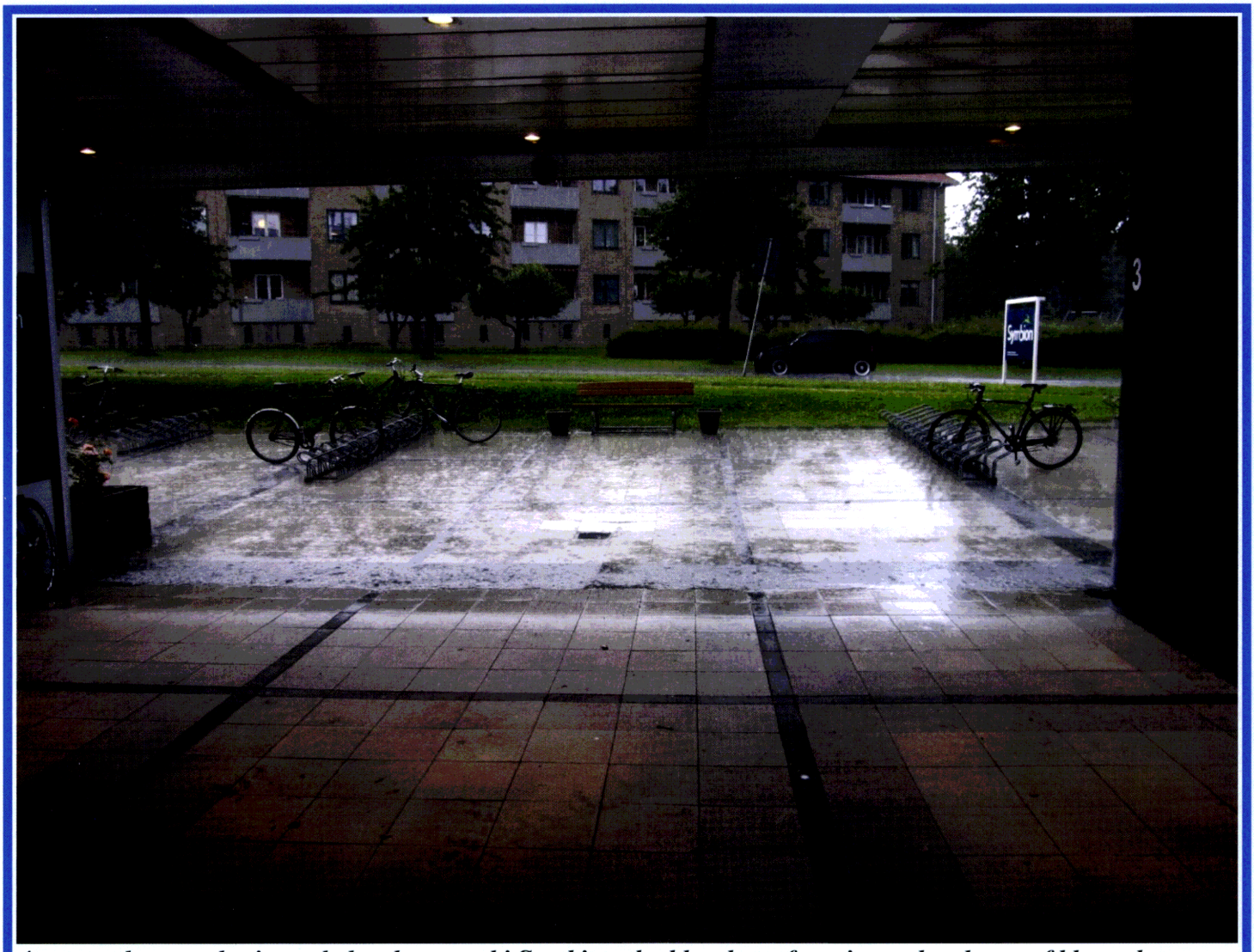
DKuug nr. 178

September 2015

NYT



4. september var der igen skybrud og vand i Symbions kældre, hvor foreningen har lager af bl.a. udstyr

Af indholdet i dette nummer:

Et brev fra redaktøren

Som foreningen, der skabte DK-net og DK-hostmaster og ikke mindst som fortaler for leverandøruafhængighed i kraft af at være brugerforening for Unix (og Posix), må DKuug til stadighed fortælle om hvorfor og hvordan deling af viden kan komme alle til gode.

Det skal ikke være nogen hemmelighed, at emnerne i dette nummer har voldt flere problemer end redaktion af tidligere numre. Undervejs gennem arbejdet med Prolog, måtte redaktionen flere gange overveje, om forklaringerne var systemmæssigt tilfredsstillende eller om det var hyggesnak om hvor let det er at programmere med Prolog dette gamle specielle logik-programmeringssprog.

Ideen var at begynde et arbejde (artikel-research) omkring "intelligente devices", videreførelse af tråden fra FI-ware i nr. 177.

Computerprogrammer, som kan svare på spørgsmål, eller spille skak, finde vej, beregne den korteste rute, finde laveste billetpris o.s.v. har ændret vores hverdag i en grad, så vi ikke ville kunne undvære det - jo, det kunne vi jo nok, ligesom spejdere kan leve i en primitiv lejr i nogle dage, men vi ville ikke synes det var morsomt.

Man har knyttet ordet "intelligens" til computeren allerede fra de tidligste stadier, hvor computere blev brugt til at beregne "påsken i 2030" eller "π med 100 cifre" og den slags. Det er ikke noget, vi almindelige mennesker lige går hen og gør, men hvis man kan beregne π med bare 4 cifre - eller slå det op i en logaritmetabel, så er man lidt intelligent. Hvis man kan stave "rigtigt", så er man lidt intelligent. Hvis man kan forudsige vejret ud fra et vejrkort, så er man dygtig. Hvis man kan svare på forskellige vanskelige spørgsmål, så har man en god hukommelse og almen bred orientering.

En computer, der kan disse ting, kunne ligne noget, der er intelligent, men de, der arbejder med det - ja endog de mennesker, der forsker i Artificial Intelligence, ved at betegnelsen "intelligens" er misvisende - der er blot tale om et forskningsfelt, der går fra data-lingvistik over statistik og måledevices til maskinstyring og robotteknik.

I dette nummer skulle begrebet "Artificial Intelligence" (AI) beskrives på en måde, så at både nye medlemmer og deres venner og bekendte fik en idé om mekanikken i de såkaldt intelligente systemer. Denne pseudo-intelligens begynder med at foretage string-compare, ordbogsopslag, og indordne alting i kategorier og underkategorier, eller domæner og subdomæner, som bestemmer regler for hvordan ordene skal forstås.

De bedste eksempler på det er søgemaskinernes måde at opdele en søgning i mulige emner, og jo flere ord der er, desto færre emneområder kan der være tale om. Men at stille spørgsmålet og sætte et mål, det er en helt anden mekanisme, som kun mennesket besidder.

Derfor kan AI vise sig at være den bedste måde at få lært hvad mennesket egentlig selv kan og hvad der er data, og hvad der er motiv eller drivkraft.

En uventet drejning

Under arbejdet med Prolog og søgning efter virkelig store programmer baseret på Prolog, fandt redaktionen henholdsvis Watson, IBM's "kloge" maskine, der kunne deltage i Jeopardy og som (naturligvis) kunne slå alle. De to grundlæggende elementer i Watson er en database med viden og en sprog-analyse baseret på Prolog (matching af mønstre).

En uventet drejning

Pludselig fandt vi ud af, at der også er firmaer, der har markedsført logik-programmeringssystemer (der er andre end Proglog) til at lave regelbaseret sagsbehandling, kommercielle systemer, som skulle kunne være en nem måde for virksomheden at oprette egne regler, der så blev mere eller mindre oversat til logik-programmering, og derefter - som i Prolog - kunne systemet finde de mulige svar og sortere dem efter egnethedskriterier (constraints) og lignende.

Mens redaktionen tyggede på det, kom nyheden om at SKAT har mistet ca. 40000 millioner kroner ved at tilgodehavender er blevet forældet. SKATs **EFI** (Et Fælles Inddrivelsessystem) var på nyhedernes forsider igen og igen, og til sidst måtte skatteministeren lukke systemet ned.

Men da alle artikler om emnet handlede om hvor forfærdeligt det var, og ingen handlede om hvorfor systemet fejlede - så måtte redaktionen jo nødvendigvis yde sit lillebitte bidrag ved at deducere, hvordan systemet er bygget op. Et nøgleord var *debitormotor*.

Prøv at søge i Google på //debitormotor//, og se, der kommer blandt andet en CW artikel fra 2012: Skat: Forsinket CSC-projekt får konsekvenser for os - Problemerne med CSC-projektet Debitormotor får konsekvenser for Skat, der nu vender blikket mod sig selv. [...] dertil kommer andre artikler og også dokumenter fra SKAT om hvad *Debitormotor* er og hvordan det må bruges. Det er, lidt forenklet sagt, et program, som skriver rykkere og bruger input af regler som ligger i databaseform, - regler, som bør være udskiftelige.

Men det er ikke så nemt, når input ikke er styret af en syntax som i et progammeringssprog - og som måske er fejlagtigt, selvmodsigende osv og hele projektet exploderer i komplexitet.

En parallel i gadget-verdenen

For et par uger siden kom Sony med en Xperia Z5 Premium smartphone, der som det første har en 4K skærm - 3996 × 2160 eller højere. Det er usædvanligt. Samtidig tilbyder dimsen en autofokus der fungerer på 0.03 sek. hvilket er helt fantastisk, som enhver foto-entusiast vil vide.

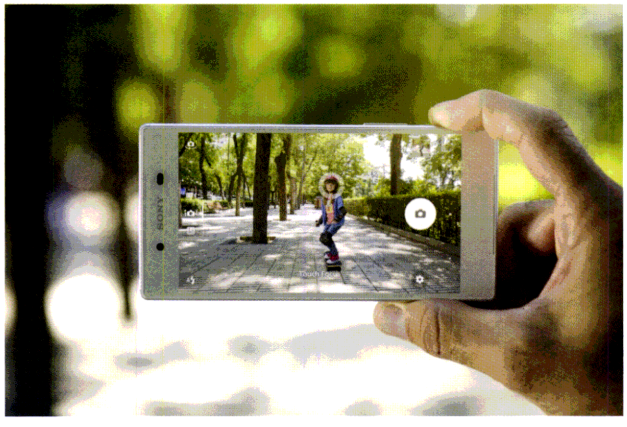
Til sammenligning viser et Canon PS-SX-170 IS lommekamera ca. 400 × 600 på den lille 7.5 cm. LCD bagside.

Man kan selvfølgelig også købe 4K fladskærme, og det har man kunnet i nogen tid. Men et af problemerne, der opstår, er at næsten ingen grafikkort kan fodre en 4K skærm hurtigt nok til at et computerspil kan vises i glidende bevægelser.

Det er selvfølgelig af minimal betydning, når det drejer sig om underholdning, spil - fornøjelse. Det er "trash-data", no offense.

Men det burde minde gadget entusiaster og statens IT-ansvarlige om at en teknik, der er meget ny, meget vel kan have indbyggede selvmodsigelser som gør det ubrugeligt.

Donald Axel



DKuug-NYT er medlemsblad for DKuug, foreningen for Åbne Systemer og Internet
Nr. 178 - September-Oktober 2015

Udgiver:
DKUUG
Fruebjergvej 3
2100 København Ø
Tlf. 39 17 99 44
email: blad@dkuug.dk
Redaktion:
Donald Axel (ansvarshavende)

Forsidecredits:
redaktionen - skybrud ved Symbion 2012; ved skybryddet i år sørgede formanden for at der ikke skete skader - tak til Dennis Jørgensen!

Forsidebilled på nr.177 var havareret pga.en bug i Libre Office.

Design og layout:
DKUUG/Donald Axel
Annoncer:
pr@dkuug.dk
Tryk:
Lasertryk i Aarhus
Oplag:
300 eksemplarer

Artikler og inlæg i DKUUG-Nyt er ikke nødvendigvis i overensstemmelse med redaktionens eller DKUUGs bestyrelses synspunkter.

Eftertryk i uddrag med kildeangivelse er tilladt.

Deadline for nr. 179: Lørdag 31. Oktober 2015

Medlem af Dansk Fagpresse
DKUUG-Nyt
ISSN-1395-1440

Vores møder og **foredrag** holdes - med mindre andet udtrykkeligt angives - på vores adresse:

DKUUG
SYMBION
Fruebjergvej 3
2100 København Ø

Hvis man kommer lidt før, er der tid til en snak på kontoret. DKUUG bor i en virksomhedsfarm, Symbion, hvor der er åbne døre indtil kl.18 eller 19 (afhængig af mødetidspunkt). Efter den tid har vi på foredragsaftener en vagt ved døren.

INDHOLD:

Symbion by night - medlemsaftener på de lange vinteraftener

Arrangementer:

Generalforsamling 11. november kl. 19 2015

Onsdag: Første onsdag i hver måned er medlemsaften på kontoret fra kl.18 ca. - somme tider 17 eller endda før. Der er spisning.

Gør-det-selv foredrag:

Vores kontor i Symbion giver mulighed for hands-on workshops, både dag og aften. Skriv til pr@dkuug.dk eller til bestyrelsen i DKUUG, bestyr@dkuug.dk, og hør om lokalet er ledigt den dag du vil arrangere et møde. Der er hurtig internetforbindelse, både wired og wireless. Er der større tilmelding, kan vi leje mødelokaler i Symbions mødested. Spørg os!

Søgning og vidensdatabaser

Først havde vi Alta Vista - udsigten fra et højt punkt - og så kom Yahoo, Google, MSN, Bing, og hvad er så Wolfram?

Egentlig skulle man også nævne Wikipedia og Archive.org, vidensdatabaser, for nogle af dem har deres egen søgemaskine, men det er mere og mere almindeligt at et site med artikler, billeder og/eller video'er overlader søgningen til Google og evt. andre søgemaskiner. Google har en service, hvor man kan tilmelde et site for at sikre sig at det faktisk bliver indexeret.

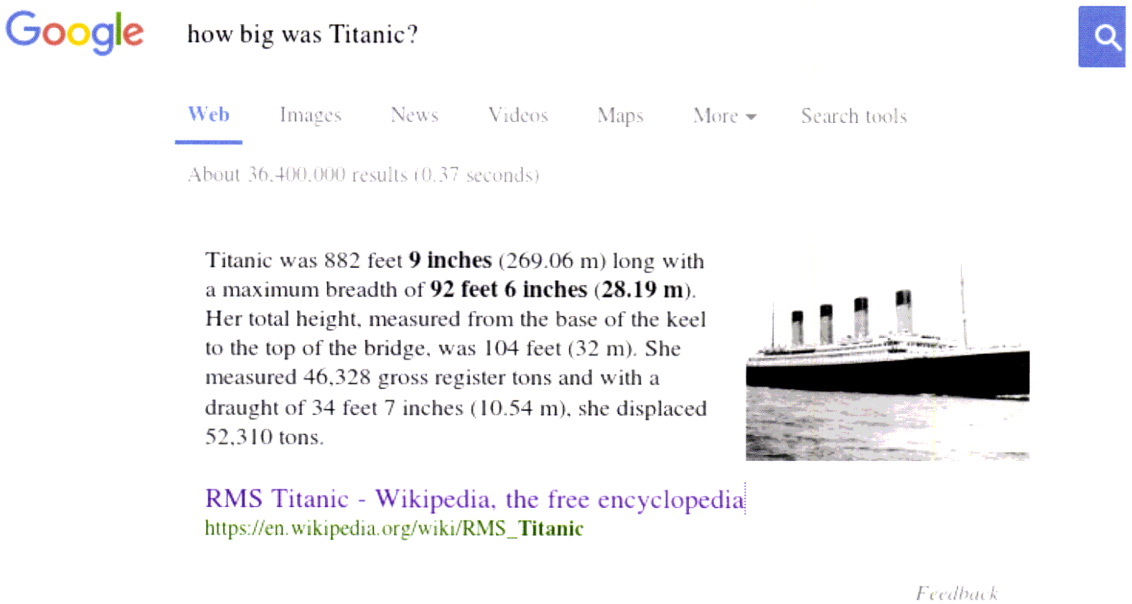
Altavista var hurtig og kunne indexere både efter emne, søgeord og relevans udtrykt som en rank, man kunne bruge til at danne sig et skøn over hvor mange af søgeordene der var fundet bogstaveligt. Ligesom Google brugte Altavista synonym-søgning, så hvis man søgte på *automobiles* fik man også svar med *car* og lignende.

Altavista var hurtig. I 1996 havde den indexet i memory, der var ufattelig stor (4 GB!) Men ejeren (DEC) bag Altavista havde det ikke godt, og Altavista forsvandt (til Yahoo).

Da Google blev lanceret i 1997, blev den hurtigt populær, i 2000 var det den mest brugte søgemaskine. En vejledning i hvordan man skulle formulere sit spørgsmål lød nogenlunde sådan: Find de centrale ord, skriv dem og tryk return. Eller skriv spørgsmålet i almindeligt sprog.

Google havde tidligt en primitiv sprogforståelse, mest på engelsk: de uvigtige ord blev filtreret fra. Men i dag har Google flere måder at "forstå" sætninger:

Eksempelvis: "How big was Titanic?"



De fleste af den slags søgninger bliver gentaget så mange gange, og så gemmer Google i søgning og resultat - og laver måske et klip som her (det svarer ikke til opsætningen i Wik.)

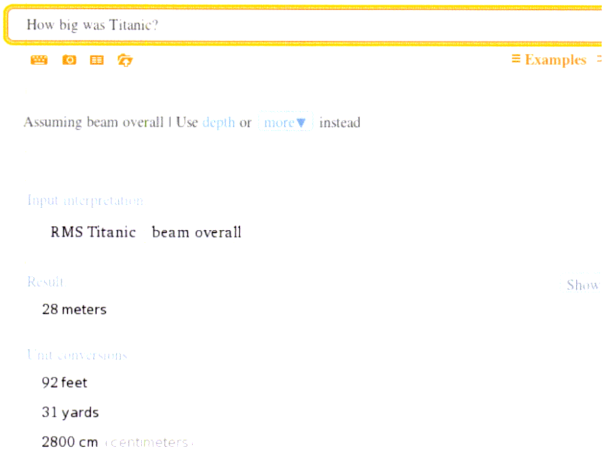
Prøver vi i stedet at søge på //How big was MS Selandia// det første dieseldrevne oceangående skib, så får vi en henvisning til Wikipedia, men ikke en så fin opstilling som den viste.

Mathematica - beregninger og godkendt information

Til forskel fra Google er Wikipedia en fact-base, en encyclopædi, hvis data imidlertid ikke er godkendt af en kvalificeret redaktionsgruppe og som derfor somme tider kan være forkert. Der gøres meget for at højne Wikipedia, men det er en anden diskussion.

Som modsætning til det har firmaet Mathematica, der står bag beregningssoftware (og visualisering af matematiske funktioner mv) sat sig for at lave en "svar-maskine", et website, ***wolframalpha.com***, som giver beregnede svar ud fra godkendte data (curated data). Til det formål har man stiftet et selskab Wolfram Alpha LLC (begrænset ansvar, Limited Liability Company) og nedsat en redaktionskomité som checker facts.

Her er nogle eksempler:



Man ser at udover bredden får man også omregnet bredden. Desuden foreslår Wolfram at man søger på *total capacity.*

Spørger man på *maximum weight of a moose* (elg) får man det ventede svar plus en pæn opstilling med elgens plads i arternes stamtræ og et lille billede.

Spørger man efter en kendt hollandsk maler, Jacob van Ruisdael, får man en ukendt amerikansk person plus et klip fra Wikipedia af hvad navnet Jacob betyder. Det hjælper ikke at søge på fulde navn Jacob Isaaczoon van Ruisdael: *using closest Wolfram Alpha interpretation,* bruger nærmeste Wolfram fortolkning: Jacob van *assuming van is a given name.*

Der er en række skærmbilleder med forslag til hvad man kan spørge om indenfor de forskellige emneområder.

■

DKUUG boggruppe

Bestyrelsen har nedsat en boggruppe til at oversætte bogen "UNIX – the Book" af M Banahan & A Rutter. Gruppen består af Bo Svarre Nielsen og Keld Simonsen fra bestyrelsen samt Kim Storm, Kim Pedersen og Niels Garde fra DIKU. Den bliver ajourført til System V og 4.2 BSD og forventes at udkomme på Teknisk Forlag til sommer.

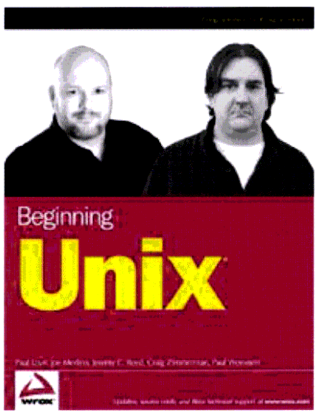
Tilbud om billigt abonnement på PC World

Forlaget Computerworld har tilbudt, at DKUUG's medlemmer kan abonnere på tidsskriftet "PC World" til nedsat pris, nemlig 100 kr pr år istedet for 250 kr. Skriftlig henvendelse herom til DKUUGs kasserer: Mogens Buhelt, Kabbelejevej 27B, 2700 Brønshøj.

