	udfør
<b>else</b>	ellers
<b>if</b>	hvis
<b>procedure</b>	definer rutine
<b>the value</b>	definer variabel_navn
<b>then</b>	hører sammen med if
<b>while</b>	så længe

**Fig. 5.** Reserverede ord i Brookshear: Computer Science, 7<sup>th</sup> ed. s. 161ff

Ovenstående er kodeord, som hver for sig løser så små problemer, at deres relevans umiddelbart kan være svær at se. De er blot ”de brikker vi har at

flytte med”, når vi skal til at løse rigtige problemer ved hjælp af en computer. Vi er nødt til at omformulere opgaverne, så de kan udtrykkes ved hjælp af sådanne kodeord, også kaldet *reserverede ord* eller *keywords*.

### Et pseudo-naturligt sprog

Før vi kigger på, hvordan disse kodeord skal bruges, vil jeg hoppe over til et frit opfundet ”naturligt” miniature-sprog. Det gør jeg først og fremmest for at vise brugen af de vigtigste enheder i beskrivelsen af programmeringssprog. En af mine kolleger bruger dette eksempel, og jeg har fået lov at bringe her også. Det drejer sig om ”*Pico-dansk*”, som er et sprog for et tænkt (jysk?) folkefærd, der vist ikke taler ret meget og heller ikke om ret mange ting. Vi kan bruge diagrammer til at beskrive, hvordan sætninger bygges op i pico-dansk tale.

Bemærk tilgangen til beskrivelsen. Vi kigger kun på, hvordan sætninger bygges op, ikke på betydningsindholdet. Sætninger, der opfylder reglerne for korrekt opbygning, er korrekte og kan siges at tilhøre sproget; også selv om indholdet er det reneste sludder.

Men først skal vi kigge på, hvilke symboler vi kan bruge, når vi taler om sprog. Der er kun tre forskellige slags symboler: pile, ovaler og rektangler.

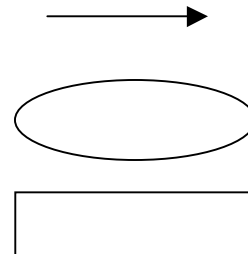
### Syntaksdiagrammer

Et *syntaksdiagram* er et navngivet diagram, der består af en række ‘pile’ afbrudt af symboler og af navne på syntaksdiagrammer.

Et syntaksdiagram defineres alle de følger af symboler, som opnås ved at gennemløbe diagrammet i pilenes retning.

Symboler kaldes *syntaktiske entiteter*

Diagrammer kaldes *syntaktiske kategorier*<sup>11</sup>



**Fig. 6.** Symboler, der anvendes til beskrivelse af syntaks

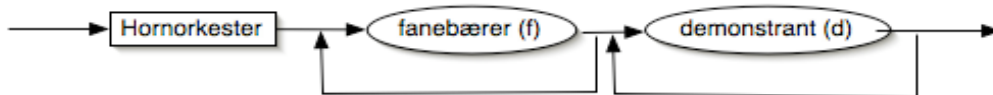
Før vi kigger på pico-dansk skal vi se på mere generelt, hvordan det er muligt med et syntaksdiagram at lave sekvenser. Sekvenser er i denne forbindelse følger af tegn eller symboler, som alle følger en fast grammatik.

Vi tager som eksempel en demonstration (det kunne også være en kamelkaravane), da den kan betragtes som en “følge af deltagere”. Syntaksdiagrammet kan så siges at udtrykke, i hvilken rækkefølge deltagerne skal optræde

<sup>11</sup> efter Michael E. Caspersen, 2000

efter “reglerne”. En demonstration har fire slags deltagere her i vores eksempel-verden. Det kan være en musiker (trompetist eller basunist) en fanebærer eller en almindelig demonstrant. Vi holder os til dette meget enkle niveau.

Demonstration



Hornorkester



Fig. 7. Enkel grammatik for sekvenser af deltagere i et demonstrationstog

NB! Det er vigtigt at indse følgende: disse diagrammer er ikke *billeder* af demonstrationer, de er *diagrammer*, der beskriver, hvordan en følge af deltagere i et demonstrationstog ”skabes”. Hvis det var et billede, ville demonstranterne se ud til at gå forrest – men iflg. grammatikken går messingblæserne forrest. Den kortest tænkelige demonstration består således af én messingblæser, én fanebærer og én demonstrant.