DKUUG For 30 år siden:

Behovet for informativ oplysning om hvad man kunne med Unix var presserende - DKUUG var på pletten med en af de bedste Unix-bøger nogensinde.

af Keld Simonsen



Vi - den daværende DKuug bestyrelse - valgte bogen af Banahan & Rutter fordi den havde et passende omfang, og jeg tror der ikke var så mange andre unix-bøger på markedet. Den engelske bog var vist lavet som kursus-bog til Banahans og Rutters undervisning. Den danske bog blev ogå brugt i undervisningen, bl.a. hos superusers og på købmandsskolen.

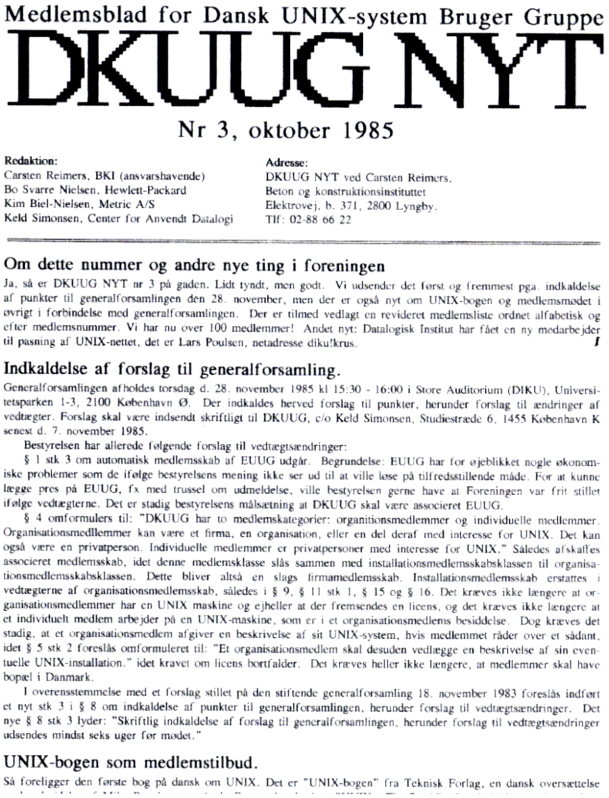
Bogen blev oversat i stedet for at skrive en ny fra bunden, fordi så havde vi ikke besværet med at skulle opfinde hele indholdet, og idéen var at få en god bog hurtigt på markedet - ikke at tjene en masse gysser på den. Jeg regnede en gang min løn ud til 3 kr. i timen.

Oversætterholdet lavede en del om på indholdet, - bl.a. skrev vi nye afsnit om **vi** og skrev både om sys V og BSD. fsck blev introduceret, og jeg mener ikke at den engelske udgave havde noget om mail.

Hilsen

Keld

Boggruppen nedsat - et klip fra DKuug-NYT nr.2



Medlemsblad for Dansk UNIX-system Bruger Gruppe

DKUUG NYT

Nr 3, oktober 1985

Redaktion:
Carsten Reimers, BKI (ansvarshavende)
Bo Svarre Nielsen, Hewlett-Packard
Kim Biel-Nielsen, Metric A/S
Keld Simonsen, Center for Anvendt Datalogi

Adresse:
DKUUG NYT ved Carsten Reimers,
Beton og konstruktionsinstituttet
Elektrovej, b. 371, 2800 Lyngby.
Tlf: 02-88 66 22

Om dette nummer og andre nye ting i foreningen

Ja, så er DKUUG NYT nr 3 på gaden. Lidt tyndt, men godt. Vi udsender det først og fremmest pga. indkaldelse af punkter til generalforsamlingen den 28. november, men der er også nyt om UNIX-bogen og medlemsmødet i øvrigt i forbindelse med generalforsamlingen. Der er tilmed vedlagt en revideret medlemsliste ordnet alfabetisk og efter medlemsnummer. Vi har nu over 100 medlemmer! Andet nyt: Datalogisk Institut har fået en ny medarbejder til pasning af UNIX-nettet, det er Lars Poulsen, netadresse diku!krus.

Indkaldelse af forslag til generalforsamling.

Generalforsamlingen afholdes torsdag d. 28. november 1985 kl 15:30 - 16:00 i Store Auditorium (DIKU), Universitetsparken 1-3, 2100 København Ø. Der indkaldes herved forslag til punkter, herunder forslag til ændringer af vedtægter. Forslag skal være indsendt skriftligt til DKUUG, c/o Keld Simonsen, Studiestræde 6, 1455 København K senest d. 7. november 1985.

Bestyrelsen har allerede følgende forslag til vedtægtsændringer:

§ 1 stk 3 om automatisk medlemsskab af EUUG udgår. Begrundelse: EUUG har for øjeblikket nogle økonomiske problemer som de ifølge bestyrelsens mening ikke ser ud til at ville løse på tilfredsstillende måde. For at kunne lægge pres på EUUG, fx med trussel om udmeldelse, ville bestyrelsen gerne have at Foreningen var frit stillet ifølge vedtægterne. Det er stadig bestyrelsens målsætning at DKUUG skal være associeret EUUG.

§ 4 omformulers til: "DKUUG har to medlemskategorier: organitionsmedlemmer og individuelle medlemmer. Organisationsmedlemmer kan være et firma, en organisation, eller en del deraf med interesse for UNIX. Det kan også være en privatperson. Individuelle medlemmer er privatpersoner med interesse for UNIX." Således afskaffes associeret medlemsskab, idet denne medlemsklasse slås sammen med installationsmedlemsskabsklassen til organisationsmedlemsskabsklassen. Dette bliver altså en slags firmamedlemsskab. Installationsmedlemsskab erstattes i vedtægterne af organisationsmedlemsskab, således i § 9, § 11 stk 1, § 15 og § 16. Det kræves ikke længere at organisationsmedlemmer har en UNIX maskine og ejheller at der fremsendes en licens, og det kræves ikke længere at et individuelt medlem arbejder på en UNIX-maskine, som er i et organisationsmedlems besiddelse. Dog kræves det stadig, at et organisationsmedlem afgiver en beskrivelse af sit UNIX-system, hvis medlemmet råder over et sådant, idet § 5 stk 2 foreslås omformuleret til: "Et organisationsmedlem skal desuden vedlægge en beskrivelse af sin eventuelle UNIX-installation." idet kravet om licens bortfalder. Det kræves heller ikke længere, at medlemmer skal have bopæl i Danmark.

I overensstemmelse med et forslag stillet på den stiftende generalforsamling 18. november 1983 foreslås indført et nyt stk 3 i § 8 om indkaldelse af punkter til generalforsamlingen, herunder forslag til vedtægtsændringer. Det nye § 8 stk 3 lyder: "Skriftlig indkaldelse af forslag til generalforsamlingen, herunder forslag til vedtægtsændringer udsendes mindst seks uger før mødet."

UNIX-bogen som medlemstilbud.

Så foreligger den første bog på dansk om UNIX. Det er "UNIX-bogen" fra Teknisk Forlag, en dansk oversættelse

UNIX-bogen som medlemstilbud.

Så foreligger den første bog på dansk om UNIX. Det er "UNIX-bogen" fra Teknisk Forlag, en dansk oversættelse og bearbejdelse af Mike Banahans og Andy Rutters kendte bog "UNIX - The Book". Bearbejdelsen er lavet af en arbejdsgruppe i DKUUG, nemlig de velkendte DKUUG mafia-medlemmer Keld Simonsen og Bo Svarre Nielsen og d'herrer Kim Storm, Kim R. Pedersen og Niels Garde fra DIKU, også kendt for deres optræden i DKUUG arrangementer.

Bogen er meget fordansket og ført helt up-to-date (som det hedder); den beskriver nemlig både System V og 4.2BSD som den eneste bog på markedet nu. De originale forfattere var da også yderst interesserede i at få den oversat tilbage til engelsk så hurtigt som muligt!

Bogen koster i bogladen 239 kr incl. moms, men tilbydes til DKUUG medlemmer til en pris af kr 175 incl. moms og porto. Bogen bestilles ved indsendelse af beløbet til DKUUGs girokonto 1 37 86 00, DKUUG c/o Mogens Buhelt, Kabbelejevej 27B, 2700 Brønshøj. Hvis det er nødvendigt med en regning, bedes I henvende jer til Mogens enten skriftligt eller tlf. 02-865533, så laver han én.

DKUUG har stået bag oversættelsen og bearbejdelsen af Mike Banahans og Andy Rutters "UNIX - the book", som nu foreligger fra Teknisk Forlag og kan købes gennem foreningen som specielt medlemstilbud.

Foreningen har fået udvirket, at medlemmerne kan tegne et meget billigt abonnement på bladet PC World, hvor

Er statens IT-leverandører dygtigere end Brugtvognssælgere?

Af Donald Axel

Der er blevet råbt alarm allerede i 2009 for det IT-system, som Skat nu lukker ned - Findes der simple principper som kan forudse IT-skandaler?

Et Fælles Inddrivelsessystem, **EFI** blev det døbt. Måske fordi det skulle være effektivt, der blev skåret ned i personalet, da systemet så småt kom til live i 2009, 300 personer mindre - i en milliard"forretning", som har brug for nøje overvågning med hvad "kunderne" laver. SKAT har lukket den automatiske del, hedder det i skrivende stund.

Beskrivelse	I fortsættelse af gårsdagens information om, at SKAT suspenderer al automatisk inddrivelse, vil vi gerne præcisere, at fordringshavers adgang til RIM fortsætter som hidtil. Det vil sige, at overdragelse og ændringer af fordringer fortsat skal ske via system-til-systemløsninger eller via Fordringshaverportalen. Også kommunikationen fra RIM til fordringshaverne følger kendte procedurer og kanaler.
Dags dato	10 sep 2015 13:09
Gyldig til	01 okt 2015 13:09

Det er kun de automatiske dele af systemet, som lukkes ned. Skats restanceinddrivelsesmyndighed (RIM) bruger stadig en portal og dele af systemet.

Man kan ikke undgå at høre om det i nyheder og der er nok at forundres over, brynene løfter sig op i stratosfæren. Men er statens IT-eksperter og politikerne virkelig så uformående og er SKATs IT indsigt virkelig så ringe, at forklaringen skal findes i inkompetente chefer? ***eller*** er det leverandørerne, - der er flere, - som har snydt SKAT som en anden brugtvognsforhandler?

De simple grundregler for IT-ledelse er blevet beskrevet mange gange, en af de mest berømte er Frederick Brooks *The Mythical Man Month,* som er aflivningen af den gængse antagelse af at hvis det går for langsomt, så sætter vi nogle flere på opgaven. Den ledelsesfilosofi eksisterer desværre stadig. Der er en tyrkertro på, at man kan *købe* udvikleres kompetence som om det var tapet i metermål. Desværre har toppen indenfor computervidenskab ikke kunnet mane den misforståelse i jorden.

Tabene

I 2007 var SKAT restancer allerede store, og i 2009 siger direktør i SKAT, Ole Kjær, at EFI er "nødlidende", der er problemer. Dengang var Peter Loft og VK-regeringen med Lars Løkke Rasmussen i spidsen i gang med sammenlægning af SKAT og skatteministeriets departement. Ole Kjærs stilling blev nedlagt, og der blev ikke reageret på hans advarsel.

Restancerne er allerede på det tidspunkt omkring 50 mia.

I oktober 2013 skriver journalist Martin Borre i Politiken, at restancerne er oppe på 77 mia. kr. og at de er vokset med 25 mia. siden 2007. Politiken siger i februar 2015 at den ikke-inddrivelige del af gælden fra 2013 til 2014 er vokset fra 28.7 til 34.4 mia.

Det tab, som staten i øjeblikket har på at inddrivelsen er stoppet, anses af Computerworld at være ca. 300 millioner om måneden!

Systemtype

I 2004, da man påbegyndte arbejdet med EFI, så det ud som en god idé at rationalisere behandlingen af skyldnere på 10-20 gældstyper. Udviklingen af EFI blev sat igang. Målet var at komme igang i 2007. Første gang, budgettet opjusteres, var allerede i 2005, et år efter udviklingen sættes igang. Systemets oprindeligt 10-20 gældstyper stiger i antal, fordi SKAT overtager alle gældstyper fra politi og kommuner mv.

I 2008 fyres IBM fra en udviklingskontrakt; samme år er budgettet steget fra de oprindelige 144 millioner kroner til nu 535 mio! Og SKAT siger, at systemet vil blive forsinket med 24 måneder. I 2009 advarer Ole Kjær, som har støttet rationaliseringen og som de ansatte mener overhører behovet for arbejdsressourcer. Budgettet stiger til 672 millioner.

Henriette Kirunen fra Cepos gør opmærksom på, at man foretog en sammenlægning af eksisterende systemer, og at de tilføjede krav har gjort udviklingen mere kompliceret samtidig med at man afskedigede de grupper, som havde indsigt i området. Med andre ord, **fejl nr.1**: Viden om forretningsgangen blev spoleret inden systemet var specificeret færdigt.

På nuværende tidspunkt har systemet 600 "flueben" (ja/nej spørgsmål, sand/falsk) - for kategorisering af gældstyper. Beskrivelsen af systemet tyder på at det er et såkaldt "ekspert system", baseret på logik-programmering, hvor meningen har været at systemet stiller nogle spørgsmål under oprettelse af en sag og derudfra kategoriserer sagen, gældstypen mv. På daværende tidspunkt, 2004, var logik-programmering stadig i rivende udvikling, og lovede store resultater.

Man forestiller sig at store koncerners sagsbehandling kan støttes af systemer, hvor brugerne kan oprette regler, som så tilpasses af systemet og automatisk bringes til anvendelse. En virksomhed, Intologic, i Sverige, skriver om sit produkt *Med ritningar i MOVIS® bygger vi regelsystem fem gångar snabbare än med programmering, med högre kvalitet!*

Måske er det den slags tanker, lederne af projektet EFI har haft.

Advarslerne

Som nævnt advarer en direktør i SKAT, Ole Kjær, i 2009 om at EFI er nødlidende, og Henriette Kirunen påpeger i P1, at da burde politikernes advarselslamper *blinke big time.* Når man så desuden hører at SKAT og departementet blev sammenlagt, så vil en overordnet betragtning konstatere, at her er der grobund for kaos og inkompetence - som selvfølgelig bør kunne rettes op ved arbejde og kompetencepleje: **fejl nr. 2:** Ingen kompetencepleje, men naive nedskæringer.

For to år siden, d.30 oktober 2013, skrev Martin Borre i Politiken, at ansatte i Skat påpeger, at fejl i EFI systemet fungerer ikke som ventet, og at inddrivelser sker manuelt.

I skrivende stund læser radioens nyhedsoplæser at anbefalingen fra rapporten er at skrotte systemet. Begrundelsen er, at *et af* ***verdens største*** *konsulenthuse mener at systemet er så fuldt af fejl at man ikke kan rette op på det.* Det er trist at journalisten lader sig besnære til at betegne firmaet "største".

Men det er ikke kun ansatte, også Universitetsfolk er begyndt at påpege elementære fejl i specifikation af systemet: For ambitiøst, kort sagt.

INTOLOGIC

Hem | Lösningar | Produkter | Vision | Kontakt | Om oss

Om oss

INTOLOGIC AB

Intologic är ett innovativt, forskningsbaserat, IT-företag med kreativa, välutbildade och entusiastiska medarbetare. Vi utvecklar det marknadsledande visuella verktyget MOVIS® och bedriver konsultverksamhet. Med ritningar i MOVIS® bygger vi regelsystem fem gånger snabbare än med programmering, med högre kvalitet!

Intologic grundades 1998 som ett avknoppningsföretag från Uppsala Universitet.

På bilden: Åke Hansson, Torkel Hjerpe, Torbjörn Josefsson, Fredrik Möllerberg, Kent Andersson.

INTOLOGIC AB i 2013 - nu er firmaet indgået i Emric, specialister i låneadministration

Kan man have tillid til SKAT i dag?

Forleden, fredag d. 11. september, havde P1-Debat inviteret skatteministeren og tidligere skatteministre til at kommentere på spørgsmål om EFI. Ingen af ministrene ønskede at deltage, men finansordfører Jesper Petersen fra Socialdemokratiet (tidligere SF) stillede op. For at kunne se sagen fra en politikers side, og for at komme med forslag om, hvad en politiker skal holde øje med, har jeg aflyttet programmets indledende afsnit og scriptet væsentlige dele.

Deltagere i debatten var udover Jesper Petersen førnævnte Henriette Kirunen og senere Jørgen Rise, der repræsenterer SKAT medarbejdere. Gitte Hansen fra DR var ordstyrer og stillede de indledende spørgsmål.

Det første spørgsmål var om man kunne have tillid til SKAT i dag. *Henriette Kirunen:*

Nej - det har jeg ikke.

Kammeradvokaten har lavet et notat, som ikke er offentliggjort endnu, men som viser, at Skat har lavet overvågning af borgere, som ikke skylder en krone væk, de har lavet overvågning af børn, familiemedlemmer, de har indkrævet gæld, som var forældet, og altså har de udsat, meget ofte iøvrig ressourcesvage borgere for lønindholdelse, og det kan man ikke have tillid til.

Gitte Hansen siger derefter:

Jesper Petersen, goddag, skatteordfører for Socialdemokraterne, hvor stor er **din** tillid til Skats metoder og IT-systemer med den viden vi har i dag?

Jesper Petersen: På nogle punkter er vi jo langt med at have IT-systemer til at fungere, for de fleste borgere, så kører det egentlig udmærket, men når det handler om for eksempel detteher inddrivelsessystem, så har det jo martret skat nu igennem 10 år, hvor man har forsøgt at få det til at fungere, og skatteministeren han måtte sætte det i bero på baggrund af denneher undersøgelse, som Henriette omtaler, som blev sat igang af den tidligere skatteminister, fordi man jo begyndte at blive mistænksom om det overhovedet kom til at fungere og om det efterlevede reglerne, og det gør det jo så ikke! Det er fuldstændig korrekt hvad du siger, Henriette, at det jo viser sig at man jo simpelthen ikke overholder loven ehm, og i al den tid hvor dether system så ikke har fungeret, da har vi jo i fællesskab faktisk mistet en masse penge fordi man må afskrive på gæld, der bliver forældet, fordi det ikke bliver inddrevet. Og det er jo virkelig virkelig, - altså - ordet skat og skandale i samme sætning er ved at være fortærsket, men det er det vi har at gøre med.

GH: Så tillid til Skat, ja eller nej?

JP: På nogle punkter ...

GH: ***Ja eller nej?***

Jesper Petersen: Så enkel er politik ikke, der er al mulig grund til at være meget kritisk overfor hvordan en serie ting fungerer, men jeg vil jo ikke sidde her og sige til alle medarbejdere i Skat at man ikke kan have tillid til hvad de laver.

Gitte Hansen: Men skat og skandale i samme sætning, det tager du ikke afstand fra?

Jesper Petersen: Absolut ikke! og jeg er blandt frontlinie-kritikerne af hvordan mange ting fungerer og mener absolut at der er behov for at vi får rettet op på en serie problemer, og for at vi som borgere kan have tillid til at Skat fungerer effektivt og når folk selv betaler det, de skal, at så sørger Skat for at det gør alle andre også, høj som lav, uanset hvor dygtig en revisor man har.

Hvorfor blev systemet ikke standset før? 1,2 mia i tab på udvikling og drift er trods alt en sjat!

Gitte Hansen: Jesper Petersen, du sad i den forrige SR regering ...

Jesper Petersen: Det ville jeg ønske, jeg gjorde, men jeg var ikke minister,

Gitte Hansen: Du var ikke minister, men du var en del af grundlaget for regeringen,

Jesper Petersen: Ja, det er rigtigt.

Gitte Hansen: ... og nej, du var ikke minister, men du var med på holdet - og ... flere gange i den periode hvor den røde regering sad, da var detteher inddrivelsessystem til diskussion, blandt andet da daværende skatteminister Benny Engelbrecht, din kollega i Socialdemokratiet i maj holdt et samråd hvor I diskuterede blandt andet EFI, eh ... med alle de problemer, det har skabt gennem årene, altså opkrævning af forældet gæld, ulovlig registrering af persondata for danskere og deres ægtefæller og børn, hvorfor gjorde I ikke noget for at stoppe galskaben?

Jesper Petersen: ***Jamen det gjorde man jo faktisk også.*** Vi overtog, da vi kom til dether IT system og udviklingen af det, som var sat i gang under Kristian Jensen, mener jeg, i 2005, - og man fik jo hele tiden at vide at nu kommer det i 2007, nu kommer det i 2009, nu fungerer det i 2013, og nu viser det sig jo nok at man må lægge det helt i graven. Men beskeden dengang var, at det ville komme til at køre, det system. Da der så begynder at blive røster, der siger, at her foregår nok noget, der ikke er lovligt, så bliver der jo ikke sat nye inddrivelser igang de steder hvor man mener at der er tvivl om det faktisk er lovligt, og så begynder man så at undersøge det, der er foregået hidtil, og det er Benny Engelbrechts foranledning at vi har den rapport, Henriette omtaler, som siger, at der *er* foregået ulovligheder, og at man også på andre punkter, udover det, der er ulovligt, simpelthen ikke har gjort det godt nok. Så nu må man nok lægge det helt i graven, tror jeg.

(fortsættes side 18)

Logik programmering med prolog.

Af David Askirk
Kursusinstruktør, SuperUsers a/s

Logik er en stor del af IT. Alle programmeringssprog har en måde at udtrykke logiske udtryk på og bruge dem.

Der er dog et programmeringssprog som virkelig kan bruges til decideret logisk programmering, nemlig Prolog.

Inden vi kaster os over prolog skal vi have styr på de logiske operatorer og begreber.

Vi har en konjunktion, også kendt som AND. Skrives som a ∧ b

a	b	res
0	0	0
0	1	0
1	0	0
1	1	1

Denne er kun sand hvis både a og b er sand (res kolonnen er ***resultat*** af a ∧ b).

En kendt bekendt af konjuktion er disjunktion også kendt som OR. Skrives som a ∨ b.

a	b	res
0	0	0
0	1	1
1	0	1
1	1	1

Denne er sand hvis enten a eller b eller begge er sand.

En der minder meget om OR er XOR. Skrives som a ^ b.

a	b	res
0	0	0
0	1	1
1	0	1
1	1	0

Denne er kun sand, hvis a eller b er sand, men ikke begge.

De forskellige operatorer kan vendes ved at benytte negate eller NOT.

a	res
0	1
1	0

Disse logiske operatorer er nogle som de fleste er bekendt med, og bruger i det daglige.

```
Det regner og det sner.
Det regner eller det sner.
Det regner ikke.
```

De mere avancerede logiske operatorer dækker implikation og bi-implikation.

Implikation

Sandhedstabellen for implikation er som følger: Den skrives som a => b

```
a b res
0 0  1
0 1  1
1 0  0
1 1  1
```

Denne kan godt se lidt mærkelig ud, da et udtryk som:

"I dag regner, derfor er bliver jeg våd" giver mening når man læser det. Derimod et udtryk som "Foden er sort, derfor spiser jeg fisk" ikke giver mening i tale, men giver mening i logik.

Man skal læse det som at implikation er en sandhedsoverførelse. Derfor hvis der er a er sand, skal b være sand, ellers er udtrykket falsk.

Bi-implikation

Den anden der minder om implikation er bi-implikation. Den skrives som a <=> b. Her er sandhedstabellen:

```
a b res
0 0  1
0 1  0
1 0  0
1 1  1
```

En bi-implikation er også sandhedsoverførelse, men begge veje.

Sætningen "I dag er det godt vejr, derfor er jeg glad" kan skrive på en logisk form således:

P: Det er godt vejr i dag

Q: Jeg er glad

Så kan ovenstående udtryk skrives som P => Q.

Nu hvor de logiske operatorer er defineret kan vi gå i gang med prolog. Prolog kan bruges til at vise om netop P => Q gælder.

Prolog kan bruges til at vises om følgende er sandt:

P,Q,R => T
Eller som man vil sige:

(P and Q and R) => T
Dette gør prolog ved at prøve at finde et tilfælde hvor **(P and Q and R)** er sand og **T** er falsk. Hvis prolog kan dette, er udtrykket **(P and Q and R) => T** falsk. Se på implikations pilen for at se at det eneste tilfælde hvor **P => Q** er falsk er netop når **P** er sand, men **Q** er falsk.

Lad os se lidt på prolog syntaks.

Når prolog *(swipl)* starter er der følgende prompt:

```
?-
```

Dette viser at prolog er klar, og kan modtage kommandoer.

Et prolog program kan deles op i to dele. En vidensbank og en regel del.

I vidensbanken kan man putte den viden man har inden programmet kører.

Eksempler på viden kunne være:

```
mac(itemA).
windows(itemC).
windows(itemK).
```

Her defineres tre items, nemlig *itemA*, *itemC* og *itemK*. Dette sættes så til at være en mac, og to windows computere.

Nu kan vi allerede spørge systemet om den viden der er opnået. Gem de ovenstående tre linjer i en fil og load dem ind i prolog interfacet. Dette kan enten gøres via menuen eller ved at consult'e i interfacet.

I dette eksempel spørger vi gennem interfacet: Dette gøres med at skrive:

```
mac(itemA).
```

Prolog svarer true. Altså den item der hedder *itemA* er en mac. Lad os spørge om mere:

```
windows(itemA).
```

Prolog svarer falsk, hvilket er heldigt, da vi har defineret *itemA* til at være en mac.

Lad os definere nogle flere fakta. Gem det i en ny fil:

I logik-verdenen opererer vi med forskellige ting, fx. atom. I denne sammenhæng er en atom et tegn, fx. P, Q eller et helt trejde.

I logik er et atom noget, som enten er sandt eller falsk. Det kan ikke være andet.

Man kan komme fra en almindelig sætning som *i dag regner det* til et atom ved at sige at udtrykket P svarer til *om det regner i dag.*

Prolog benytter sig at formler som er bygget op af atomer. En **functor** *mac* og et **argument** *itemA* skrives mac(itemA) og kan bruges til at symbolisere at itemA er en Apple Mac-computer.

En formel er enten en regel eller et faktum.

I prolog udtrykker operatoren **:-** at hvis højre side er sand, så er venstre side det også:

```
mac(itemA):- os_x(itemA).
```

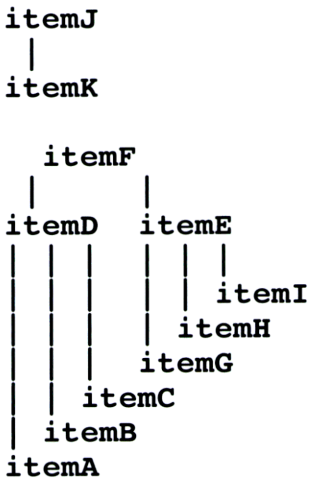
siger at hvis itemA er af kategorien *os_x*, så er det også en computer af kategorien *mac*.

Linjen **mac(itemA).** er et faktum, men det svarer til en regel hvor højre side altid er sand. Linjen **mac(itemA).** svarer til:

```
mac(itemA) :- true.
```

```
mac(itemA).
windows(itemC).
windows(itemK).
linux(itemB).
linux(itemG).
linux(itemH).
linux(itemI).

switch(itemD).
switch(itemE).
switch(itemF).
switch(itemJ).

connected(itemD, itemF).
connected(itemE, itemF).

connected(itemA, itemD).
connected(itemB, itemD).
connected(itemC, itemD).
connected(itemG, itemE).
connected(itemH, itemE).
connected(itemI, itemE).

connected(itemK, itemJ).
```

Den sidste del viser hvilke enheder der er forbundet med hvilke enheder.



Lad os læse det ind i prolog og spørge på det. Skriv eller hent programmet (kan hentes på dkuug.dk eller dnyt ...) Filnavne, som skal loades i prolog, må ikke have bindestreger i navnet.

```
Term: xterm-color, Display: localhost:0.0
sat2:/ #cd /wb5/arb178
sat2:/wb5/arb178 #swipl -s dkuug_prolog-1.pl
Welcome to SWI-Prolog (Multi-threaded, 32 bits, Vers
Copyright (c) 1990-2015 University of Amsterdam, VU
SWI-Prolog comes with ABSOLUTELY NO WARRANTY. This
and you are welcome to redistribute it under certai
Please visit http://www.swi-prolog.org for details.

For help, use ?- help(Topic). or ?- apropos(Word).

?- listing(connected).
connected(itemD, itemF).
connected(itemE, itemF).
connected(itemA, itemD).
connected(itemB, itemD).
connected(itemC, itemD).
connected(itemG, itemE).
connected(itemH, itemE).
connected(itemI, itemE).
connected(itemK, itemJ).

true.

?-
```

Når programfil er indlæst af Prolog ...

Nu kan vi prøve at spørge:

```
?- connected(itemA,itemD).
true.
```

Systemet svarer meget fint og siger at *itemA* er forbundet til *itemD*.

Filen (**dnytprolog1**) skulle forestille en netværks struktur i et firma. Vi vil gerne kunne spørge på om en maskine er forbundet til en switch og om to computere er fundet.

Lad os starte med at løse det med at se om en enhed er forbundet til den øverste switch.

I vores lille verden her, skal vi kunne se at *itemA* er forbundet til *itemF*, hvor *itemF* skal ses som en core switch.

Hvis vi spørger nu:

```
connected(itemA, itemF).
```
svarer Prolog "falsk" tilbage.

Der er nemlig ikke en direkte forbindelse mellem *itemA* og *itemF*. Forbindelse mellem *itemA* og *itemF* går gennem *itemD*.

Derfor skal vi definere en regel der fortæller hvornår noget er forbundet. Lad os lige formulere den med ord:

To enheder er forbundet, hvis der fra den ene enhed kan findes en vej til den anden enhed.

Der er dog en lille krølle på dette. Da vi har defineret enheders forbindelse således:

```
connected(itemA, itemD).
```
bliver der faktisk sagt: Der er en forbindelse fra *itemA* til *itemD*, men der bliver ikke sagt noget om at der er en forbindelse den anden vej.

Dette er dog intet problem, som vi kommer til at se senere.

Lad os se om vi kan finde en vej fra enhed *itemA* til enhed *itemF*.

itemA er forbundet til *itemD*. *itemD* er forbundet til *itemF*, derfor er *itemA* forbundet til *itemF*.

I prolog kan det defineres således:

```
isConnected(X,Y) :- connected(X,Y).
isConnected(X,Y) :- connected(X,Z), isConnected(Z,Y).
```

Der er to tilfælde når vi spørger på om to enheder er forbundet. Enten så er de direkte forbundet, som *itemA* og *itemD*, eller også er de indirekte forbundet som *itemA* og *itemF*.

Det første tilfælde er defineret således:

```
isConnected(X,Y) :- connected(X,Y).
```

Det andet tilfælde er lidt mere kringlet, men er defineret således:

```
isConnected(X,Y) :- connected(X,Z), isConnected(Z,Y).
```

Denne linje skal læses som:

Vi spørger på om der findes en forbindelse fra X til Y. Da der ikke er en direkte forbindelse (ellers var vi stoppet i første tilfælde) må der altså findes en mulighed for at komme fra X til Z, og fra Z til Y. Nu kalder vi så funktionen igen, hvor vi søger efter en forbindelse fra Z til Y. Dette kan vi så blive ved med, til det enten er opfyldt eller ikke er opfyldt, fordi der ikke kan findes en forbindelse mellem X og Y.

Skriv denne regel (de to linjer) ind i toppen af filen og læs den ind i prolog.

(Kan evt. hentes under filnavn dnytprolog2.pl)

Afprøv den med:

```
isConnected(itemA, itemF).
```
og systemet returnerer sand. Tryk på . (punktum) for at få den til at stoppe.

Nu kommer den store styrke ved prolog, for hvis vi i stedet stiller spørgsmålet: Hvilke switche er *itemA* forbundet til?

Nu kan vi prøve alle switche af en af gangen, eller vi kan få prolog til at finde dem for os.

Hvis vi indtaster:

```
isConnected(itemA, Y).
```
og trykker enter, vil prolog finde de tilfælde hvor isConnected er opfyldt, med *X* låst til *itemA* og Y er fri. *Y* er en variabel. (Variable skal skrives med stort begyndelsesbogstav).

Ved at køre dette får vi at vide at *itemA* er forbundet til *itemD* og *itemF*. Tryk på ; (semi-kolon) for at få prolog til at give næste løsning.

Nyt spørgsmål: Er to enheder på samme netværk?

Den næste information vi gerne vil bruge er om to enheder er forbundet på netværket. Vi definerer to enheder til at være forbundet netværksmæssigt, hvis der findes en switch som begge enheder er forbundet til.

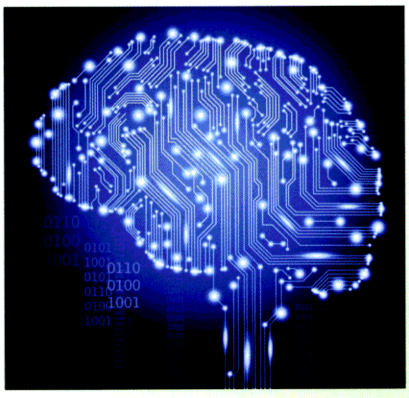
Vi kan benytte vores *isConnected*, samt spørge om noget er en switch ved at bruge se om en enhed er en switch.

```
isInSameNetwork(X,Y) :- isConnected(X,Z), isConnected(Y,Z), switch(Z).
```
Denne regel går ind og siger at to enheder (X og Y) er forbundet hvis der kan finde en switch hvor begge er forbundet til.

Hvis vi spørger på om enhed *itemA* og *itemC* er forbundet for vi at vide at det er de.

Nu kan vi også spørge om hvad *itemA* er forbundet til. Dette gøres med dette kald:

```
isInSameNetwork(itemA, Y).
```



Nu kommer der et resultat der ligner dette:

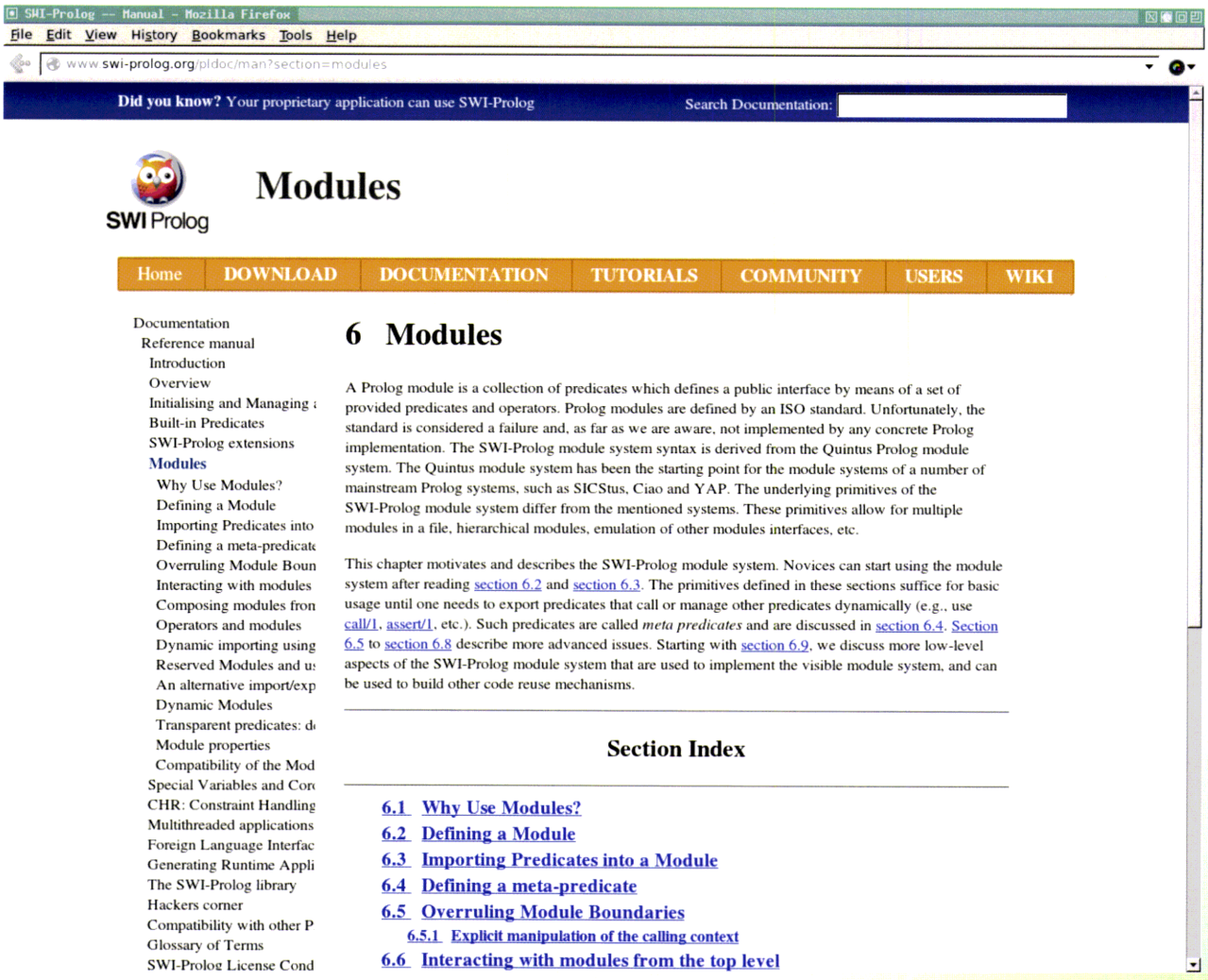
```
Y = itemA ;
Y = itemB ;
Y = itemC ;
Y = itemD ;
Y = itemE ;
Y = itemA ;
Y = itemB ;
Y = itemC ;
Y = itemG ;
Y = itemH ;
Y = itemI ;
false.
```

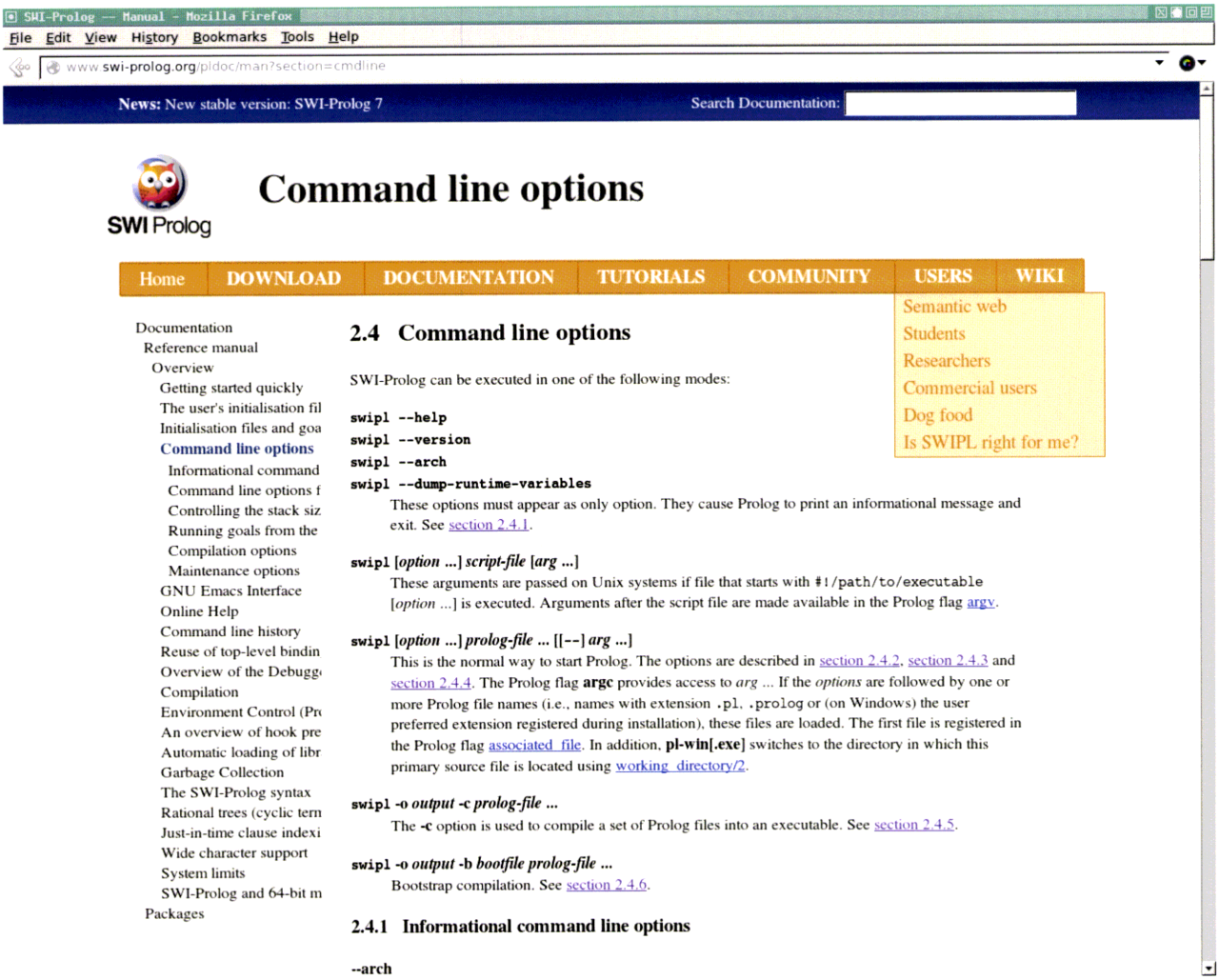
Man kunne undre sig over at nogle enheder optræder flere gange. Dette er fordi at, for at komme fra *itemA* til *itemB*, kan vi nøjes med at gå op til den switch de begge sidder på, men systemet går videre op, og ser at man også kan komme fra *itemA* til *itemB* ved at gå til *itemD*, dernæst så *itemF* og tilbage til *itemD*.

Dette er blot en af de mange brugs muligheder der er for prolog.