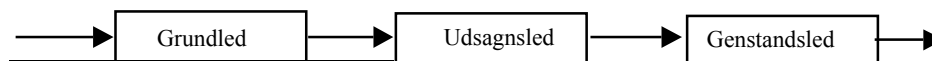
Nu kender vi “grammatikken” for en demonstration. Den kan bruges til at checke, om rækkefølgen af deltagerne overholder “reglerne for sammensætning af demonstrationer”. Overvej, om følgende sekvenser overholder syntaksen (forkortelser, se figuren):

- ttbtffd
- tbtff
- tftbd
- ttbtbfdddd<sup>12</sup>

Grammatikker viser altså “lovlige rækkefølger” af tegn, og specificerer på den måde, hvorpå en sekvens af fx. ord (eller demonstrationsdeltagere) skal konstrueres.

I det følgende kommenteres **syntaksen for pico-dansk**. Vi tager det linie for linie:

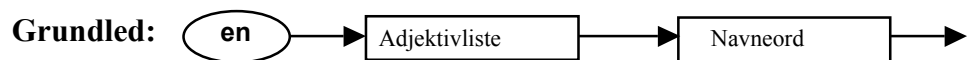
**Sætning:**



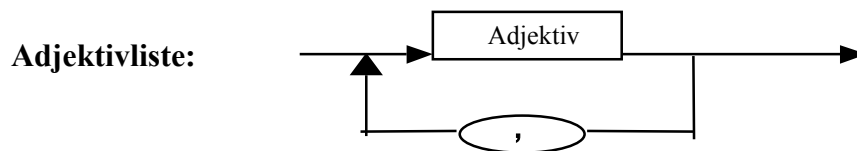
<sup>12</sup> facit: ttbtffd OK; tbtff forkert – mangler d, tftbd forkert – ikke t efter f, ttbtbfdddd OK.

Denne regel siger, at en korrekt Pico-dansk sætning skal starte med et grundled (subjekt), fortsætte med et udsagnsled (verbum) og slutte med et genstandsled (objekt). Vi kunne altså sige, at vi definerer Pico-dansk som et SVO-sprog. Denne regel betyder fx., at vi i Pico-dansk kun kan lave hovedsætninger.

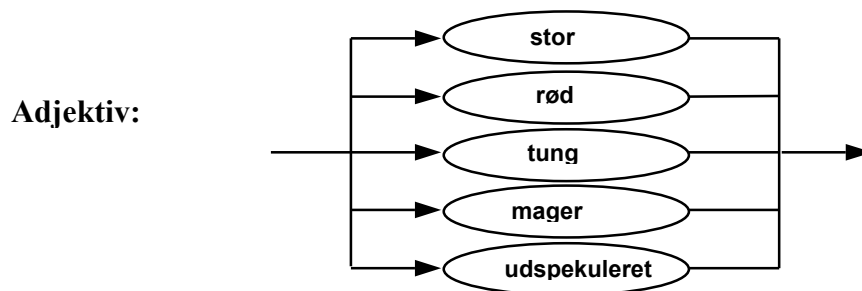
Dernæst skal vi bruge en regel for, hvordan vi laver et grundled:



alle sætninger starter altså med den ubestemte artikel "en". Her ser vi næste begrænsning: der findes i Pico-dansk hverken bestemt form eller flertal. Det er hermed ikke endnu formuleret, hvordan en "Adjektivliste" skal konstrueres. For at vise, hvordan sådan en ser ud, bringes flere syntaksdiagrammer:



Vi ser, at en "adjektivliste" simpelthen består af et eller andet antal tillægsord (adjektiver), som anbringes efter hinanden med et komma imellem. Og af "Adjektiv"-kategorien fremgår det herunder, at vi har fem tillægsord at vælge imellem: "stor", "rød", "tung", "mager" og "udspekuleret".



I semiotikken bruges begreberne syntagme og paradigme. Uden at ville komme for alvor ind på disse, kan det nævnes, at der er slægtskab mellem vores anvendelse af begrebet "sætning" og begrebet *syntagme*, ligesom "kategori" er beslægtet med begrebet *paradigme*.