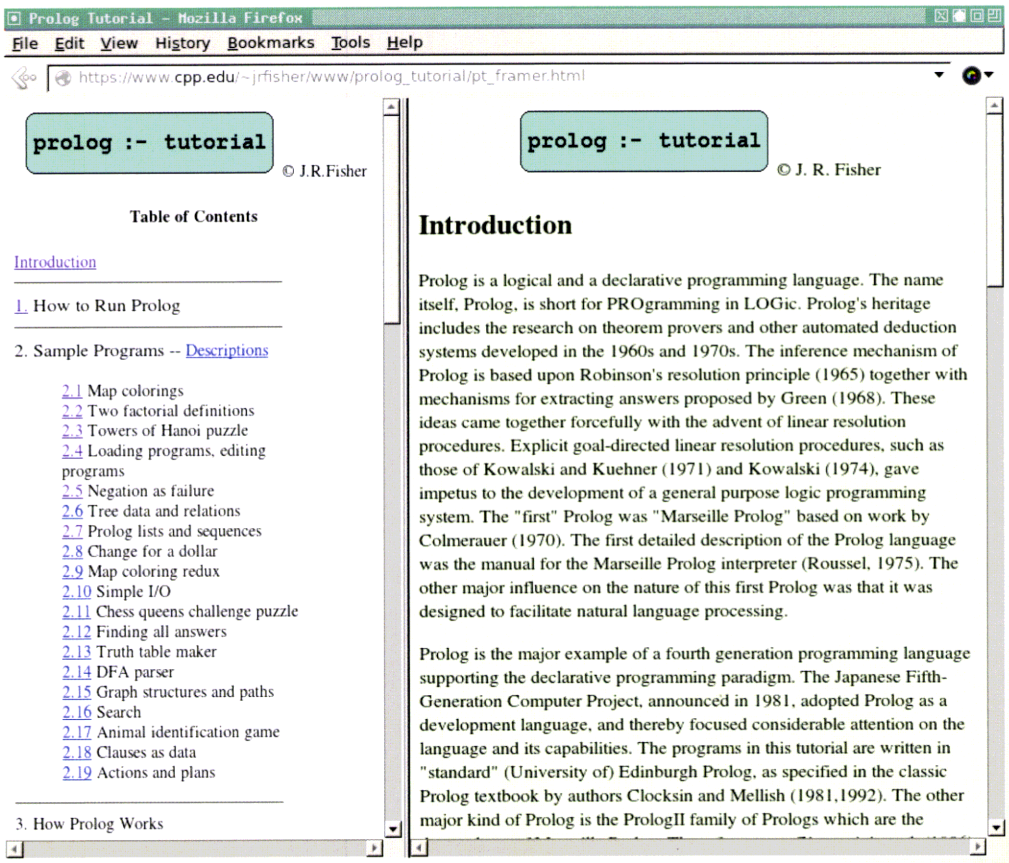
IBM's Watson bruger prolog til at udlede konklusioner.

✸





De vigtigste kommandolinie-funktioner er -s (indlæs et program i kør det i fortolkeren) og -c (indlæs et program og foretag oversættelse til en binær-executable).



Denne lærebog kan anbefales - tak til cpp.edu/~jrfischer

Prolog på Linux

af redaktionen

Rødder

Prolog blev udviklet i begyndelsen af 1970'erne af to forskere, Alain Colmerauer og Philippe Roussel ved universitetet i Marseilles, nogenlunde samtidigt med sproget C, men ud fra helt andre ønsker og tanker: Man ønskede en mere læselig notation til linguistisk analyse, og til at udtrykke logik med såkaldte prædikater ("Per er et menneske", "hest er et navneord") og uddrage konklusioner ("mennesker er dødelige - altså er Per dødelig"). Det kaldes også prædikat-kalkule, på engelsk predicate calculus eller formal logic. Prolog udgør et begrænset sæt af prædikat-kalkule.

Prædikater kaldes også udsagn eller meninger.

Alain Colmerauer fortsatte arbejdet, udviklede videre på sproget, skrev compiler, dannede et firma ***PrologIA*** til varetagelse af sproget, og arbejdede med constraint-logic-programming Der er ikke noget i prolog, som ikke kan udtrykkes med andre sprog, men regelsæt til fx. beskrivelse af naturligt sprog bliver meget omfattende og bliver hurtigt ulæselige i procedurale programmeringssprog som Pascal og C.

Colmerauer arbejdede også med maskinoversættelse, og fik mange hædersbevisninger bla fra Association for Constraint Programming, Research Excellence Award (a4cp.org).

Fordele og ulemper

Kategorier er menneskets måde at analysere og forstå omverdenen. Prædikat kalkule i eksempler til Prolog og andre logiske sprog er ofte forsimplinger, men de bedste af dem afspejler mulighederne når metoden skaleres op. Hvis man fx. tilføjer afstand i eksemplet i artiklen side 8, *Logik-programmering med Prolog,* så vil man kunne lave queries, der undersøger afstand, i mindre netværkssammenhæng er det antal "hop" men hvis det var i en rutefinder, kunne man såmænd også finde korteste rute fra A til E - om det var via B, C eller D.

Der findes flere Open Source implementeringer af Prolog - vi brugte SWI-prolog

SWI står for **Sociaal-Wetenschappelijke Informatica** (*Social-videnskabelig Informatik*, eller *Social Science Informatics*), det tidligere navn for den gruppe ved Amsterdams Universitet hvor programmøren, manden bag SWI-Prolog, Jan Wielemaker, arbejder. Navnet på hans afdeling er siden dengang i 1987, da han begyndte arbejdet med Prolog, ændret til HCS - Human Computer Studies.

SWI-Prolog bruges ofte i undervisning i det semantisk web - en generel måde at klassificiere websider - fx. som varekatalog, prisinformation eller lokalnyhed - på samme måde som aviser er opdelt i sektioner, emneområder og "klummer". Til det formål understøttes RDF, (ressource beskrivelser, opmærkning af indholdstyper, Resource Description Framework).

Der er mange features i SWI prolog:

- Biblioteker, som understøtter interval-kontrol eller avanceret mængdelære (constraint logic programming),
- multi tråd (thread) programmering,
- unit-test,
- GUI, - delvis, i det mindste,
- interface til Java,
- ODBC (opslag i databaser, Open Database Connectitivity), webserver,
- SGML (markup),
- og IDE og debugger osv.

Hvad er der i et symbol?

I logik-programmering bruges symboler for ting, begreber og egenskaber. Symbolerne har ikke anden mening end den, vi mennesker giver. Det svarer nærmest til C-sprogets enumerations, som ofte bruges til at nummere kategorier, fx enum `sports {TENNIS, SQUASH};`

`likes(peter,tennis).` rummer ikke mere information for Prolog fortolkeren end fx. **`m(a,b).`** (Der sættes altid punktum efter Prolog udsagn eller facts; de kaldes også prædikater. Symboler altid med små bogstaver.)

Det, som Prolog giver os, er en automatik som danner kode til besvarelse af et spørgsmål (query) fx.
`likes(peter,What).`

`What` i eksemplet her kaldes en ubundet variabel (signaleres med stort begyndelsesbogstav) og Prolog søger så igennem alle "likes" og finder tennis. Sammenlign med **`m(a,W)`** som vil returnere (svare) b.

`likes(peter,What)` kommer for os mennesker let til at lyde som et spørgsmål **"Hvad kan Peter lide?"** Det er meget praktisk.

Men prolog syntax kan blive utroligt obfuscated, kryptisk, sort magi, når programmerne vokser i størrelse. Man kan måske forestille sig at en ekspert, der bruger det hver dag, vil kunne læse og skrive uden problemer, men det er nok snarere ligesom med APL, (A Programming Language) der var populært for statistikere i 80'erne, og som har sit eget symbolsæt, hvormed man kan skrive formler - efter 14 dage kan end ikke programmøren selv læse formlen eller huske hvorfor han skrev den sådan.

Til mange opgaver vil det derfor være rart, at SWI-Prolog kunne læse en anden syntax, oversætte den til prolog-program og derefter køre den.

SWI kan (og en del andre Prolog) læser og oversætter *Definite Clause Grammar, **DCG,*** en "præcis sætning grammatik". DCG egner sig godt til analyse af naturligt sprog. Det er så simpelt at skrive, at man kalder det ***"syntaktisk sukker"***!

En beskrivelse i DCG af en sætning meget forenklet:

```
sætning --> navneled, udsagnsled.
navneled --> artikel, navneord.
udsagnsled --> udsagnsord, navneled.
artikel --> [en].
navneord --> [kat].
navneord --> [ræv].
udsagnsord --> [æder].
```

Ja det er ikke helt korrekt, der bør skelnes mellem grundled/subjekt og genstand/objekt, men for nu er det nok. SWI-prolog tillader æ,ø,å i symboler.

Der er i dette eksempel ikke taget højde for at ord også kan være "T"-ord (intetkøn) - men det giver forhåbentlig alligevel en fornemmelse af hvad man kan med DCG.

Hvis vi kører ovenstående eksempel igennem SWI compileren, **`(swipl -c saetning.pl)`** får vi prolog kode, som man kan liste med kommandoen **`listing(sætning).`**

```
dax2:arb178 #cat saetning.pl
sætning --> navneled, udsagnsled.
navneled --> artikel, navneord.
udsagnsled --> udsagnsord, navneled.
artikel --> [en].
navneord --> [kat].
navneord --> [ræv].
udsagnsord --> [æder].

dax2:arb178 #swipl -c saetning.pl
% autoloading prolog_codewalk:must_be/2
from /usr/lib/swi-prolog/library/error
% autoloading
prolog_codewalk:portray_clause/1 from
/usr/lib/swi-prolog/library/listing
[...klip]
% Autoloader: loaded 11 files in 2
iterations in 0.169 seconds

dax2:arb178 #./a.out
Welcome to SWI-Prolog (Version 7.2.0)
Copyright (c) 1990-2015 University of
Amsterdam, VU Amsterdam [klip...]

For help, use ?- help(Topic). or ?-
apropos(Word).

?- listing(sætning).
'sætning'(A, C) :-
        navneled(A, B),
        udsagnsled(B, C).

true.

?- listing(navneord).
navneord([kat|A], A).
navneord(['ræv'|A], A).

true.

?- sætning(X,[]).
X = [en, kat, æder, en, kat]
...(fortsætter med permutationer).
?- halt.
dax2:arb178 #
```

Husk punktummet efter kommandoerne (prædikater).

Det er et indbygget prædikat, **expand_term,** som oversætter fra DCG til Prolog.

Bemærk prædikatet for *navneord:* det består af ord-symbolet og lodret streg og stort 'A' i firkantet parentes - det betyder at ordet (kat) og en ubundet variabel A udgør en liste. Prolog er god til at arbejde med lister, men - vil nogen mene - gør det på en mere læselig måde end Lisp.

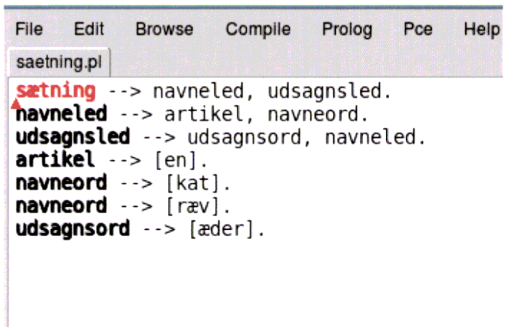
`([kat|A],A)` er en differensliste; det er den måde, Prolog håndterer at vores grammatik består af flere "lag", fra sætning til navneled til (artikel+navneord). Differenslister og avanceret Prolog forklares glimrende på engelsk på et website https://www.cpp.edu/~jrfisher/www/prolog_tutorial/ - eller søg på California Cal Poly Pomona, J.R. Fischer, Prolog.

Prolog kan danne sætninger - men det er sjældent det, man har brug for.

Prolog som fortolker

```
dax2:# swipl -s saetning.pl
```

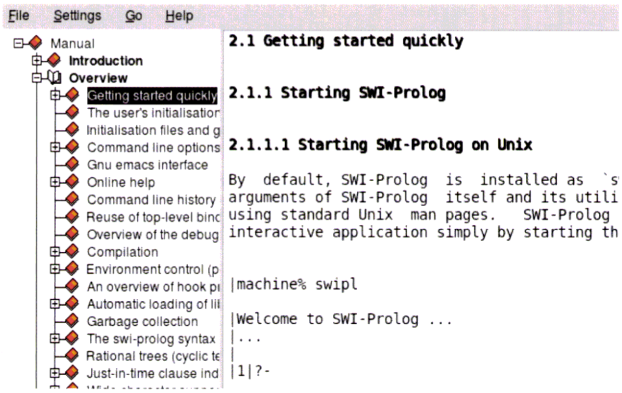
Et program, som er indlæst med -s optionen (script/fortolket program) kan hives over i den indbyggede GUI-editor, og kan editeres, brug kommandoen (eller prædikatet) edit.



Man kan også liste alt, hvad prolog-fortolkeren har indlæst og foretaget sig - brug kommandoen **`listing().`** Der kommer flere skærmfulde output, som bl.a. viser, hvad *expand_term* har indlæst af libraries for at kunne oversætte fra DCG til Prolog.

GUI eller ej: Hvis man undrer sig over den spøjse kombination af vinduer, så kan man have glæde af at læse en side på SWI-prolog.org om IDE eller ikke IDE (Why GUI-Builders are evil).

Prøv også help - enten ved at skrive help(help) til fortolkeren eller ved at vælge help fra editor-menuen:



NLP - Natural Language Processing

Ved analyse af naturligt sprog (eller programmeringssprog) ønsker vi at vide om sætningen overholder grammatiske regler og i givet fald hvilket sætningsmønster, vi står overfor.

Et Prolog program, der skal være nyttigt, bliver derfor hurtigt stort. Et sådant program kan holdes mere læseligt ved at lade Prolog queries udgå fra C kode, benytte databaser til søgning af ord, der vil være sandsynlige for det pågældende emneområde (domæne).

IBM lavede et større system, kaldet Watson, til at spille Jeopardy, og til den opgave valgte IBM's hold af programmører at bruge både C-sproget og Hadoop.

■

Gitte Hansen: Så det du siger, det er, at Socialdemokraterne gjorde, hvad I kunne, for at løse problemet?

Jesper Petersen: Ja det vil jeg sige; at vi håndterede jo det, at der blev givet advarsel om at her var der nok ting, der ikke var lovligt, og så blev der sat stopper for nye sager og sat en undersøgelse igang af det, der var der, og derfor har den nye skatteminister måttet stoppe systemet, og det viser jo at det var rigtigt, det, som Benny Engelbrecht gjorde.

Gitte Hansen: Men hvorfor stoppede ***Socialdemokraterne*** ikke systemet? Hvorfor skal der en Venstre-minister til, før det bliver stoppet?

Jesper Petersen: Ja det bliver han jo tvunget til, nu skal han rydde op efter Kristian Jensen, øh, og det er, øh, en stor opgave efterhånden at være skatteminister med alle de ting, der skal holdes styr på.

Men hele tiden har Finansudvalget jo fået rapporter om *hvad sker det egentlig med systemerne i Skat,* nogen er det gået godt med, det her er det gået rigtig skidt med og der er altid, når man har med sådanne IT projekter at gøre jo en afvejning af, tror man at ved nye ting kan få det til at virke, eller man må sige, nu stopper vi, og det er den konklusion, som vi var ved at nå til og som den nye skatteminister er nået til, og jeg tror ikke at man bare må sætte det midlertidigt i bero, jeg tror simpelthen at man kommer til at starte mere eller mindre på en frisk, og ja, endnu et meget stort problem-sag for Skat.

Gitte Hansen henvender sig derefter til *Henriette Kirunen* med spørgsmålet om det er for let at være bagklog. Ja, det er det jo, siger *Kirunen*, og uddyber, hvorfor en politiker skal være forsigtig, og skal lytte til sine rådgivere. Det giver god mening. Men hvornår skal man træde ind? Det skal man, siger Kirunen, når en direktør siger at der er problemer.

Gitte Hansen vender sig igen til Jesper Petersen og siger: Hvordan kan det være at det fik lov til at køre af sporet så længe?

Jesper Petersen: Jeg har været skeptisk hele tiden, og det vi gør som regering i 2011 det er at overtage forsøget på at få detteher system til at fungere, og når frem til, mod slutningen (af regeringsperioden) at det kan det nok ikke; og [derfor] har [vi] sat gang i den undersøgelse af systemet, som snart barsler og - det er den ny skatteminister, der må træffe den afgørelse, som vi var på vej til, at man nok mere eller mindre må lægge det i graven.

Men som sagt, når det lader sig gøre at løbe i den tid, ja så handler det om at man undervejs har nogle behandlinger i Folketingets Finansudvalg og får indstillinger fra SKAT, som siger, *det kan komme til at fungere,* og det forlader vi os jo på, men [SKAT siger også] at man skal træffe forskellige foranstaltninger, men når det så kommer til stykket, så kan det det ikke.

Og jeg, eh - forventer at der kommer til at skulle køre nogle retssager, fordi der jo også er nogle leverandører, der ikke har kunnet levere det de kan [sic!] så det tab, du taler om, det mener jeg ikke at vi skal acceptere og - og bære, der kan komme til at være et retsligt efterspil, men problemet jo det, som man har været nødt til at afskrive i mellemtiden, og der mener jeg især at den tidligere regering, da man satte gang i denneher sammenlægning af skattedelene i 2005, var for ambitiøse og overså de farer, der var, og derfor må man i mellemtiden afskrive en hel del gæld. Og det er tab for vores fælles kasse.

Hvad er det han siger?

Sammenlagt siger Jesper Petersen ikke noget om systemet og dets opbygning, han siger noget om respekt for SKATs medarbejdere, men ikke noget om respekt for deres udtrykte ønske om flere ressourcer (flere manuelle behandlinger af sager og derfor flere sagsbehandlere). Han gentager, at man som politiker må arbejde ud fra rådgivere (Finansudvalg og indstillinger fra SKAT), men han siger ikke noget om hvorfor man ikke i 2009 reagerede lyttede på at både medarbejdere i SKAT og skattedirektør Ole Kjær gjorde opmærksom på problemer med "ulovlige" opkrævninger.

Politikeren tænker i skyld og erstatning, og flere gange siger, at "systemet må lægges i graven".

Han nævner ikke, at der (efter al sandsynlighed) er store dele af systemets databasedel, som vil være rygrad i et nyt system, og heller ikke om det er en kun er logik delen, eller om det er specifikationen af reglerne, som halter. Det kan man jo godt forstå at han undgår! Det er ikke hans gebet.

Men det havde nu alligevel været rart, hvis en finansordfører havde allieret sig med en IT-planlægger og programmør som kunne fortælle ham mere om, hvilke dele systemet består af, og hvilke dele, der fungerer nogenlunde - og nøjagtigt hvad det er, der bevirker at systemet foretager ulovlige inddrivelser (tilbagehold af lønninger og den slags).

Ud fra hvad man få at vide i pressen, kan man skimte omridset af tre processer i forløbet:

1: Den oprindelige ide med 10-20 forskellige gældstyper styret af et IT-system, på linie med et debitor-bogholderi, der kan foretage kørsler og som (automatisk) udsender rykkere.

2: Nogle uerfarne brugere ser nogle nye ting på de første smartphones og øjner store muligheder: "kan man ikke også medtage fartbøder og lave en ..." hvorefter man prøver at putte flere regler/attributer for gældstyper og regler for behandling af disse gældstyper ind i henholdsvis database og programkode.

3: I 2007 ændres flere forudsætninger, IBM Danmark begynder (gætter jeg) at tvivle på projektet og forlanger kompensation, hver gang der kommer nye tilføjelser, rimeligt nok, men budgettet stiger. I 2009 gætter jeg at IBM Danmark kommer med protester eller stritter imod hvad de ser er problematisk og det accepteres ikke. Spørgsmålet er om det er SKAT (Ole Kjær) eller Peter Loft, Skatteministeriet, som fyrer IBM. IBM er (gætter jeg) lykkerlige over at komme ud; begrundelsen er, at der var problemer med en såkaldt debitormotor (automatik i opkrævninger, styret af regler i databaseform).

4: Men der er også problemer med den "debitormotor" som efterfølgende udvikles af CSC. I artikler fra 2011 ser man at SKAT også overvejer at skille sig af med CSC. Uagtet at der sidder mange dygtige mennesker i CSC rinder det i hu, at man hyrede underbetalte "kodenegre" (undskyld udtrykket!) i CSC i den tro, at man kunne købe sig til kompetence.

5: Iflg. en artikel på Version2 Januar 2015 har efterfølgende KMD og Cap Gemini "prøvet kræfter" med EFI.

Man henter tilbud fra konsulenter (programmørhuse) som skal forsøge at gøre EFI færdigt, med vægt på modularisering i service moduler (SOA).

6: En rapport fra Kammeradvokaten, bestilt af SR-regeringen, er på trapperne og anbefaler at systemet skrottes.

Projektledelse i fire trin

Det er ikke nogen ny ting, at det er bedre at begynde forfra end at lappe på et eksisterende system.

En **første** tommelfingerregel kunne lyde sådan her:

IT-ledelse kræver evnen til i nogen grad at programmere og deltage i beslutninger om design. De to vigtigste "værktøjer" til ledelse er **1:** modularisering (d.v.s. modul-interface definitioner) og **2:** konceptet defensive programming, som sikrer, at en fejl, der ikke er fatal, ikke udløser endnu flere fejl.

Uden modularisering, intet overblik, ingen rettelser er mulige.

Modularisering forudsætter dokumentation og afhængighedstest. Nogle former for test giver ikke mening. Den bedste afprøvning sker i samarbejde med brugerne på et fuld-skala testsystem.

En **anden** tommelfingerregel er, at et projekt-modul ikke må kræve mere end 6 mand i 4 måneder. Efter 4 måneder har man glemt, hvad man begyndte med, og mere end 6 mand er ikke en gruppe, hvor alle kommunikerer med hinanden og holder de andre opdaterede om hvad de ser og finder frem til.

Den regel svarer til Jesper Petersens konstatering af, at projektet i starten udviklede sig til noget, der var alt for ambitiøst.

En **tredie** tommelfingerregel er planlægning i trin eller "**terrasser**", og at man allerede tidligt i det større forløb sørger for at definere mål i form af programmer / moduler (menu-punkter) som fungerer.

En *terrasse* er en release, som fungerer, uagtet at den ikke opfylder alle de endelige specifikationer.

Den **fjerde** tommelfingerregel er, at man kun committer stabil kode, og at hver udvikler skal have mulighed for at afprøve sin kode, sine moduler, i realistiske kørsler, uden besvær.