Vi kan prøve at lave nogle sætninger i dette "sprog". Er følgende sætninger "lovlige" ifølge grammatikken for Pico-dansk?:

- *en gammel mand graver et stort hul*
- *en udspekuleret, mager hest spiser en stor balle hø*
- *den tunge mand skovler en stor skovl*
- *en mager bil kører en tung mand*
- *en tung mand kører en rød bil*

Det må hermed være klart, at korrekt syntaks ikke i sig selv giver meningsfulde sætninger. Kan I lave en sætning, som kunne give mening?

*Øvelse for lidt viderekomne med lyst til syntaksdiagrammer.*

Prøv at indføre følgende ændringer i syntaksdiagrammet:

- vi vil gerne kunne undvære objekt i sætningerne; fx. “en rød hest sparker”
- vi vil gerne have et ”og” i stedet for det sidste komma
- vi vil gerne indføre biord (adverbier) i Pico-dansk

Bemærk

Vi kunne også have vist alle detaljer i sproget udskrevet i eet stort diagram, hvor alle mulige ”ruter” var tegnet ind. Men den anvendte måde får alle muligheder med, og de anvendte diagrammer sparer plads i præsentationen af syntaksen.

### **BNF syntaks**

Syntaksdiagrammer fylder forholdsvis meget, så der er opfundet andre måder at beskrive syntaks på, fx. EBNF (Extended Backus-Naur notation). Pico-dansk ville se således ud i EBNF-notation.

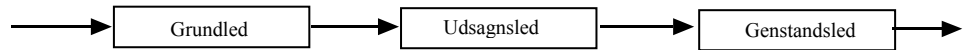
```
sætning ::= grundled udsagnsled genstandsled
grundled ::= en adjektivliste navneord
udsagnsled ::= spiser | kører | sparker | skovler
genstandsled ::= grundled
adjektivliste ::= adjektiv { , adjektiv } *
adjektiv ::= stor | rød | tung | mager | udspekuleret
navneord ::= mand | bil | hest | skovl
```

Det er jo en meget koncentreret udtryksform, men også vanskeligere at læse.

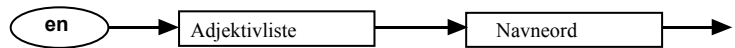
### Pico-dansk: samlet syntaks

efter Michael E. Caspersen 2000

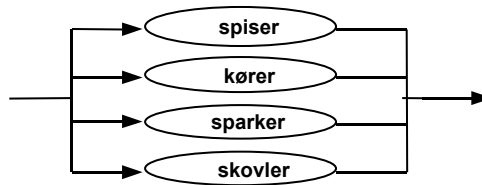
**Sætning:**



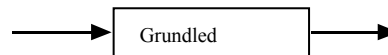
**Grundled:**



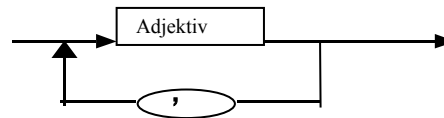
**Udsagnsled:**



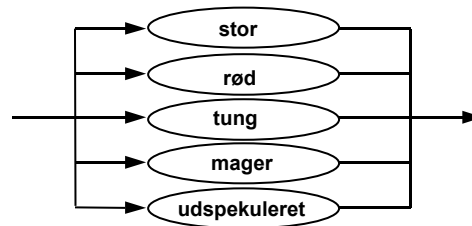
**Genstandsled:**



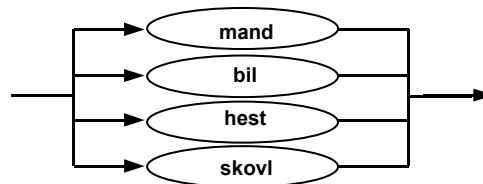
**Adjektivliste:**



**Adjektiv:**



**Navneord:**



### Naturlige sprog (NL) og programmeringssprog (PL) sammenlignet

En kort sammenligning mellem de egenskaber, som naturlige sprog har i forhold til programmeringssprog kan opremses som i følgende opstilling. Det er selvfølgelig en første og ret primitiv sammenligning, men den antyder alligevel vigtige forhold.