Det ved vi jo godt alle sammen

Man kan tilføje mange flere tommelfingerregler, fx. levealder og afskrivning: Et modul vil som regel være forældet efter 3-4 år. Maskiner afskrives på 3-4 år. Nye moduler skal kunne kære på et fuld-skala testsystem uden at andre moduler brager ned osv.osv.

Men som en kollega sagde til mig engang: Det er jo noget alle ved.

Det har tydeligvis været en ambition i EFI at man skulle kunne indrive alt fra renovationsafgifter til skattebøder, og det er grundlæggende set ikke en modulær tanke.

Det mærkeligste ved den historie er, at man ikke har set lønudgifter til manuelt arbejde i forhold gevinsten ved at få tilgodehavenderne betalt.

■

Donald Axel har arbejdet med programmering af ERP og systemadministration, kursusinstruktion og planlægning.

Kommandocentralen

Diske på 6TB: Hvis den har bitfejl for hver 40 GB har den 150 bitfejl.

Før eller senere løber man ind i at en disk taber en bit og dermed en blok. Hvis man ikke bruger ret meget af sin disk, så mærker man det næppe. Men hvis man fx. optager videoer, kan man nemt få behov for nogle hundrede gigabyte pr. år, og efterhånden løber det jo op! (se fx. **`video.thecamp.dk`**)

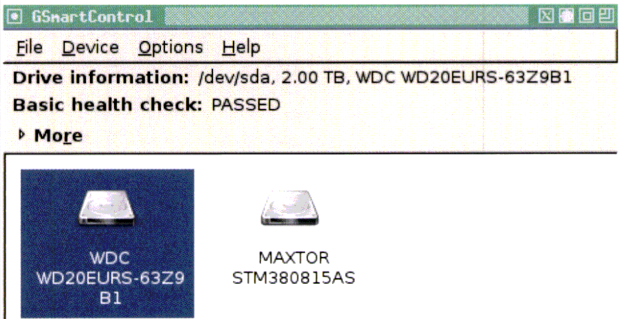
Der er mange programmer til vedligeholdelse af disk og test for badblocks, gparted er et af de bedste. Peg og klik, menuerne husker, hvad kommandoerne hedder og kører dem "for dig".

Men fx. gparted kan ikke checke en disk for bad blocks, læsefejl, hvis den er mounted. Dér kommer S.M.A.R.T. til undsætning, på Linux/Unix i form af softwarepakken smartmontools og gsmartcontrol for grafisk menu.

Fra kommandolinien er det nemt af få vist options:

```
# smartctl --help | less
...
 -a, --all
    Show all SMART information for device
# smartctl -t long /dev/sda
```

Så bliver der kørt en test i baggrunden, uden at man behøver at unmounte disken - belastningen kan mærkes, selvfølgelig.



På en server er det ofte en god ting at man ikke kører GUI programmer, men der er også servere, hvor det ikke har større betydning. Med ssh og X11 forwarding kan man køre grafik-menu til smartctl på en server via netværksforbindelse.

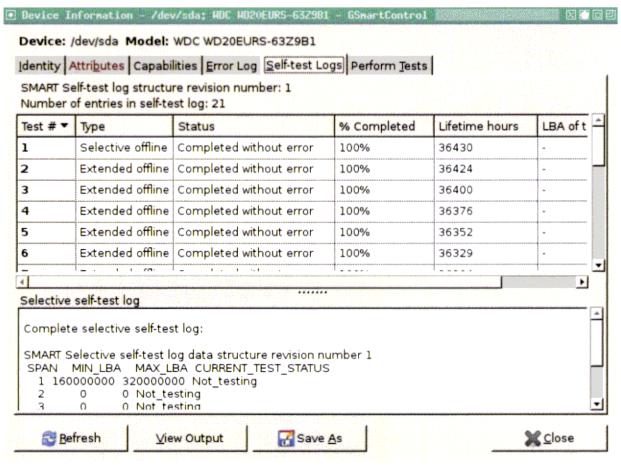
Selve X11-trafikken er krypteret, så det er ikke en sikkerheds-risiko. **`gsmartcontrol`** kan man så bruge til fx. at se på errorloggingen. Men at køre den i baggrunden om natten gøres som bekendt via **`/etc/crontab`**

Hvis man også helst vil tilgå den med et grafisk menuprogram, kan man evt. bruge kde-cron-config til at opsætte en daglig test af disken med smartctl. Men det er næsten nemmere selv at opsætte **`/etc/crontab`**

```
# m h dom mon dow user  command
27  2   *   *   * root  /hjem/bin/daxDisk.sh
```

Som det ses, har vi oprettet et script-dir der kan kopieres til andre systemer (det er "skjult" for pakkesystemet).

Cron systemet starter denne kommando (som account *root* 27 minutter over 2 hver nat.



■

TIL DET DANSKE IT-FOLK

KURSER TESTS CERTIFICERINGER FIRMAKURSER KONSULENTER

Garanti-kurser

Kurserne på denne side har alle garanteret afholdelse.
Vi forstår, at det ikke er rart, når en dato ændres. Derfor har vi indført garanti-kurser med afholdelses-garanti, uanset deltagerantal.

Kurser med garanteret afholdelse

I alt 200+ kurser med garanti for afholdelse i nærmeste fremtid

Filtrer efter søgeord

Teknologi	Kursusnr.	Kursustitel	Dage	Dato	Sted	Garanti
	LX-102 Tilmeld	Linux Administration and Networking	4 d	14. september 2015	Hillerød	✔
	SU-902 Tilmeld	Web App Programmering Datahåndtering	3 d	14. september 2015	Hillerød	✔
	BX-102 Tilmeld	Linux LPIC-1 102 Bootcamp Basic Level Administration	1 d	18. september 2015	Hillerød	✔
	SU-204 Tilmeld	C++ Programmering Videregående	5 d	21. september 2015	Hillerød	✔
	CI-099 Tilmeld	Interconnecting Cisco Network Devices (CCENT/ICND1) v2.0	5 d	21. september 2015	Hillerød	✔
	SU-101 Tilmeld	UNIX og Linux Videregående	5 d	21. september 2015	Hillerød	✔
	SU-241 Tilmeld	SQL Programmering Videregående	3 d	21. september 2015	Hillerød	✔
	SU-093 Tilmeld	jQuery - Det samlede client web-udviklingsforløb	3 d	21. september 2015	Hillerød	✔
	AP-101 Tilmeld	OS X Yosemite Support Essentials	4 d	22. september 2015	Aarhus	✔
	SU-240 Tilmeld	SQL Programmering Grundkursus	3 d	23. september 2015	Aarhus	✔
	SU-205 Tilmeld	UML (Unified Modeling Language)	2 d	24. september 2015	Hillerød	✔
	LX-902 Tilmeld	Android Programmering Datahåndtering	3 d	28. september 2015	Hillerød	✔
	SU-097 Tilmeld	Building Real Time Web Apps with AngularJS	3 d	28. september 2015	Hillerød	✔
	LX-901 Tilmeld	Android Programmering Grundkursus (inkl. Android Tablet)	2 d	01. oktober 2015	Hillerød	✔
	CI-100 Tilmeld	Interconnecting Cisco Network Devices (ICND2) v2.0	5 d	05. oktober 2015	Hillerød	✔
	CI-101 Tilmeld	CCNP Switch	5 d	05. oktober 2015	Hillerød	✔
	LX-201 Tilmeld	Linux Administration Advanced	3 d	06. oktober 2015	Hillerød	✔
	BX-201 Tilmeld	Linux LPIC-2 201 Bootcamp Advanced Level Administration	1 d	09. oktober 2015	Hillerød	✔
	SU-201 Tilmeld	C Programmering Videregående	3 d	12. oktober 2015	Hillerød	✔
	SU-239 Tilmeld	Introduktion til Databaser	3 d	12. oktober 2015	Hillerød	✔
	SU-050 Tilmeld	Netværk Grundkursus	1 d	12. oktober 2015	Hillerød	✔
	SU-901 Tilmeld	Web App Programmering Grundkursus (inkl. Mobile Device)	2 d	12. oktober 2015	Hillerød	✔
	SU-056 Tilmeld	TCP/IP InternetWorking	4 d	13. oktober 2015	Hillerød	✔
	SU-240 Tilmeld	SQL Programmering Grundkursus	3 d	14. oktober 2015	Hillerød	✔
	SU-057 Tilmeld	TCP/IP Videregående	2 d	14. oktober 2015	Aarhus	✔
	SU-095 Tilmeld	Node.js - Det samlede server-side web-udviklingsforløb (inkl. Raspberry Pi)	3 d	14. oktober 2015	Hillerød	✔

... Fortsættes i næste kolonne

I alt 200+ kurser med garanti for afholdelse i nærmeste fremtid

Filtrer efter søgeord

Teknologi	Kursusnr.	Kursustitel	Dage	Dato	Sted	Garanti
	LX-904 Tilmeld	Android Programmering Brugergrænseflade	3 d	19. oktober 2015	Hillerød	✔
	CI-101 Tilmeld	CCNP Switch	5 d	19. oktober 2015	Aarhus	✔
	LX-101 Tilmeld	Linux Install and Use	4 d	19. oktober 2015	Hillerød	✔
	SU-280 Tilmeld	MySQL Programmering	3 d	19. oktober 2015	Hillerød	✔
	SU-159 Tilmeld	Git Versionsstyring	2 d	22. oktober 2015	Hillerød	✔
	BX-101 Tilmeld	Linux LPIC-1 101 Bootcamp Basic Level Administration	1 d	23. oktober 2015	Hillerød	✔
	SU-199 Tilmeld	Introduktion til Programmering	2 d	26. oktober 2015	Hillerød	✔
	SU-211 Tilmeld	Java Programmering Videregående	5 d	26. oktober 2015	Hillerød	✔
	SU-225 Tilmeld	Python Programmering	3 d	26. oktober 2015	Hillerød	✔
	SU-242 Tilmeld	MS Transact SQL (T-SQL) Programmering	3 d	26. oktober 2015	Hillerød	✔
	SU-242 Tilmeld	MS Transact SQL (T-SQL) Programmering	3 d	26. oktober 2015	Aarhus	✔
	LX-202 Tilmeld	Linux Networking Advanced	3 d	27. oktober 2015	Hillerød	✔
	SU-202 Tilmeld	Objektorienteret Grundkursus for C++/C#/Obj-C og Java	2 d	28. oktober 2015	Hillerød	✔
	SU-904 Tilmeld	Web App Programmering af brugergrænseflader - Foundation & Bootstrap	3 d	28. oktober 2015	Hillerød	✔
	BX-202 Tilmeld	Linux LPIC-2 202 Bootcamp Advanced Level Administration	1 d	30. oktober 2015	Hillerød	✔
	CI-103 Tilmeld	CCNP Route	5 d	02. november 2015	Hillerød	✔
	SU-210 Tilmeld	Java Programmering Grundkursus	5 d	02. november 2015	Hillerød	✔
	LX-102 Tilmeld	Linux Administration and Networking	4 d	02. november 2015	Hillerød	✔
	SU-260 Tilmeld	Software test - Det samlede overblik	1 d	02. november 2015	Hillerød	✔
	SU-263 Tilmeld	Software test - C# & Visual Studio - Metoder og værktøjer	2 d	03. november 2015	Hillerød	✔
	BX-102 Tilmeld	Linux LPIC-1 102 Bootcamp Basic Level Administration	1 d	06. november 2015	Hillerød	✔
	LX-905 Tilmeld	Android Programmering Sensors/Services	3 d	09. november 2015	Hillerød	✔
	AP-201 Tilmeld	OS X Yosemite Server Essentials	4 d	10. november 2015	Aarhus	✔
	CI-099 Tilmeld	Interconnecting Cisco Network Devices (CCENT/ICND1) v2.0	5 d	16. november 2015	Hillerød	✔
	CI-103 Tilmeld	CCNP Route	5 d	16. november 2015	Aarhus	✔
	SU-100 Tilmeld	UNIX Grundkursus	5 d	16. november 2015	Hillerød	✔
	LX-100 Tilmeld	Linux Grundkursus	5 d	16. november 2015	Hillerød	✔
	SU-905 Tilmeld	Web App Programmering Videregående - Hardware adgang med PhoneGap (inkl. tablet)	3 d	16. november 2015	Hillerød	✔

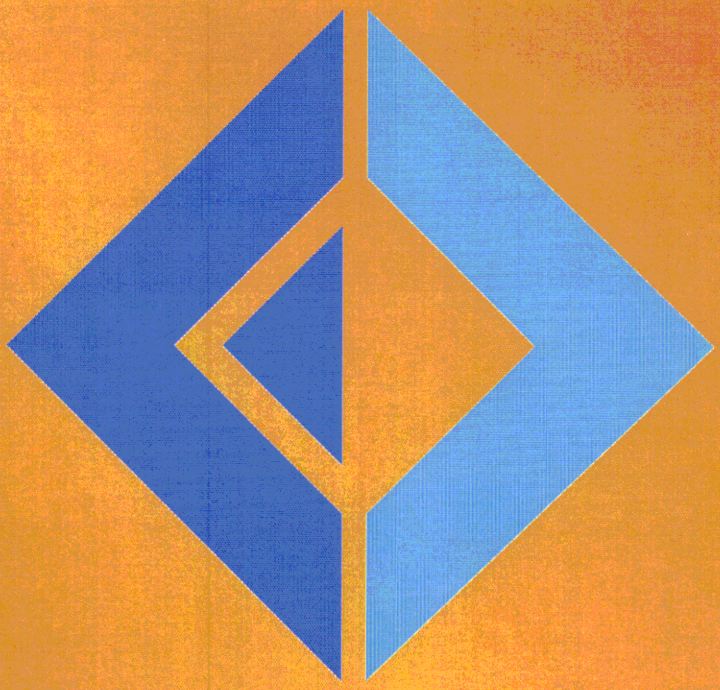
... Se flere kurser på website:

www.superusers.dk

DKuug nr. 176

NYT

November-December 2014



Dansk Forum for Åbne Systemer

DKUUG - Unix Brugere (Linux/BSD) system administratorer

Community for IT-specialister og IT-interesserede.

DKUUG

Kære Læser

Begreberne flyver omkring -Viden fordi det er sjovt - men det er også nyttigt

Under arbejdet med at finde dokumentation for, hvad der er de vigtigste egenskaber ved Mono og .NET frameworket, blev det klart at det var nødvendigt med kendskab til CPU-arkitektur og maskin-instruktioners opbygning for bare at forstå sætningerne i teksten. Her i redaktionen ville vi gerne have givet dig, kære læser, endnu mere information på en let overskuelig måde om virtuelle maskiner, et begreb, der ofte nævnes i forbindelse med .NET programmer (apps) som en slags trylleformular for, hvordan man kan gøre noget bedre.

I dette tilfælde er "bedre" en omskrivning af "så det fungerer ordentligt". Virtuelle maskiner er en måde at lave ét program én gang og bruge mange gange: et program, som kender den hardware, det kører på, og lader alle andre programmer udnytte denne viden.

Der er forskellige måder at bruge det princip - man kan udnytte det til at lave virtuelle hosts, typisk en webserver i en serverfarm hos et hosting-firma, som kører på samme hardware som en stribe andre webservere.

En anden Virtuel Maskine (VM) kendes fra Java. SUN, som "opfandt" Java, havde tænkt sig engang at komme med en silicium-implementation af JavaVM - men det blev vist aldrig til noget. Af flere forskelllige grunde er Intels CPUer bedre og billigere end de fleste alternativer - nåja, bortset fra AMD, og Intel har jo også overtaget ARM processoren, som ikke ligefrem var et Intel design ... sådan er det så meget.

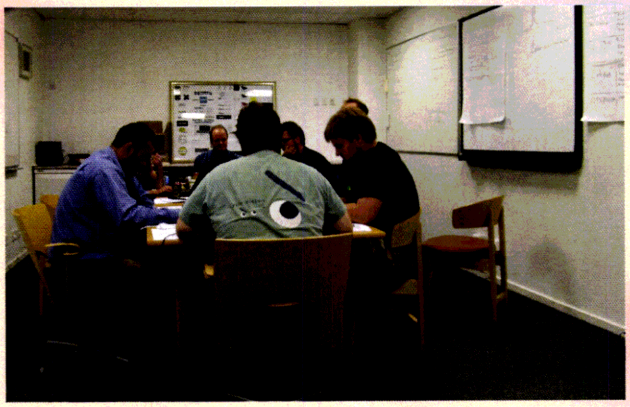
I et kommende nummer vil DNyt kaste sig over fænomenet Virtuelle Maskiner, men også vise de alternativer, som har eksisteret på Unix siden kommandoen ***chroot*** blev "opfundet".

Problemet er, at alle IT systemer i dag er mere komplicerede end noget som helst menneskeheden tidligere har set.

Poul Henning Kamp, som vi ynder at citere, nævner som en kendsgerning de fleste ikke bryder sig om at tænke på, at vi har expanderet software-komplexiteten i en grad, så vi ikke kan overskue konsekvenserne. Kamp siger (citeret efter hukommelsen): Den lille mobil-telefon, som vi har i lommen - selv de mindste og billigste - har samme komplexitet som 12 hangarskibe. Udbredelsen af iPhones og iPads er ikke nogen garanti for, at vi er et foregansland i IT-henseende, siger han også. Hør mere på Radio24syv.dk - man kan downloade programmet Aflyttet uge 44, gå fx. 36 minutter ind og hør Poul Henning Kamp fortælle om CSC og komplexitet.

Allerede Joseph Weizenbaum gjorde opmærksom på de ofte oversete konsekvenser af computersystemers stigende kompleksitet. Hans mest slående eksempel (fremsat i bogen *Computer Power And Human Reason*, Cambridge 1975) går på at de mange bankers børshandels-systemer er en tidsindstillet bombe under økonomien - fordi, som han siger, de udgør en automatisk deterministisk (men uoverskuelig) maskine, som ved et kursfald vil få alle til at sælge og derved udløser endnu mere kursfald.

- Altså, det samme, som udløste børskrak i 1929 og igen i 2008, som dog udløstes af helt andre årsager end børskrakket i 1929. Noget har man lært: IT systemerne lukker ned, hvis kursfald er for voldsomme og uforudsigelige, men afhængighederne fra den ene globale spiller til de andre har vi ikke fået under kontrol.



Der arbejdes i vores kontor

Dette nummer ...

F# - F sharp - har fået en fremtrædende plads i dette nummer, sammen med baggrundsstof om Mono - Open Source impelementering af .NET - selv om emnerne ikke er nyheder. Det nærmeste, vi kan komme, er at det er en langsom revolution, som foregår mens vi ser på det.

Begrundelsen for at vælge dette emne er, at funktions-orienteret programmering, funktions-programmering (FP) for kortheds skyld, er en sag som slår mange professionelle med forundring og ubehag, det er noget underligt noget, som man er lidt valen overfor. Også selv om man godt ved, at dette at lære nye programmeringssprog er gavnligt, omend man ikke bruger det. Lidt ligesom at det er en fordel at kende lidt til Latin. Vi har en masse latinske låneord. Men Paul Graham har en lidt anden og mere interessant indfaldsvinkel på funktionel-stil. I en efterhånden 13-14 år gammel artikel, som man kan finde på nettet. Hans artikel handler ikke om F#, som ikke fandtes på det tidspunkt, men om Lisp, som også er et funktions-sprog (og som vi har omtalt i et tidligere nummer).

Der er ingen, der taler latin, skriver Paul Graham, men der er nogen, der "taler" Lisp. Hvis det kan give dig en konkurrence-fordel på markedet, så brug det - ellers lad det ligge.

Graham startede sammen med en ven firmaet Viaweb, som tilbød udvikling og hosting af webshops. Software er en meget konkurrencebetonet branche, skriver han, en branche, som ofte ender med at den bedste har en slags monopol. Når man starter et firma op, mærker man dette meget tydeligt, det tenderer mod alt eller intet.

Så drastisk vil vi nu ikke formulere begrundelserne for at kende flere sprog. I en helt anden boldgade fandt vi flg. citat om programmering: Hvad der adskiller Funktionel Programmering (FP) fra andre sprog er den fiksering på variable, som ikke kan variere! Jeg har aldrig skrevet et program, som ikke indeholdt "immutable" (uforanderlige) objekter, men vi kalder dem altid bare ***konstanter. [...]*** Jeg kan ikke forestille mig at skrive noget komplekst uden variable og konstanter. Men at skrive et enormt antal 3-liniers funktioner for at undgå en simpel variabel er ikke et fremskridt, skriver David Clarkd, BSc, RCC consulting.

Det skal dog understreges, at David Clarkd finder mange nyttige ideer i funktionel programmering, ligesom det skal understreges, at Paul Graham gik fra Lisp til C, da ordredelen i web-shop systemet skulle udvides.