Begreb	Naturlige sprog NL	Programmeringssprog PL
leksikon	stort	lille
syntaks	flere korrekte muligheder	een korrekt
semantik	kompleks	entydig, enkel
tvedydighed	et udtryksmiddel – ironi fx.	forbudt
sociale funktioner	mange	ingen – men pragmatisk succes
afsender	den talende	implicit, rekvirent eller programmør
modtager	indbygges i udtryk	maskinen – implicit bruger
type	dialog, successiv tilnærmelse	kommando – rigtig/forkert
nye begreber	afprøves, aftales	konstrueres frit -(data/aktivitet)

**Fig. 8.** Løs sammenligning mellem naturlige sprog og programmeringssprog

Designeren af et programmeringssprog har som regel valgt kun een eneste måde hvorpå programmøren kan angive, at han vil have en bestemt detalje-aktivitet udført af maskinen. Varianter forekommer normalt ikke, omend en opgave stadigvæk som regel kan udføres på adskillige måder.

Et naturligt sprog har behov for et stort ordforråd, ligesom det har behov for at kunne udtrykke nuancer i andre henseender. Forskellige udtryk vil ofte dække over nuanceforskelle i betydningen.

Tilsvarende kan vi se vedr. syntaksen, at naturlige sprog har en varieret syntaks, hvor subtil nuancering kan finde sted. Tænk blot på, hvordan jurister kan undersøge, vende og dreje en formulering af en lovparagraf for at finde ud af, om den passer på netop en foreliggende situation eller ej. I dagligdags kommunikation bruger vi faktisk som regel ufærdige formuleringer, som alligevel forstås af vores omgivelser, fordi de i forvejen er medvidende i, hvad vi taler om, og hvad vi plejer at mene. Selv meningsløse ytringer kan bringes til at betyde noget, blot vores for-forståelse er tilstrækkelig stor.

Og – igen – er programmeringssproget nådesløst: enhver sætning skal udformes på en ganske bestemt måde, før maskinen lader den passere som "forståelig". Oversætterprogrammet tillader kun én type formulering af en bestemt (primitiv) handling. Derfor opfatter mange førstegangbrugere en oversætter som "besværlig" eller "tvær", og i hvert fald passer det, at den skal være nøjeregnende.

### Hvem taler programmøren til?

Det kunne se ud til, at en programmør blot skal lære sig at tale sin maskines sprog, så kan han programmere og få den til at udføre alle de opgaver han har lyst til at få udført.

Men helt så enkelt er det alligvel ikke, når man kigger på sagen i en lidt større sammenhæng. Det er let at lade sig snyde og indbilde sig, at blot maskinen forstår meningen med en sætning (semantikken er korrekt), så er det program i orden. Det duer dog ikke; som regel er programmet rettet mod en større offentlighed eller større gruppe. Så designeren og programmøren af et program må hele tiden have for øje, hvilken bruger han henvender sig til. Dette forhold er ganske svært at efterleve, og der er da også udviklet særlige fag, som forsker i passende metoder hertil. Der er så at sige to slags semantik: den for maskinen og den for brugeren.

Vi kan betragte computeren og vores program som et medie, hvorigennem vi udtrykker os til en modpart. Kommunikationen er en mellemtung mellem envejs- og tovejskommunikation. Den er envejs i den forstand, at vi selv og på forhånd skal gennemtænke alt, hvad vi ønsker at meddele til vores publikum – brugeren. På den anden side er mediet interaktivt, så vi kan ikke helt vide, i hvilken rækkefølge vores meddelelser vil blive præsenteret for brugeren. Derved kan planlægningen af kommunikationen blive ganske omfattende – vi skal tage højde for alt, som kan ske; alle reaktioner, som vores bruger kan tænkes at have. Brugeren vil under interaktiv kommunikation foretage en udvælgelse, så vi må tage stilling til, hvilke faste spor vi vil lægge og hvilke løse tilføjelser til dette spor vi ønsker.

Det kan nu også være ganske omfangsrigt at gennemtænke de situationer, maskinen selv kan tænkes at komme i. Har vi opstillet en betingelse, bliver vi nødt til at gennemtænke både hvad der skal ske hvis betingelsen er opfyldt, og hvis den ikke er opfyldt. Hvis en simuleret ekspedient kun må sælge spiritus til folk over 15 år, skal den også have instruktion om, hvad den skal gøre, hvis en kunde under 15 år ønsker at købe spiritus. Ellers vil han gøre ingenting, så en 14 årig simuleret kunde ville blive ignoreret. I praksis giver betingelser altid anledning til ekstra overvejelser.