Vi håber at læserne får en god oplevelse med bladets eksempler på F# programmering

Donald Axel

DKuug-NYT er medlemsblad for DKuug, foreningen for Åbne Systemer og Internet
Nr. 176 - Novmber 2014

Udgiver:
DKUUG
Fruebjergvej 3
2100 København Ø
Tlf. 39 17 99 44
email: blad@dkuug.dk
Redaktion:
Donald Axel (ansvarshavende)

Forsidecredits:
redaktionen, logo fra fsharp.org

Design og layout:
DKUUG/Donald Axel med LibreOffice
Annoncer:
pr@dkuug.dk
Tryk:
Lasertryk i Aarhus
Oplag:
400 eksemplarer

Deadline for nr. 177: ons. 14. januar 2015

Medlem af Dansk Fagpresse
DKUUG-Nyt
ISSN-1395-1440

Vores møder og **foredrag** holdes - med mindre andet udtrykkeligt angives - på vores adresse:

DKUUG
SYMBION
Fruebjergvej 3
2100 København Ø

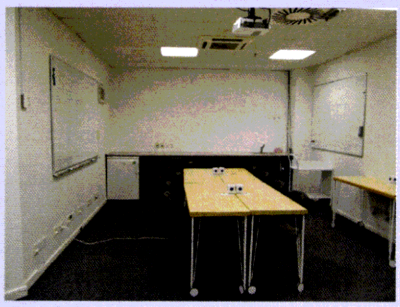
Hvis man kommer lidt før, er der tid til en snak på kontoret. DKUUG bor i en virksomhedsfarm, Symbion, hvor der er åbne døre indtil kl.18 eller 19 (afhængig af mødetidspunkt). Efter den tid har vi på foredragsaftener en vagt ved døren.

INDHOLD:

Arrangementer:

Onsdag d. 26. November: Generalforsamling.

Andre arrangementer vil blive annonceret på web og via mail.



Gør-det-selv foredrag:

Vores kontor i Symbion giver mulighed for hands-on workshops, både dag og aften. Skriv til pr@dkuug.dk eller til bestyrelsen i DKUUG, bestyr@dkuug.dk, og hør om lokalet er ledigt den dag du vil arrangere et møde. Der er hurtig internetforbindelse, både wired og wireless. Er der større tilmelding, kan vi leje mødelokaler i Symbions mødested. Spørg os!

Designfilosofi bag *Mono* - en platform for applikationer

Hvad ønsker en applikationsprogrammør? og hvordan prøver Mono på at gøre det lettere at udvikle applikationer?

Af Donald Axel

Det erklærede mål bag *Mono* er at være en platform (og en udviklingsplatform, *monodevelop*) som gør det lettere og hurtigere at udvikle en applikation. Men hvilket programmeringssprog har ikke det mål? Det særlige ved Mono er, at de resulterende programmer skal kunne udnytte Internettet så let som muligt - plus et par ting mere, såsom at kunne køre på flere typer hardware og operativ-systemer, bruge touchscreens mv.

Miguel de Icaza sagde ved et foredrag i 2010 om den fortsatte udvikling af Mono *(Youtube video Mono Edge, 26 min.inde)* :

> *Innovation er at få markedsført idéer. Man får masser af idéer når man tager brusebad eller går på arbejde. Men innovation er at få det til at ske, få det ud til forbrugerne.*
>
> *Det, som vi ønskede at opnå med Mono for mange år siden, var at gøre programmører mere produktive. Hvordan kunne vi hjælpe programmører med at komme fra idé til marked, - til et produkt, der kan markedsføres? Jeg ved ikke om vi har svaret på det, men jeg vil fortælle jer hvad vores intentioner er, fortælle om basics og hvordan vi kan hjælpe jer med at komme dertil.*

Det har Icaza gentaget ved alle senere konferencer. Han uddyber: At lave et produkt består i at tage et program, rette fejlene og skrive dokumentationen, hvilket som regel tager 3 gange så lang tid som at skrive programmet.

[Ingen blandt publikum protesterer ... Her på redaktionen var der en protest; der findes udviklingsmiljøer, som arbejder i omvendt rækkefølge, først dokumentation, så kode!]

Icaza fortsætter:

> *Hvis man laver en fremragende feature, men ikke skriver dokumentation, kan man lige så godt lade være med at skrive koden, for ingen vil opdage, at den feature findes.*
>
> *For at lave et godt produkt skal man følge konventioner, dokumentere, arbejde sammen, bruge aftalt API (systemkald og kald til hjælpefunktioner, Application Programming Interface).*

Det er også gammelkendte sandheder: standard-libraries - C++ libraries er udviklet og justeret gennem de sidste 20 år.

Der findes andre applikations-miljøer, som har gjort det let at udvikle og dokumentere API'er, moduler, afhængigheder. C, C++ og Java kan køre på mange devices.

Der har været multiplatform miljøer siden 1978, da University of California, San Diego (UCSD) Institute for Information Systems udviklede UCSD Pascal for at de studerende skulle kunne arbejde i et kendt miljø på de mange forskellige typer PC'er, som den gang eksisterede, såvel som på universitetes minicomputer. Det system blev kendt som UCSD p-System.

Men så må der i det mindste være noget nyt ved Mono?

Nu vil vi have den tekniske forklaring, hvis der er en, og ikke blot stille os tilfredse med at Mono svarer til .NET for Linux og Android!

Factbox om Mono

Mono er et Open Source projekt, som styres af Xamarin, tidligere af Novell, og oprindelig af Ximian, gående ud på at skabe ECMA standard-compliant .NET framework-kompatibelt sæt af udviklingsværktøjer, med blandt andet en C# compiler, og en Common Language Runtime, **CLR**, en virtuel maskine, som er specielt designet til at afvikle applikationer (Se nærmere i factbox side 17).

Men hvad går Mono ud på?

Men Mono har ambitioner om at være mere end .NET-for-Linux/Android - det er også for iPhone med Xamarin proprietary library.

Mono kører i dag på Android, BSD, Windows, Solaris, og endda game-konsoller som Playstation 3, Wii, og Xbox-360.

Mono-logoet er en stiliseret abe - mono er spansk for abe.

Svaret kan koges ned til ét ord, er: CIL. (-er det ikke et ord? Nå så tre ord da!) ***Common Intermediate Language.*** Et intermediært sprog er et mellemled mellem programmørens sprog og de maskininstruktioner, som CPU'en ender med at læse, afkode og udføre.

Vi kommer ikke uden om en vis nødvendig grundviden her, men et billede er bedre end mange ord: Et lille F# program (ikke særlig nyttigt) oversættes, først til .exe fil, så kigger vi på den og derefter genererer vi x86 assembler ud fra .exe filen med ***mono***. Kommando-sekvensen

```
// Program som skal give genkendelig AOT code (Ahead Of Time compilation).
// Parenteser om arg til printf-format string er obligatorisk.
let myfirstval = 26
let mysecondval = 7
let divider myv1 myv2 =
    printfn "Division med heltal giver heltal, 26/7: %d" (myv1/myv2)

divider myfirstval mysecondval
```

Her ses et minimalt F# program; den sidste linie sætter programmet igang ved at kalde funktionen "divider"

⬇

```
fsharpc daxdivide.fs -o daxdivide.exe
monodis daxdivide.exe > daxdivide.cil
```

```
  // method line 4
  .method public static
        default void main@ ()  cil managed
  {
    // Method begins at RVA 0x2084
    .entrypoint
    // Code size 9 (0x9)                    0x1a = 16 + 10 => 26
    .maxstack 8
    IL_0000:  ldc.i4.s 0x1a
    IL_0002:  ldc.i4.7
    IL_0003:  call void class Daxdivide::divider(int32, int32)
    IL_0008:  ret
  } // end of method $Daxdivide::main@

} // end of class <StartupCode$daxdivide>.$Daxdivide
}
```

Common Intermediate Language - kun den sidste linie af programmet, `divider myfirstval mysecondval` ***som omsættes til et funktionskald, call void class Daxdivide::divider ...***

`ldc.i4` betyder load en 4 bytes integer (parametret til vores kald af funktionen) med værdien 0x1a (hexadecimal for 26) - det er *myfirstval* som vi satte til 26, den er her blevet *optimeret* til en konstant.

⬇ ➡

Når man i *mono* miljøet laver en programfil, er navnet på sourcefilen en del af identifikationen af objekterne i filen. man skal huske at alle mulige programmeringssprog kan bidrage til et større projekt. Hvis ikke man brugte filnavnet, kunne man risikere nameclash - at en identifier kom til at hedde det samme som en anden variabel (eller F# funktion mv.) og derved ændrede programmet ved at skygge for den oprindelige ting med det navn. Derfor ses i disassembleringen (CIL-listningen) af divider-programmet, at ordet "Daxdivide" har sneget sig ind. Det var filnavnet på det lille demonstrationsprogram og har ikke nogen dybere mening, bortset fra at det er nemt at finde egne programmer, hvis man prepender dem med sine initialer.

Næste trin er at generere native code, assembler-instruktioner til den CPU, som programmet kører på. Programmet *mono* (som egentlig er et symbolsk link, et "shortcut" om man vil, til ***mono-sgen***) oversætter **A**head **O**f **T**ime, **AOT** (i forvejen).

Det er en af de særlige fordele ved Mono at den ***i forvejen*** kan generere native code, maskinkode til den maskine, man ønsker at køre på. Det er ikke en måde at deploye til markedet, eftersom der er så mange maskin-forskelle, man lokalt er nødt til at tage hensyn til. Optimering af koden spørger den aktuelle CPU, hvilke instruktionsgrupper, den har, for at sikre sig at kunne optimere mest muligt.

```
$ mono --aot=writesymbols,\
outfile=daxdivide.exe.so \
     --optimize=sse2 daxdivide.exe
...
$ file daxdivide.exe.so
daxdivide.exe.so: ELF 32-bit LSB shared
object, Intel 80386, version 1 (SYSV),
dynamically linked, not stripped
$ objdump -d daxdivide.exe.so >\
daxdivide.exe.so.dis
```

⬇

```
add    $0x23b4,%ebx
sub    $0x8,%esp
push   $0x7
push   $0x1a
call   119e <plt_Daxdivide_divider_int_int>
add    $0x10,%esp
lea    -0x4(%ebp),%esp
```

Et klip fra en meget længere disassembly-fil: Kaldet til divider-funktionen; assembler-kommandoen sub $0x8 læses* subtraher værdien 8 fra stackpointeren *- derved skabes plads til to 4-byte variable; register-navnet esp (for extended stackpointer) viser at der er tale om x86 kode.

Fordelen ved CIL-kode er med andre ord, at den kan køre på mange platforme (i modsætning til native kode, maskinens eget instruktionssæt). Den erfarne programmør ved, at man allerede i 1978 havde et p-Code system på universitetet i San Diego. Dengang var der mange slags hjemmecomputere, og for at de studerende kunne køre alle mulige øvelsesprogrammer, - og spil - lavede man et system, som dannede byte-kode, d.v.s. kommandoer, som var i byte-form, og som blev udført af en fortolker - d.v.s. et program, som læste byte-koderne og udførte dem. Til sidst ender alting i at være maskin-instruktioner af den slags, som maskinen er bygget til (native code).

Mono kan køre CIL kode på flere måder

Læg mærke til, at den CIL fil, vi klippede i på foregående side, ikke var den .exe fil, som F# compileren (fsharpc) danner, men derimod en tekstfil genereret af ***monodis***-kommandoen.

På Unix (Linux/BSD/OS-X/AIX m.fl.) kan man køre et ***mono-program*** på flg. måder fra kommandolinjen:

```
./daxdivide.exe
```

eller

```
mono daxdivide.exe
```

Normalt skal man ikke forsøge at køre .exe kommandoer på et Linux system (med mindre man ved nøjagtigt hvor den kommer fra og hvad dens formål er.) Spørg, hvad det er for en type fil, vi har genereret, med ***file*** kommandoen:

```
file daxdivide.exe
daxdivide.exe: PE32 executable (console)
Intel 80386 Mono/.Net assembly, for MS
Windows
```

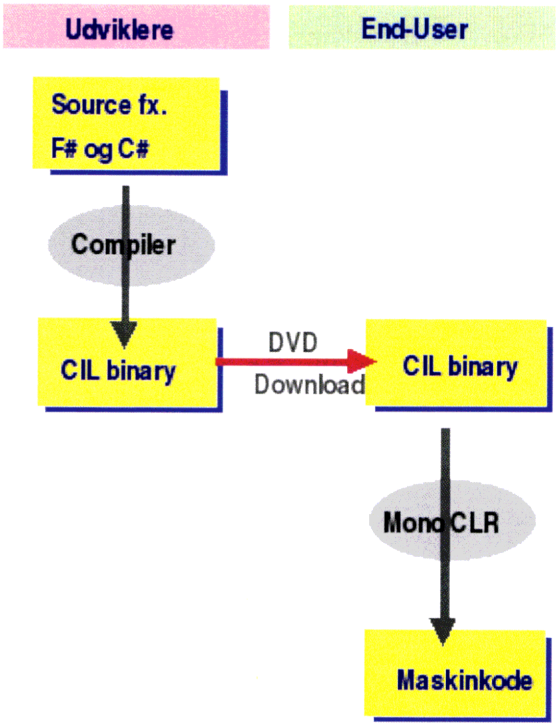
PE32: Portable Executable, en fil som indeholder CIL bytekoder + diverse environment information og symboler.

Hvordan kan Linux se, at det er et program, som kræver Common Language Infrastructure? Prøv igen:

```
./daxdivide.exe
lastcomm | head -n 2
cli              0.16 secs Wed Nov 12 13:40
binfmt-detector  0.00 secs Wed Nov 12 13:40
```

Der er klippet et par kolonner væk for at screen-dump kan være i bladet - men det vigtigste er med: Programmet ***binfmt-detector*** er kørt for at finde ud af, hvad det nu er for en fil - for den er jo ikke en almindelig (Linux/Unix) programfil.

De programmer, som er kørt sidst, står øverst, og øverst står ***cli,*** som er ***C**ommon **L**anguage **I**nfrastructure.* Det er denne infrastruktur, som i sidste ende åbner for funktioner til kommunikation via en touch-screen - og dermed muliggør start og kørsel af programmer, *apps,* på en mobiltelefon ved tryk på skærmen.



Common Language Infrastructure skal naturligvis være specifikt tilpasset hardwaren og vil normalt følge med Android systemer og Microsoft Mobile

Som det fremgår af illustrationen, kan flere sprog kompileres til CIL. Man skal huske, at det vil være specielle dialekter af disse sprog (med undtagelse af C# som er "født" til CIL).

➥***fortsættes side 17***

F# eller F sharp

Kan man få både funktionsprogrammering, imperativ- og objektorienteret programmering i ét og samme programmeringssprog?

Af David Askirk

F# logo - fra fsharp.org

I denne artikel kommer der en kort intro til F#. F# er et sprog fra Microsoft som kører på .NET platformen. F# er et funktionelt sprog så tankegangen bag sproget er anderledes end andre sprog.

"Jamen hvorfor sidder jeg DKUUG-nyt og læser om noget fra Microsoft der kører på .NET?" Jo, F# kører på .NET platformen, så derfor kører det også på Mono, som er en cross-platforms implementation af .NET. Derfor kan vi bruge F# på mange platforme, og ikke kun på windows. Læs mere om Mono andetsteds i bladet.

For at komme igang med F# skal det installeres. Det kan man gøre fx. med apt-get.

Når F# er installeret er den nemmeste måde at komme igang at bruge det interaktive miljø.

Det startes med kommandoen **fsharpi**.

Nu kommer der en prompt frem og systemet er klart:

```
>
```

Kommentarer kan skrives efter // - dobbeltslash:

```
> // filnavn: dkuug-fsharp-eksempel.fs
```

Til at starte med, lad os undersøge nogle operatorer. I F# bliver linjerne afsluttet med **;;** (to semikoloner).

Vi kan skrive

```
> 2+2;;
```

og systemet giver os resultatet 4.

```
> 2+2;;
val it : int = 4
```

F# fortæller os at værdien (val) med navnet **it** er en int og den har værdien 4. It er standard navnet der blot henviser til sidste resultat. F# gætter på typen, og gætter på at typen af resultatet er en int.

Endnu et regnestykke:

```
> 2.0/0.1;;
val it : float = 20.0
```

Her bliver resultatet en float med værdien 20.0.

Et sprog et ikke meget bevendt, hvis vi ikke kan gemme de værdier vi finder. F# er et sprog der har meget fra matematikken, herunder også variabel-erklæringer.

En variabel erklæres således:

```
> let a = 2;;
```

Systemet svarer så dette tilbage:

```
val a : int = 2
```

Her fortæller F# at a har værdien 2 som er en integer.

Beregninger

Vi kan også gemme resultatet af en beregning:

```
> let b = 2 + 2;;
```

Systemet fortager beregningen og svare dette tilbage:

```
val b : int = 4
```

Hvad nu hvis vi gerne vil lave mere avancerede ting?

Funktioner

F# er et funktionelt sprog, og derved er funktioner en vigtig del af sproget. I funktionelle sprog bliver der lagt op til at funktioner ikke har nogle side-effekter. Dette betyder en funktion altid giver samme resultat givet samme input.

En funktion i F# erklæres med denne syntaks:

```
let <funktions navn> <parametre> = <funktions body>
```

Eksempel:

```
let foo a b = a + b;;
```

Dette giver en funktion der hedder foo som tager to parametre a og b. Den laver så beregningen a + b og dette bliver returneret.

```
> let add a b = a + b
- let sub a b = a - b
- let mul a b = a * b
- let div a b = a / b;;
val add : a:int -> b:int -> int
val sub : a:int -> b:int -> int
val mul : a:int -> b:int -> int
val div : a:int -> b:int -> int
```

Læg mærke til her at vi erklærer fire funktioner på en gang. Det kan man nemlig godt, det er først ved den sidste hvor der bliver afsluttet med `;;` at det bliver sat i værk.

F# fortæller os vi har fire funktioner.

Typen af add er:

```
val add : a:int -> b:int -> int
```

Dette skal læses som:

Add er en værdi der 1) tager en int (nemlig a), og 2) returnerer ***en funktion der tager en int (b), som returnerer en int.***

Ekspansionen eller forklaringen af, hvad *add* er, kan fås ved at taste ***add;;*** - den angiver kun værdier/typer, ikke operation på de to parametre. Med andre ord, *sub* og de andre funktioner er i dette tilfælde af samme opbygning.

Dette virker som en anderledes forklaring i forhold til at funktionen blev defineret med to parametre. Læg mærke til at vi ikke har fortalt F# at vores funktion arbejder med integers, men det gætter F# selv på.

I et sprog som javascript ville det se nogenlunde således ud:

```
function add (a) {
       return function (b) {
              return a+b;
       };
}
```

Med *nogenlunde* menes, at der ikke umiddelbart er andre sprog, hvor man nemt kan skrive noget der ligner.

Lad os prøve dem af:

```
> add 1 2;;
val it : int = 3
```

Her kalder vi add med to parametre 1 og 2. Disse tal bliver lagt sammen og resultatet 3 bliver returneret.

Her kommer den første lille skægge ting med funktioner. Vores **`add`** er ikke bare en funktion som får to int og returnerer summen (som heltal, int).

Typen af **add** er (som vi så på side 7) mere end bare det, den er *en funktion der returnerer* ***en funktion, der returnerer en int.***

Dette gør, at vi kan køre funktioner halvt, kan man sige.

```
> let add2 = add 2;;
val add2 : (int -> int)
```

Her bliver der lavet en ny funktion add2, som er en add hvor der mangler en parameter. Dette gør, at vi har bundet den ene parameter, og kun holder én fri, som vi kan bruge på et andet tidspunkt.

```
> add2 3;;
val it : int = 5
```

Her kalder vi vores ny funktion med værdien 3 og resultatet 5 kommer frem, helt som ventet. Dette kaldes currying af funktioner.

En anden måde at binde funktioner sammen er en måde som er kendt fra unix verden. Nemlig muligheden for at pipe funktioner sammen. Nu vil vi gerne fortage denne beregning:

```
(3 - (2 + 3))
```

Dette kan vi gøre på to måder:

```
> sub 3 (add 2 3);;
val it : int = -2
```

Eller ved at pipe resultaterne sammen:

```
> add 2 3 |> sub 3;;
val it : int = -2
```

I begge tilfælde bliver add funktionen kørt først og derefter sub funktionen. Hvad man skal bruge afhænger af konventioner samt af situationen.

Et andet eksempel er brug af **printfn**:

```
> printfn "Res: %d" 34;;
Res: 34
val it : unit = ()
```

Her kan vi også fortage beregningen og derefter skrive den ud på skærmen:

```
> 2+2 |> printfn "Resultat: %d";;
Resultat: 4
val it : unit = ()
```

Type systemet i F#.

Da F# kører på .NET platformen er alle de typer, man er vant til fra .NET, til stede.

Dog er der en speciel type. Da alt i F# skal have en type, eller returnere en type, kan man ikke have void (typeløse) funktioner.

Derfor er der den specielle type der hedder **unit**, som bliver brugt når man andre steder ville skrive void. **Unit** er stadig en type der bliver returneret, dog signalerer **unit** at der ikke er en værdi.

Et eksempel er print funktionen.

```
> printfn "Hello World";;
Hello World
val it : unit = ()
```

Der er flere ting ved denne funktion. For det første har den en side effekt (den skriver til skærmen) og for det andet returnerer den unit som type, dvs. der kommer ingen værdi tilbage fra funktionen andet end unit.

Lister

Da F# er et funktionelt sprog er lister en stor del af sproget.

En liste i F# erklæres således:

```
> let a = [];;
val a : 'a list
```

Her fortæller F# os at listen a er en liste af en anonym type. Vi erklærede en tom liste uden elementer med [].

En liste uden elementer er ikke sjov, så lad os lave en liste med noget i.

```
> let a = [1;2;3];;
val a : int list = [1; 2; 3]
```

Her er en liste med tre elementer. Tre integers med værdierne 1, 2 og 3. Når en liste erklæres på denne måde, bliver elementerne adskilt af **;** (semikolon).

Dette kan også læses som en tom liste, hvortil der er tilføjet talet 3, dernæst tilføjet 2 for til sidste at tilføje 1.