### Algoritmer

De færreste bruger begrebet “algoritme” i dagligdagen. Vi taler om *opskrifter*, *fremgangsmåder*, *procedurer* og lignende, og disse kommer meget tæt på det tekniske begreb algoritme. Det er iøvrigt et låneord fra arabisk, og det defineres i Brookshear således: (bemærk, at der er en trykfejl i fig. 4.1 i 7. udg. af bogen. Den korrekte version er her)

*An algorithm is an ordered set of unambiguous, executable steps, defining a terminating process.*

Vi skal senere kigge mere i detaljer på denne definition; her vil vi kun se på nogle af formuleringerne. En algoritme består altså af en sekvens af trin, der hver for sig er utvetydige og kan udføres. Definitionen kræver også, at processen skal holde op igen; vi skal kunne vide at den ikke fortsætter uendeligt. Reglen gælder dog ikke for programmer, som fx. kontrollerer kraftværker eller robotter. Deres algoritmer kører uafbrudt, så længe robotten eller værket kører. Men i den klassiske, datalogiske version tænkte man ikke på den slags algoritmer.

En almindelig madopskrift er et udmærket eksempel på en algoritme, selvom den ofte er udformet som fortløbende tekst, hvor de enkelte trin ikke er meget tydelige.

Jeg har fundet et eksempel <sup>13</sup>:

### 932. Forlorne spejlæg

1 rouladebund  
 1/4 l kagecreme  
 100 g abrikoser (store, ensartede)  
 2 teskf. melis  
 2 dl piskefløde

*En rouladebund bages (se nr. 931 II). Udstikkes i runde kager med glas eller kolonneform.*

*På disse kager anbringes cremen, der må være ret stiv. Abrikoserne udblødes fra den ene dag til den anden og koges i udblødningsvandet, der sødes med de 2 teskf. sukker. Flødeskum smøres over cremen, og en abrikos anbringes midt på hvert "spejlæg".*

I opskrifter er mange ting underforstået. Prøv at forestille dig at følge denne opskrift punkt for punkt, ligesom en maskine vil skulle gøre det. Man kan forestille sig, at de forskellige ingredienser skal afmåles i små containere. Angående rouladebunden, er der henvist til en anden opskrift, hvor bagning af sådan en er gennemgået. Denne detalje ligner programmeringsmetodik meget: man afgrænser en del af fremgangsmåden som en "delopskrift", kaldet *rouladebund* der kan udføres separat, når en hvilken som helst anden opskrift har brug for det. Så er man fri for at gentage fremgangsmåden i alle opskrifter, som skal bruge en rouladebund. I pseudokode kunne det formuleres sådan:

```
procedure rouladebund ()
    her står så, hvordan man fremstiller en rouladebund
    (det viser sig, at denne procedure varer under 1 time at udføre)
end_procedure rouladebund
```

Et andet problem for en maskine kunne være sætningen: *Abrikoserne udblødes fra den ene dag til den anden...* Nu skal der pludselig gå 12-24 timer, inden vi kan gå videre eller hvad? Nej, selvfølgelig skal man have læst opskriften igennem og så have forstået, at abrikoserne skulle sættes i blød dagen før man gik i gang med at lave forlorne spejlæg. Det er helt normal

<sup>13</sup> Ingeborg Suhr: *Mad*, 25. oplag, Gjellerup 1968

madlavningspraksis. Det samme gælder den tid, der skal bruges for at fremstille rouladebunden, selv om det er mindre end en time.

Vi kan også sige, at en normal, menneskelig hjemmebager vil kigge fremad i opskriften for at finde ud af, hvad der skal gøres først. I datalogien taler man om et “*look-ahead*”; for en computer vil det normale være et look-ahead på ét symbol; maskinen læser et symbol og udfører den tilsvarende operation med det samme.

Hvis opskriften skulle skrives om, således at en (fremtidig) bagemaskine skulle kunne forstå den, ville den i pseudokode komme til at se ud noget i retning af:

```

procedure forlorne_spejlaeg
  sæt_i_bloed (abrikoser)
  vent (12 timer)
  kog (abrikoser, sukkervand)
  rouladebund ()
  while (not abrikoser_er_kolde) vent (15 minutter)
  udstik (rouladebund, cirkel)
  while (antal_abrikoser > 0 AND antal_runde_kager > 0)
    paasmoer (kagecreme (stiv))
    paasmoer (floedeskum (2dl))
    laeg_paa (abrikos)
    stil_på (fad)
  endwhile
  server
end_procedure

```

Programmet “kalder” en lang række andre procedurer, såsom floedeskum(), vent() osv. Vi må forestille os, at disse enten eksisterer i forvejen, eller at vi agter at skrive dem selv. De skal i hvert fald være til rådighed før vi kan udføre programmet *forlorne\_spejlaeg*. Vi har altså stadig en del arbejde foran os, hvis disse procedurer ikke stilles til rådighed af vores system.

Tænk over, hvor meget flødeskum, der kommer på hvert forlorent spejlæg. Der står 2dl, men hvordan bør det forstås? Måske er programmet ikke perfekt og færdigt endnu.

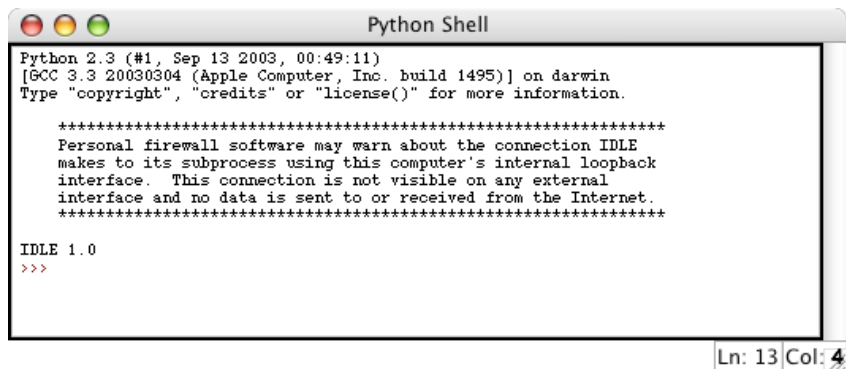
## Python

I VICA kurset har vi ikke noget program, som kan bage kager. Et af vores vigtigste værktøjer er Python. Python er et programmerings sprog på side med C, Java og Pascal. Algoritmerne vi udtænker, skal oversættes til de primitiver, som Python stiller til rådighed. For at I kan komme igang med at programmere straks fra kursusstart, vil jeg i dette afsnit præsentere nogle grundlæggende keywords fra Python, som I straks kan gå i gang med at bruge. Eksemplerne til at begynde med er der ikke meget kød på, men de er også blot beregnet på at introducere Jer til Python’s udtryksmåde.

Selve Python fortolkeren og dens udviklings miljø IDLE kan frit hentes fra <http://www.python.org>. Python kan køre på alle operativ systemer såsom Windows, Mac og Unix.

### Python prompt

Den mest direkte (og primitive) måde at snakke Python er via prompten, eller nogen steder finder man den også under navnet Python shell. Du kan få en sådan prompt frem ved at starte IDLE, som er det udviklingsmiljø som følger med Python. Det ser således ud:



```

Python 2.3 (#1, Sep 13 2003, 00:49:11)
[GCC 3.3 20030304 (Apple Computer, Inc. build 1495)] on darwin
Type "copyright", "credits" or "license()" for more information.

*****
Personal firewall software may warn about the connection IDLE
makes to its subprocess using this computer's internal loopback
interface. This connection is not visible on any external
interface and no data is sent to or received from the Internet.
*****

IDLE 1.0
>>>
    
```

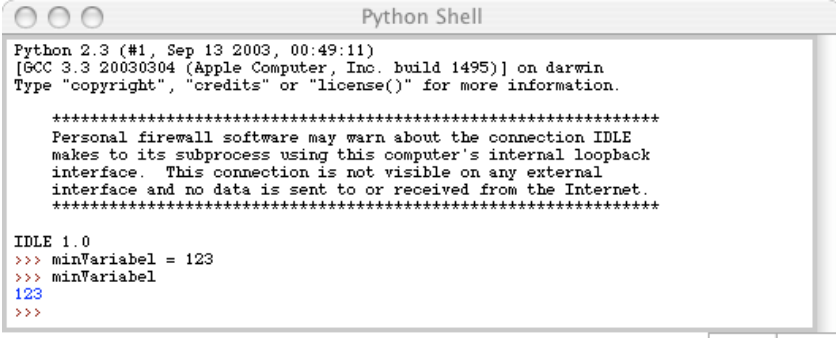
Ln: 13 Col: 4

**Fig. 9.** Dette er prompten, når man starter IDLE. Der har man mulighed for at udføre forskellige Python kommandoer direkte.