Dette kan skrive i F# således:

```
> let b = 1::2::3::[];;
val b : int list = [1; 2; 3]
```

:: operatoren (dobbelt-kolon) tager et element og tilføjer det til en liste.

Når vi giver en liste med til en funktion har vi mulighed for at "skille" listen ad på denne måde.

Lad os skrive en funktion der finder det første element i en liste:

```
> let head x =
      match x with
            | [] -> failwith("Empty list")
            | a::[] -> a
            | a::rest -> a;;
val head : x:'a list -> 'a
```

Her kan vi se det er en funktion der tager en liste ind og returnerer et element.

Lad os prøve vores funktion:

```
> head [1;2;3];;
val it : int = 1
```

Vi kan se den virker.

Magien i denne funktion ligger i ***match .. with*** formen. Her går vi ind og prøver at matche det data vi får ind med de tre forskellige former:

1. En tom liste, så smider vi en fejl.
2. Hvis der kun er et element i listen.
3. Hvis der er mere end et element i listen.

Ved at bruge :: splitter vi listen op.

Sum af elementer i en liste

Lad os skrive endnu en funktion. Nemlig en laver en sum af elementer i en liste:

```
> let rec sumOfList x =
      match x with
            | [] -> 0
            | a::rest -> a+(sumOfList rest);;
val sumOfList : x:int list -> int
```

Her kan vi se at F# fortæller os at det er en funktion der tager en liste af integer og returnere end int. Lad os prøve den:

```
> sumOfList [1;2;3];;
val it : int = 6
```

I denne funktion bliver der brugt det at man kan matche med ***match .. with*** så en liste bliver splittet op. Da det er en funktion vi kalder rekursivt, dvs. funktionen kalder sig selv, putter vi **rec** ind efter **let**.

Lad os lave en funktion der kører en funktion for hvert element i en liste og returnerer en ny liste:

```
> let myMap f x =
      let rec innerMap f x acc =
            match x with
            | [] -> acc
            | a::rest -> innerMap f rest ((f a)::acc)
      innerMap f x [];;
val myMap : f:('a -> 'b) -> x:'a list -> 'b list

> let square x = x*x;;
val square : x:int -> int
```

Her laver vi to funktioner. Den anden, kaldet square, tager et tal ind og ganger det med sig selv.

Den første funktion er en funktion med en funktion inden i sig. Dette kan lade sig gøre da funktioner i sig selv også er værdier. Her er der en ydre funktion der tager i mod en liste og en funktion. Læg mærke til at F# kan se at funktionen tager en værdi ind og returnerer en anden værdi.

Vi kan ikke se noget om den indre funktion, da den er skjult. Dette gør også, at vores midlertidige lager, kaldet acc, er skjult. Dette er et eksempel på en closure.

Fakta-box

In programming languages, a **closure** (also **lexical closure** or **function closure**) is a function or reference to a function together with a *referencing environment*—a table storing a reference to each of the non-local variables (also called free variables or upvalues) of that function.

Lad os bruge denne map funktion:

```
> myMap sqaure [1;2;3;4];;
val it : int list = [16; 9; 4; 1]
```

Her kan vi se at vi får listen af kvadrat tal, men i omvendt rækkefølge. Dette skyldes denne linje i den indre funktion:

```
| a::rest -> innerMap f rest ((f a)::acc)
```

Den linje siger:

kør funktionen på resten af listen og dernæst, tag og kør funktionen f på det første element og tilføj det til den interne liste.

Egne typer

I F# er det muligt at definere sine egne typer. Dette gør det nemt at lave et program der passer til et bestemt domæne. Her vil der blive lavet et kryds og bolle spil.

Et spil skrevet i F#

Nu vil vi gerne erklære typerne til et kryds og bolle spil:

```
> type Cell =
  | X
  | O
  | E;;
type Cell =
  | X
  | O
  | E
```

Der skal ikke mere til. Nu findes type **Cell** der kan have tre værdier: X, O **eller** E.

I vores kryds og bolle spil bruger vi en liste til at gemme brædtet i.

Vi giver positionerne tal således:

```
0 | 1 | 2
---------
3 | 4 | 5
---------
6 | 7 | 8
```

Ved at gøre dette kan vi mappe positionerne i en liste til positionerne på et spille brædt.

Lad os oprette det tomme brædt, altså sådan et brædt ser ud når vi starter et nyt spil.

```
> let emptyBoard = [E;E;E;E;E;E;E;E;E];;
val emptyBoard : Cell list = [E; E; E; E; E; E; E; E; E]
```

Her kan vi se det er en liste af vores type Cell. Der er ni i alt. Dette er et tomt brædt som det ser ud inden spillet går igang.

For at se hvem der har vundet skal man undersøge alle mulighederne, vandret, lodret og diagonalt.

Fra frokoststuen:

A general user calling: "Who is General Failure, and why is he reading my disk?"

Lady calling in: "FAT bread failed? That is an insult!"

Kollega i direktionen: Jeg forstår ikke, der står tryk på ***alt + F1*** ... men når jeg trykker på alt, trykker jeg jo også på F1?

Fornærmet kunde: De siger, at ***alt*** er under ***kontrol***, men jeg synes, de sidder ved siden af hinanden!

Kryds og bolle spil:

Dette kan gøres således i F#:

cell (med lille forbogstav) er instantiering af noget - det er altså ikke en Cell-type med mindre vi gør den til det senere? Kan erstattes af whatever.)

```
> let hasWon cell board =
  match board with
        | [a;b;c;_;_;_;_;_;_] when a = cell && b = c && a = c -> true
        | [_;_;_;a;b;c;_;_;_] when a = cell && b = c && a = c -> true
        | [_;_;_;_;_;_;a;b;c] when a = cell && b = c && a = c -> true
        | [a;_;_;_;b;_;_;_;c] when a = cell && b = c && a = c -> true
        | [_;_;a;_;b;_;c;_;_] when a = cell && b = c && a = c -> true
        | [a;_;_;b;_;_;c;_;_] when a = cell && b = c && a = c -> true
        | [_;a;_;_;b;_;_;c;_] when a = cell && b = c && a = c -> true
        | [_;_;a;_;_;b;_;_;c] when a = cell && b = c && a = c -> true
        | _ -> false;;
val hasWon : cell:'a -> board:'a list -> bool when 'a : equality
```

Her kan vi se det er en function der tager et element at teste imod, en liste af Cell ind og returnerer sand eller falsk.

Vi laver samligningen ved at sige vi ønsker at tage fat i de tre første variabler - og vi er ligeglade med resten.

Dette bliver gjort i denne:

```
[a;b;c;_;_;_;_;_;_]
```

Da vi gerne vil undersøge om de tre valgte variabler er ens, samt om de er det samme som den brik vi undersøger for ligger vi en guard condition ind:

```
when a = cell && b = c && a = c
```

Denne siger at når a matcher den *cell* vi kigger efter, samt b er lig med c og a er lig med c så er der tre ens på plads 0, 1 og 2, og derved er der en spiller der har vundet.

Til sidst bliver der angivet hvad der returneres hvis systemet kan matche med de værdier vi har givet ind. Hvis der er fundet en vinder bliver der returneret true.

Hver mulighed i matchingen bliver adskilt af | (pipe) og den sidste er en catch-all der matcher alt og returnerer falsk. Den returnerer falsk hvis der ikke er fundet en vinder.

Med dette i hånden er det nemt at lave de to funktioner for at se om X eller O har vundet.

Vi benytter os af den delvise funktions opbygning således:

```
> let hasXWon = hasWon X;;
val hasXWon : (Cell list -> bool)
```

og

```
> let hasOWon = hasWon O;;
val hasOWon : (Cell list -> bool)
```

Ved at gøre dette, får vi to funktioner mere, som kan teste om enten X eller O har vundet. Læg mærke til at vi skrev en generel funktion og brugte currying til at give os to specifikke tilpassede funktioner.

For at fortage et træk på vores brædt skal vi lave en funktion der går ind og erstatter et element i en liste:

```
> let rec replaceNth l pos elem =
      match (l, pos) with
            | ([], _) -> []
            | (a::rest, 0) -> elem :: rest
            | (a::rest, n) -> a::(replaceNth rest (n-1) elem);;
val replaceNth : l:'a list -> pos:int -> elem:'a -> 'a list
```

Denne funktion giver vi en liste, en position der skal erstattes på og det element som skal sættes ind på det sted i listen.

Ved matchingen laver vi et trick, da vi gerne vil matche på listen og på vores position. Vi pakker dem sammen i en tupple, så vi kan bruge begge værdier i vores matching.

Nu kan den funktion der fortager et træk laves:

```
> let placeMove cell pos board = replaceNth board pos cell;;
val placeMove : cell:'a -> pos:int -> board:'a list -> 'a list
```

Den kalder blot replaceNth, dog pakket ind på en pænere måde.

Lad os lave et kald:

```
> placeMove X 5 emptyBoard;;
val it : Cell list = [E; E; E; E; E; X; E; E; E]
```

Den sætter X'et lige der hvor vi gerne vil have det.

Så har vi vores funktioner til at styre vores spil.

Disse funktioner kan man så kalde i den interaktive shell og man har derved et kryds og bolle spil i F#.

Herunder er et eksempel på et spil:

```
> let newBoard = placeMove X 4 emptyBoard;;
val it : Cell list = [E; E; E; E; X; E; E; E; E]

> hasXWon newBoard;;
val it : bool = false

> hasOWon newBoard;;
val it : bool = false

> let newBoard = placeMove O 2 newBoard;;
val it : Cell list = [E; E; O; E; X; E; E; E; E]

> hasXWon newBoard;;
val it : bool = false

> hasOWon newBoard;;
val it : bool = false

> let newBoard = placeMove X 3 newBoard;;
val it : Cell list = [E; E; O; X; X; E; E; E; E]

> hasXWon newBoard;;
val it : bool = false

> hasOWon newBoard;;
val it : bool = false

> let newBoard = placeMove O 8 newBoard;;
val it : Cell list = [E; E; O; E; X; E; E; E; O]

> hasXWon newBoard;;
val it : bool = false

> hasOWon newBoard;;
val it : bool = false

> let newBoard = placeMove X 5 newBoard;;
val it : Cell list = [E; E; O; X; X; X; E; E; O]

> hasXWon newBoard;;
val it : bool = true
```

→→→→ *forts.næste side* →→→→

Spillet slutter med X der vinder.

Vi har kigget på F# som er en anden måde at programmere på. I modsætning til objekt orienteret programmering er funktioner i højsædet i F#, samt adskillelsen af kode med side-effekter og det som ingen sideeffekter har.

Da F# kører på Mono er det til rådighed på alle platforme samt giver mulighed for at skrive kode i F# som så efterfølgende kan bruge i C# eller andre sprog på .NET/Mono platformen.

F# er et funktionelt sprog, der lægger op til at man adskiller kode, der har sideeffekter fra kode der ikke har. Det er generelt en god måde at programmere på, også i ikke funktionelle sprog. Så er det nemmere at have en funktion der beviseligt er korrekt.

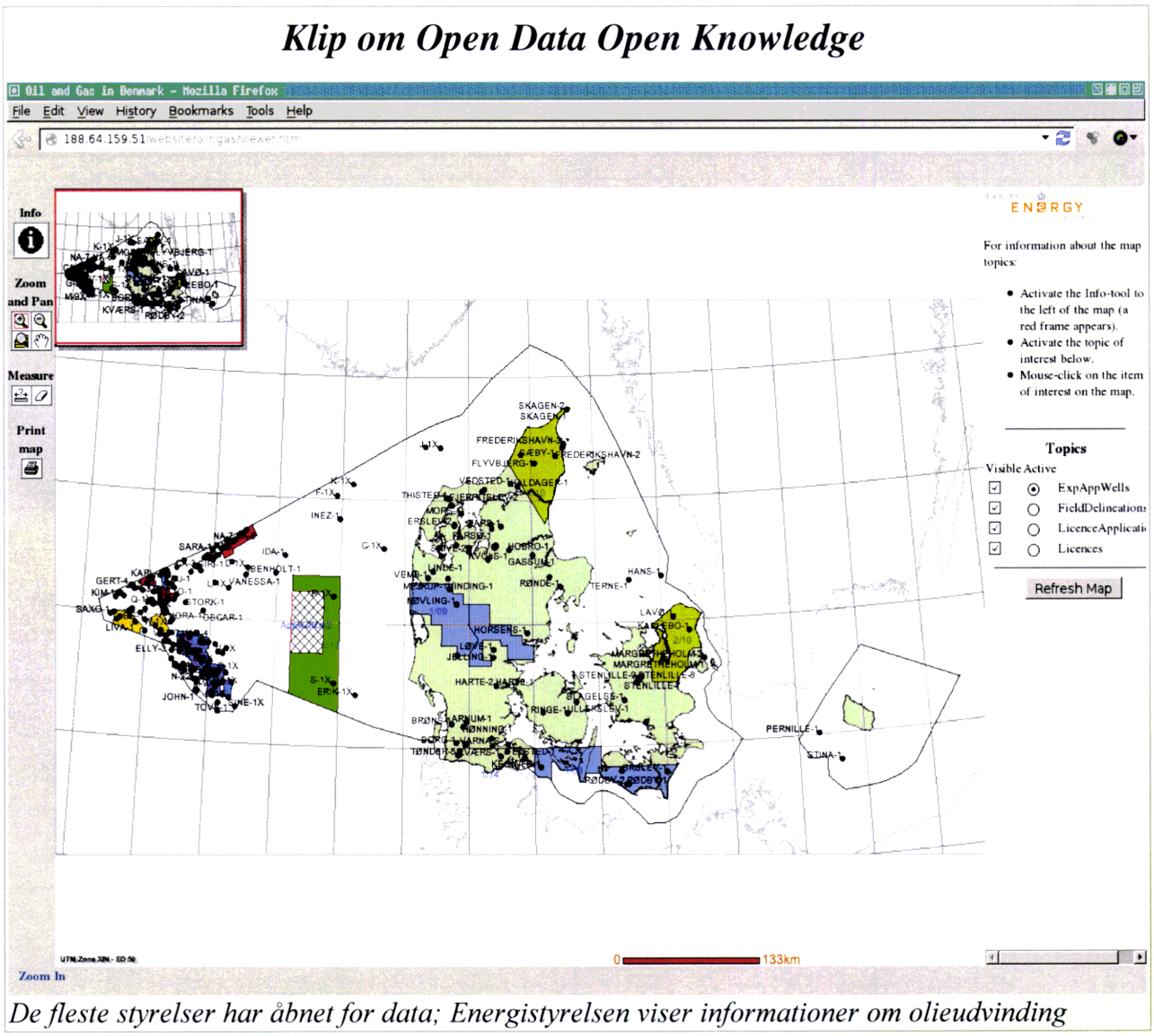
Links og bøger til at komme videre med F#:

http://www.tryfsharp.org/

https://www.microsoft.com/learning/en-us/book.aspx?ID=16172&mc=US

http://shop.oreilly.com/product/0636920024033.do

http://fsharpforfunandprofit.com



De fleste styrelser har åbnet for data; Energistyrelsen viser informationer om olieudvinding

Open Government Data -

Her er nogle pluk fra styrelser, ministerier og institutioner:

Digitaliseringsstyrelsen

Open Government Danmark har i lighed med mere end 60 andre lande tilsluttet sig det internationale initiativ, Open Government Partnership (OGP), som blev stiftet i 2011 af den amerikanske regering i samarbejde med syv andre lande, heriblandt Norge og Brasilien.

Energistyrelsen

Sidste år havde vi en artikel, hvor åbning af GIS (Geografisk Informations-System). Energistyrelsen, ens.dk, har også offentliggjort data om olieudvinding. Der er interaktivt kort med oplysninger, men også ganske almindelige rapporter. Der blev fundet olie i Danmark (Vesterhavet) 1966, og der er udvundet olie siden 1972. Alle rapporter gennem årene er offentligt tilgængelige online med link til download.

Københavns Universitet

Teknologiske trusler kan afværges med åbenhed om viden.

FRI VIDENSDELING

Som led i 100 års jubilæet for Niels Bohrs atommodel samler Københavns Universitet den 4.-6. december internationale forskere og beslutningstagere til konferencen "An Open World". Konferencen følger op på Bohrs kamp for en verdensorden, der bygger på åbenhed. Højaktuelle hovedtalere skal debattere fri cirkulation af viden – målet er at skrive et fælles brev til FN med anbefalinger til, hvordan international åbenhed kan bidrage til at håndtere teknologiske og videnskabelige udfordringer.

-

Konferencen "An Open World: Science, Technology and Society in the Light of Niels Bohr's Thoughts", tager udgangspunkt i et åbent brev, som den verdensberømte fysiker skrev til FN i 1950. I brevet opfordrer Bohr, på baggrund af udviklingen af atombomben, til åbenhed omkring forskning og teknologi for at forhindre destruktive kapløb drevet af mistillid og frygt og i stedet samarbejde om at realisere produktive muligheder.

- Med inspiration fra Bohrs brev til FN skal konferencens hovedtalere sende et nyt brev til FN, der giver anbefalinger til, hvordan internationale udfordringer kan håndteres med åbenhed og fri vidensdeling. Målet med konferencen og brevet til FN er at skabe debat om åbenhed. I modsætning til Bohr vil vi afsende et fælles budskab, der giver konkrete anvisninger, siger Ole Wæver.

På konferencen vil en lang række prominente internationale forskere og praktikere diskutere muligheder og risici ved åbenhed i forholdet mellem videnskab, teknologi og samfund. Heriblandt:

- *Jimmy Wales, stifter og talsperson for Wikipedia*
- *Irina Bokova, generalsekretær for UNESCO*
- *Susan Crawford, Obamas tidligere rådgiver for teknologi og videnskab*
- *Rolf-Dieter Heuer, direktør for CERN*
- *Sir Nigel Shadbolt, medstifter og bestyrelsesformand for The Open Data Institute*
- *Wael Ghonim, cyberaktivist med central rolle i den egyptiske revolution*
- *Richard Rhodes, Pulitzer-vindende forfatter og historiker med speciale i atomvåben*
- *Dennis Meadows, professor og medforfatter til den revolutionerende bog 'Limits to Growth'*
- *Abdallah Daar, kirurg og professor i folkesundhedsvidenskab med fokus på global sundhed*

OKFN, Open Knowledge Foundation Danmark

Der afholdes en dataworkshop i samarbejde med Informations (avisens) Datablog 26 november 16:30 i Informations lokaler i København. Man kan tilmelde sig hos Niels Erik Kaaber Rasmussen.

> *I Open Knowledge arbejder vi for udbredelse af åben viden. Vi arbejder for åben adgang, åben regeringsførelse, åben uddannelse, åbne data og åben-alt-muligt.*

dk.okfn.org

Internationalt

Open Data Barometer tager et overblik på Open Government Data (OGD) over hele verden ved at se på tilgængeligheden af 14 nøgleemner.

I rapporten for 2013 finder man dette kort:

The heat map below contrasts with the previous map of policy diffusion, showing the availability of open data currently lags behind the formation of open data policies in many countries.

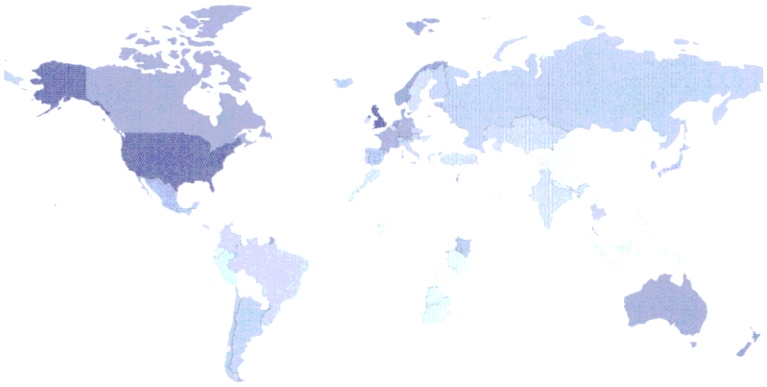
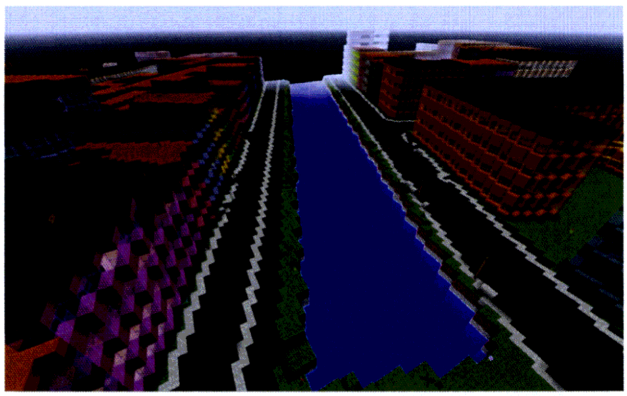


Figure 3: Heatmap of ODB implementation score by country - based on openness of 14 key datasets.

Hvis man vil deltage i den offentlige debat eller bedre, forske i hvordan økonomiske og politiske forhold er globalt set, er der andre kilder til internationale informationer: United Nations Development Programme (UNDP) og US CIA factbook.



Since the US government opened its troves of public data we've seen some pretty neat projects like climate-change prediction tools and deforestation-monitoring systems. Denmark, on the other hand, has taken a different approach: the Danish Geodata Agency used internally developed topographic maps and elevation models to build a 1:1 recreation of the happiest country within Minecraft's blocky confines.