Man skriver sine Python-sætninger i dette vindue. Der vil være en cursor, hvor der står >>>, det er her man skriver de sætninger som Python skal udføre. Python fortolkeren vil også svare i samme vindue.

### Variable oprettes halv-automatisk

I Python kan du oprette en variabel blot ved at skrive dens navn og tildele den en værdi; fx: `minVariabel = 123`  
 Derefter kan du skrive den ud igen ved sætningen `minVariabel`. Se fig. 10:



```

Python 2.3 (#1, Sep 13 2003, 00:49:11)
[GCC 3.3 20030304 (Apple Computer, Inc. build 1495)] on darwin
Type "copyright", "credits" or "license()" for more information.

*****
Personal firewall software may warn about the connection IDLE
makes to its subprocess using this computer's internal loopback
interface. This connection is not visible on any external
interface and no data is sent to or received from the Internet.
*****

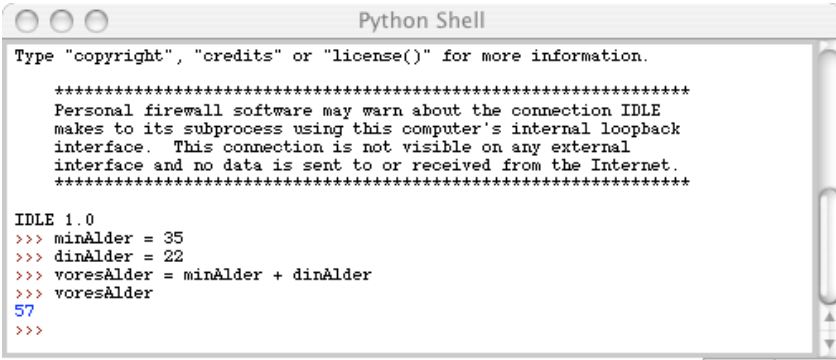
IDLE 1.0
>>> minVariabel = 123
>>> minVariabel
123
>>>

```

Ln: 16 Col: 1

**Fig. 10.** Opretelse af variabel og udskrift i Python

Alle mulige dialogiske interaktioner kan finde sted i dette vindue. Vi kan bl.a. bruge vinduet som “lommeregner”, fx. som følger:



```

Type "copyright", "credits" or "license()" for more information.

*****
Personal firewall software may warn about the connection IDLE
makes to its subprocess using this computer's internal loopback
interface. This connection is not visible on any external
interface and no data is sent to or received from the Internet.
*****

IDLE 1.0
>>> minAlder = 35
>>> dinAlder = 22
>>> voresAlder = minAlder + dinAlder
>>> voresAlder
57
>>>

```

Ln: 18 Col: 1

**Fig. 11.** Opretelse af tre variable, der derefter udskrives. Man skal oprette en variabel for at kunne gemme en værdi. Selve regnestykket kunne formuleres som “35 + 22” uden at oprette en eneste variabel.

Regnestykket i fig. 10 ser mere indviklet ud, en den måde vi normalt vil notere det på. Jeg går ud fra, at vi gerne vil kunne gemme de to aldre, og derfor har jeg oprettet en variabel til hver. Den samlede alder skal også gemmes, så der oprettes også en variabel til den. Fordelen herved er, at vi bagefter (forestil dig et stooort regnestykke) kan gå de enkelte dele af beregningen efter og checke, at input var korrekt.

## Større programmer i Python

For at kunne lave rigtige programmer, der ikke skal indtastes manuelt hver gang de skal udføres, kan man lave scripts. Et *script* er et tekstdokument, som kan læses ind i Python, oversættes til maskinsprog og udføres i et hug.

Når vi har flere linier ad gangen, kan vi begynde at tænke på fx **if**- og **while**-konstruktioner, som tidligere blev omtalt. De fylder typisk flere linier, og de skal udføres som en enhed.

## Betingelser

“ - Hvis du giver mig 1000,- kr. vil jeg klippe dit hår, servere en bedre middag og pudse din cykel. - “

Alle tre handlinger vil blive udført, hvis den samme betingelse er opfyldt. Vi siger, at “betingelsen har værdien *sand*”. Udsagnet “du giver mig 1000,- kr.” kan være sandt eller falsk. Betingelser i programmeringssprog kan ikke være *næsten sande* eller *næsten falske*. De er 100% sande eller falske.

Som eksempel kan vi tage en helt almindelig situation i spil. Spilleren har opnået en eller anden score – her benævnt “my\_score”. Spillet angiver, hvad hidtil højeste score har været – her kaldet “hi\_score\_max”. Hvis nu spilleren er den hidtil dygtigste, skal hi-score maksimum herefter afspejle dette. I Python kunne det fx se således ud:

```
if my_score > hi_score_max :
    hi_score_max = my_score
```

Udsagnet “my\_score > hi\_score\_max” er enten sandt eller falsk. Hvis spilleren har tangeret rekorden, er udsagnet faktisk falsk, fordi der bruges “>”, også kaldet “skarpt større end”. Når to tal er lige store, er det ene ikke større end det andet, så derfor er udsagnet falsk.

## Løkker

*Løkke*, *gentagelse* eller *repetition* – alle sammen angiver de, at en handling skal udføres een gang til. En sådan konstruktion er som nævnt helt grundlæggende for programmering.

Der er flere forskellige løkke-konstruktioner i Python. Her vil vi koncentrere os om den, der svarer til en standard while-konstruktion.

Situationen kan fx være følgende: jeg ønsker at et billede på skærmen skal flytte sig 100 pixels ned med ryk af 1 pixel hver gang. Den tilhørende Python kode kunne se således ud:

```
def movePicture()
    currentPos = 10
    destination = 100
    while currentPos < destination:
        currentPos = currentPos + 1
```

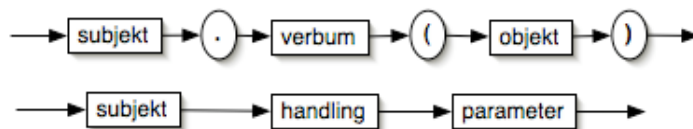
## Syntaksen i forhold til naturlige sprog

Hvordan er så rækkefølgen af “sætningsled” i en sætning i et programmeringssprog. Tidligere omtales kommandosprog som meget lig almindelige sproglige udsagn, blot henvender sætningen sig direkte til sætningens subjekt.

Hvem eller hvad er så henvendelsen til, når vi skriver en sætning som ovenstående while-eksempel? Vi kunne jo sige, at vi henvender os implicit til python fortolkeren. Virker det lidt langt ude, så bliver det nok klarere efterhånden som Pythons virkemåde bliver vane. Vi kan skrive det på forskellige måder, fx:

- *“Oh fortolker: når du får kommandoen “movePicture” skal du gentage osv. osv.”.* Eller
- `__main__.movePicture() { while... }`

Læg godt mærke til den sidste måde at notere det på. Den bruger punktum til at adskille subjektet fra verbet. Vi kan med en tilsnigelse sige, at en kommando i Python har den sproglige form:



Måske skulle vi skrive noget andet end “objekt” mellem paranteserne. Der kunne også stå “alt det andet”.

Den sproglige syntaks kommer vi tilbage til mange gange i løbet af kurset,

Rækkefølgen af sætningernes forskellige led ligner den i “pico-dansk”, så for tilhængere af små sprog kan den fornemmes naturlig.

## Afrunding

Det fremgår af ovenstående, at der kan være grund til at udnævne de kodesystemer som bruges til kommunikation med maskiner som “sprog”. Dog er der mange facetter fra vores rige hverdagssprog, som nærmest er i vejen, når vi skal programmere en computer.

Programmeringssprogenes tilsyneladende lighed med naturlige sprog er dog en vældig hjælp i omgangen med maskinerne, da vi både kan lægge betydning (i form af genkendelse af handlinger mv.) og får klart præsenteret rækkefølgen af programmets handlinger.

Det er vigtigt i omgangen med denne slags sprog at holde sig denne dobbelthed for øje. Programmeringssprogenes grænser går, hvor vi overholder krav om entydighed mht. hvad vi vil have udført. Læselighed for programøren er en vigtig egenskab, men primært for at gøre det muligt at genkende maskinens handlinger via den sproglige lighed.

Nu er de vigtigste elementer i programmeringssproget Python præsenteret, og det er muligt at komme i gang med at lære selve håndværket.

Vi ønsker god fornøjelse.