Havde det ikke været fordi Svensk Politi havde fået kløerne i Warg til at begynde med, ville vi idag ikke have anet at der var indbrud hos CSC, ej heller at eller hvilke filer der var blevet kopieret.

Poul-Henning Kamp 31. okt. 2014 (Version2).

Shell Shock - eller chok

Shell Shock (på dansk Shell chok) er en familie af sikkerhedshuller i Bash

Den første offentliggørelse af dette sikkerhedshul kom 24. september 2014. Som med alle sikkerhedshuller var det meningen, at der skulle være en periode, hvor rapporter om denne kun blev sendt til virksomheder og institutioner med servere.

Men den dato, 24. sept. er jo ikke datoen for hvornår sikkehedshullet er opstået! Det går længere tilbage, meget længere, ligesom med Heartbleed buggen er det et sikkerhedshul, som meget vel kan være udnyttet af diskrete agenturer i fjendtligtsindede spiontjenester. Nu vil redaktionen på bladet her jo helst ikke blive kendt for at være tilhænger af konspirationer og lignende, men vi kan i det mindste sige så meget, at vore egne sikkerhedstjenester gør opmærksom på aktiviteter, som kan tyde på at andre stater har etableret servere, som kan medvirke til at stoppe trafikken på Internettet. Sådanne mere diskrete institutioner kan have et helt arsenal af sårbarheder, som kan udnyttes, hvis det skulle ønskes.

```
donax@sat2:~$ ls -l
total 8
-rw-r--r-- 1 donax donax  239 Mar 13  2013 MVI_1274.mov.kino
-rw-r--r-- 1 donax donax 2627 Nov 14 02:56 shellshock_test.sh
donax@sat2:~$ X='() { (a)=>\' bash -c "echo date"
bash: X: line 1: syntax error near unexpected token `='
bash: X: line 1:
bash: error importing function definition for `X'
donax@sat2:~$ echo $X

donax@sat2:~$ ls
echo  MVI_1274.mov.kino  shellshock_test.sh
donax@sat2:~$ cat echo
Fri Nov 14 03:04:41 CET 2014
donax@sat2:~$ ls
echo  MVI_1274.mov.kino  shellshock_test.sh
donax@sat2:~$
```

Kommandolinie eksempel som viser sikkerhedshullet

Eksemplet viser kommandolinien

```
$ X='() { (a)=>\' bash -c "echo date"
```

Bemærk placeringen af single-quotes (apostrof, kaldes den ofte på dansk). Det skyldes at der er space mellem parenteserne og de krøllede parenteser. Derimod gør det ikke noget, at der er "hul" mellem den sidste singlequote og ordet bash.

Der er *heller ikke en slut - krøllet parentes!* (Krøllede parenteser = braces). Det er selvfølgelig en fejl, men sikkerhedshuller består ofte i at udnytte, hvordan et program reagerer på fejl-input. Her er der yderligere fejl - efter den fejlagtige funktions-definition (slutter efter sidste singlequote) er der en kommando - og bash vil forsøge at gøre et eller andet med den.

Træd tilbage og se!

Der sker det, at bash danner en fil, "echo", med indholdet at output fra ***date*** kommandoen!

I eksemplet er indholdet af directory vist *inden* sårbarheden udnyttes. Der er *ikke* en fil ved navn echo.

Efter den mislykkede definition af en funktion X køres en bash-kommando, som sender output af kommandoen ***date*** til en fil, som hedder *echo.*

Det er en ganske almindelig fil, som tilhører den konto, bash kører på (user og group). Det åbner for mange problemer.

Naturligvis burde bash ikke køre nogen kommando og burde blot afvise hele den mislykkede funktionsdefinition.

Det er kun én af de sårbarheder, som har med denne mekanik at gøre og den kan udnyttes på mange måder.

Desværre kan vi her på redaktionen konstatere, at den samme sårbarhed findes for den pdksh, som vi ellers har været så glade for!

Desværre kan vi også konstatere, at den version af bash, som vi opdaterede med, ***ikke*** *var blevet rettet så den kunne modstå scripting med Shell Shock!*

Prompte reaktion

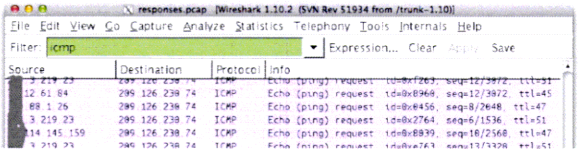
Ikke nok med at der er et sikkerhedshul! - Der er mange flere problemer med Shell Shock: Det kan konstateres, at der efter prenotifikation opstod en læk - en af de informerede parter sendte sårbarheds-specifikationen videre ud på nettet og blot timer efter kunne man konstatere, at der var angreb, ondsindede hackere, som etablerede bot-nets, d.v.s. lange lister med maskiner, som var blevet "åbnet" med diverse scripts som jo kan udføres, hvis denne bug ikke er rettet. Det medfører spekulationer om hvordan man kan sikre server-community mod lækager, og det er en sørgelig udvikling.

Flere variationer af Shell Shock

Det er ikke kun ved distribution af underlige shell-linjer (fx. i programpakker), at man kan angribe. Shell-sårbarheden kan udnyttes gennem HTTP protokollen, som har mange typer headers. Troy Hunt, website for "Alt hvad du behøver at vide om sårbarhed X" viser et eksempel:

```
target = 0.0.0.0/0
port = 80
banners = true
http-user-agent = shellshock-scan\
(http://blog.erratasec.com/2014/09/bash-
shellshock-scan-of-internet.html)
http-header = Cookie:()\
 { :; }; ping -c 3 209.126.230.74
```

Det var nødvendigt at bryde linjerne. Se mere på Troyhunt.com



Filtrering af tcpdump - CC - Creativ Commons, Troy Hunt

Navnet spiller selvfølgelig at det er samme ord som "granat chok", den traumatiske tilstand, som opstår hvis man har været udsat for tæt beskydning.

IPv6 i medierne

TLDs with IPv6 nameservers: 714
Percentage of TLDs with IPv6 nameservers: 96.0%

Screen-capture fra Tunnelbroker.net

➥ ***fortsat fra side 5***

Mono status og udvikling

Mono version 3.6.0 blev releaset i august 2014. Denne version giver kernen af .NET Framework og der er nu support for flere sprog:

har også support for Visual Basic.NET og C# version 2 3 og 4.0 Desuden har den LINQ support til objekter (fx. in-memory tabeller og lister). XML og SQL er en del af distributionen.

C# 4.0 er default modus for C# compileren. Windows Forms 2.0 er også mulige at bruge, men der udvikles ikke aktivt på det, og alt i alt er Monos understøttelse af Windows Forms ikke komplet.

Mono sigter efter at kunne give fuld support for .NET 4.0 med undtagelse af Windows Presentation Foundation (WPF) (som Mono-team'et ikke planlægger at supportere på grund af opgavens størrelse.

Der er underprojekter i Mono, som arbejder på implementering af de manglende dele af .NET.

Et af elementerne i .NET er en streaming media afspiller, Microsoft Silverlight, og der har været forsøgt en tilsvarende Open Source *Moonlight*, som Icaza i 2011 annoncerede at man ikke ville fortsætte, blandt andet kritiserede han det for at udvikle sig i retning af ***bloatware,*** d.v.s. software, som prøver for meget og fejler i et orgie af broken features.

Ud af disse og andre *gamle* nyheder kan man se at .NET både er et framework for applikationer på mobile enheder af meget forskellige fabrikater og typer, og at en vigtig del her som i andre applikationsmiljøer er beherskelse af markedet.

Alt i alt må man dog konkludere, det subset af .NET, som fungerer på Mono platformen, giver producenter af apps mulighed for at løse de fleste opgaver på mobile platforme.

- Og man kan tilføje, at navnet Mono måske i højere grad leder tankerne hen på et ensartet miljø for mange platforme, **mono**kultur.

Hvorfor er det godt at Mono supporterer flere sprog?

Når man ser nogle af de mange F# introduktioner, man kan finde på nettet, bl.a. på YouTube, opdager man hurtigt, at F# har mulighed for at bruge C# queries og andre libraries, som hører til .NET (og Mono) miljøet.

Denne egenskab gør sproget mere egnet til specielle opgaver, fx. et mailfilter skrevet i F# og et system til håndtering af indgående betalinger skrevet i C.

Paul Graham, som har arbejdet for bl.a. Yahoo Store, oprettede i midten af 1990'erne virksomheden ***Viaweb,*** der senere blev solgt til Yahoo. Han valgte *Lisp* - F# var ikke en mulighed dengang - men Lisp var (og er) et funktionsorienteret sprog ligesom F#, og Lisp kan (ligesom F# i Mono-miljøet) bruges sammen med andre sprog i en større applikation.

Graham skriver:

> *Viaweb at first had two parts: the editor, written in Lisp, which people used to build their sites, and the ordering system, written in C, which handled orders. The first version was mostly Lisp, because the ordering system was small. [...]*

Ovst. er fra en fodnote, men er nødvendig for at forstå resten:

> *So you could say that using Lisp was an experiment. Our hypothesis was that if we wrote our software in Lisp, we'd be able to get features done faster than our competitors, and also to do things in our software that they couldn't do. And because Lisp was so high-level, we wouldn't need a big development team, so our costs would be lower. If this were so, we could offer a better product for less money, and still make a profit. We would end up getting all the users, and our competitors would get none, and eventually go out of business. That was what we hoped would happen, anyway.*
>
> *What were the results of this experiment? Somewhat surprisingly, it worked. We eventually had many competitors, on the order of twenty to thirty of them, but none of their software could compete with ours. We had a WYSIWYG online store builder that ran on the server and yet felt like a desktop application. Our competitors had CGI scripts. And we were always far ahead of them in features.*
>
> *Sometimes, in desperation, competitors would try to introduce features that we didn't have. But with Lisp our development cycle was so fast that we could sometimes duplicate a new feature within a day or two of a competitor announcing it in a press release. By the time journalists covering the press release got round to calling us, we would have the new feature too. It must have seemed to our competitors that we had some kind of secret weapon--that we were decoding their Enigma traffic or something. In fact we did have a secret weapon, but it was simpler than they realized. No one was leaking news of their features to us. We were just able to develop software faster than anyone thought possible.*

Det er grunden til at Lisp (**LIS**t **P**rocessing) i dets mange varianter har spillet en stor rolle indenfor AI forskning. Et andet eksempel: Richard Stallman valgte Lisp som extension sprog for Emacs editoren, som han begyndte på i 1984. Og det var effektivt! Tænk lige på hvor små selv store computere var dengang ☺

I/O i F#

Men man **kan** lave I/O med F#. Se følgende minimal-eksempel fra Rosettacode.org:

```
open System.IO

[<EntryPoint>]
let main argv =
    File.ReadLines(argv.[0]) |> Seq.iter (printfn "%s")
    0
```

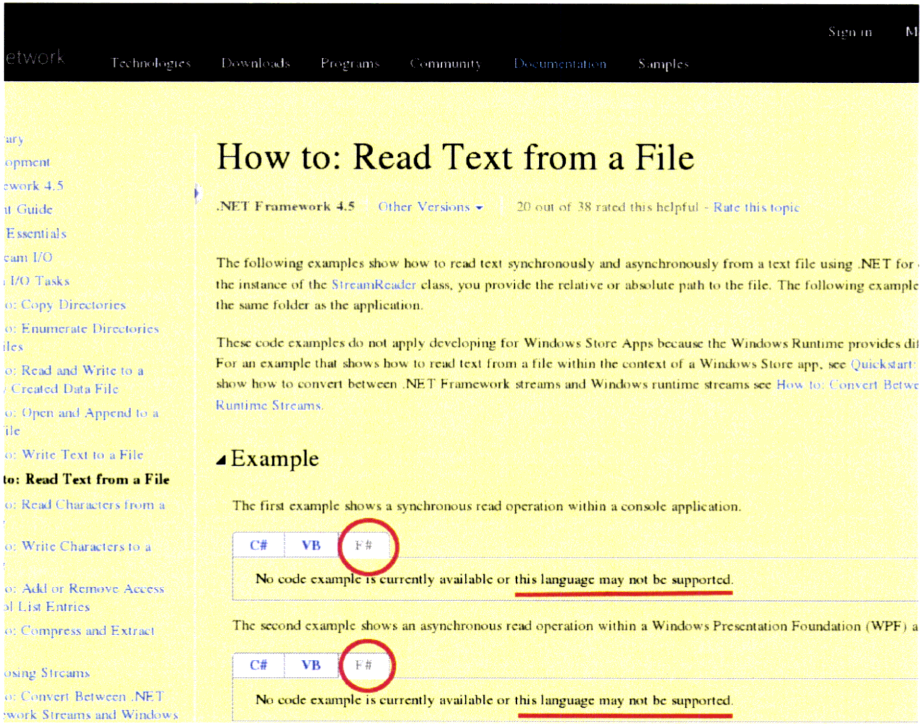
Men hvis man søger efter dokumentation af stream-I/O på fsharp.org, bliver man henvist til MSDN, Microsoft Developer Network - og dér finder man ikke altid dokumentation og eksempler for F#. Der står fx.:

> *The following examples show how to read text synchronously and asynchronously from a text file using .NET for desktop apps [...]*

og lige nedenunder:

> *No code example is currently available or this language may not be supported.*

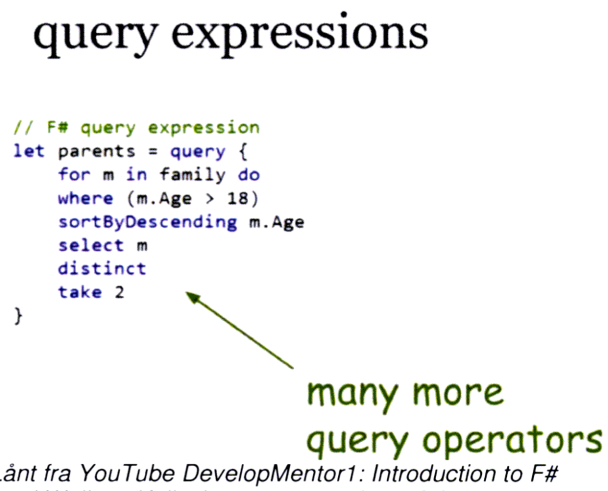
➥ ***forts. næste side***



Men andre steder kan man finde gode vejledninger og dokumentation; det vigtigste er specifikation af, hvordan F# programmoduler kan sættes i system med andre, og det sker typisk via en database. F# supporterer LINQ, og det er et nyttigt emne ☺ - som en smagsprøve er her et klip fra MSDN for F#:

```
// Use the OData type provider to create
// types that can be used to access the
// Northwind database.
open Microsoft.FSharp.Data.TypeProviders

type Northwind =
 ODataService <"http://s.org/North.svc">
let db = North.GetDataContext()

// A query expression.
let query1 =
  query { for customer in db.Customers do
                    select customer }
```

Query expressions er veldokumenterede - og der findes mange, mange, mange:



Lånt fra YouTube DevelopMentor1: Introduction to F# med Wallace Kelly, interessant og let at følge

En lignende query i C# kan fx. se sådan ud:

```
var queryLondonCust =
  from cust in customers
       where cust.City == "London"
       select cust;
```

Overordnet set er sprogene i Mono - miljøet sideordnede. De kan tale sammen via class interfaces/libraries.

Med tanke på artiklen i dette blad om F# kan det konkluderes, at F# skal i Mono kunne det samme som de andre sprog (særligt tænkes her på C#) men som vi har set, er der områder, hvor eksempler og vejledninger mangler, iøvrigt for operationer, som man typisk vil udføre med andre sprog eller system-tools.

Man kæder de forskellige (Mono-)sprog sammen via ***interface*** definitioner, svarende til prototypedefinitioner i header-filer for C-programmer. Interface, abstrakte typer, mv. er måden at administrere store projekter - de kan deles op i mindre, som derved er lettere at holde styr på.

I webserver miljø, hvor programmer kan kommunikere via pipes eller filer, vil man også kunne kombinere Mono-miljøet med andre (typisk ældre) funktionsmoduler.

Opsummering

Filosofien bag Mono er at være et nyttigt, effektivt værktøj til cross-platform programmering, som gør det muligt at afvikle programmer skrevet i forskellige sprog - også database query sprog mv. Vi ser det hver dag på smartphones.

Grunden til at disse programmer kører godt og ikke kræver omskrivning, hver gang der kommer en ny device på markedet, er at Mono - som er Open Source i modsætning til .NET - danner CIL kode, som derefter kan afvikles (eller oversættes) på brugerens device uden større forsinkelse. Ideen er ikke ny, men omfanget af libraries er stort og foreløbig er der udstrakt kompatibilitet; ***derfor*** opnår udviklere af apps og andre applikationer en større effektivitet og større udbredelse på alskens devices.

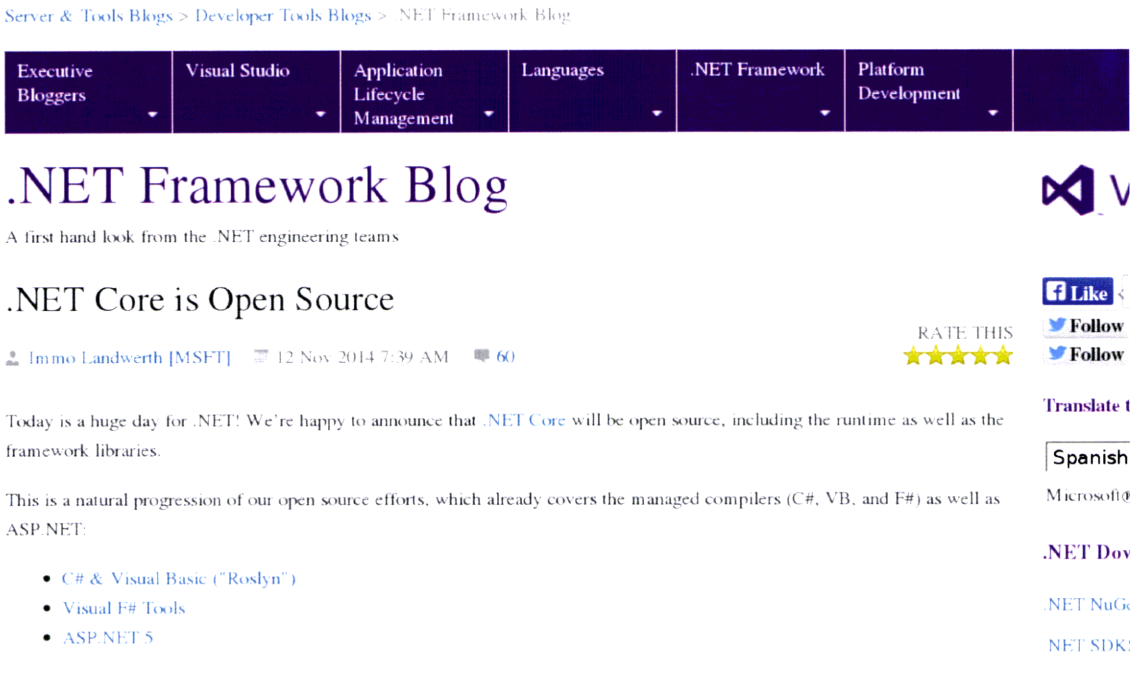
Med Mono bliver man en del af et stort netværk, og dermed også bliver man også afhængig af, at projektet fortsætter og at der vil være support-classes/libraries for nye devices. Derfor må man spørge sig selv om forretningsmodellen går fri af licenser og copyright i alle dele af systemet. (Fx. Apple-monotouch er ikke Open Source) Prisen er overkommelig, også for mindre virksomheder. Gevinsten er et stort marked for apps.

Sidste nyt! ... i skrivende stund:

Fremtiden for Mono

Netop i denne måned - faktisk i skrivende stund! - er der kommet en annoncering fra Microsoft Developer Net Blog, en af dem ses her: ***We are happy to announce that .NET Core will be open source, including the runtime as well as the framework libraries.***

Er det begyndelsen til enden af Mono, Open Source CIL-platform?


Server & Tools Blogs > Developer Tools Blogs > .NET Framework Blog

Executive Bloggers | Visual Studio | Application Lifecycle Management | Languages | .NET Framework | Platform Development

.NET Framework Blog

A first hand look from the .NET engineering teams

.NET Core is Open Source

Immo Landwerth [MSFT] 12 Nov 2014 7:39 AM 60

RATE THIS

Today is a huge day for .NET! We're happy to announce that .NET Core will be open source, including the runtime as well as the framework libraries.

This is a natural progression of our open source efforts, which already covers the managed compilers (C#, VB, and F#) as well as ASP.NET:

- C# & Visual Basic ("Roslyn")
- Visual F# Tools
- ASP.NET 5

Like Follow Follow

Translate t

Spanish

Microsoft

.NET Dov

.NET NuG

.NET SDK

Kommandocentralen

... i dag med eksempler på ting, som er lettere at gøre fra kommandolinjen:

Har du nogen gange siddet og tastet på en laptop med touchpad, og pludselig var cursoren langt borte: "det var som bare ... "

Hvor skal du støtte hånden uden at ramme touchpad og derved få cursoren til at fare forvildet henover skærmen?

syndaemon er et program, som overvåger tastaturet, og hvis der er aktivitet på tastaturet, bliver touchpad disablet.

Man kan indstille, hvornår den skal slås til igen - 2 sek. efter sidste tastetryk er default, så aktiveres touchpad. Poll-interval kan også justeres.

Det er et af de programmer, som man ikke aner kører hvis ikke man ser efter med proces-monitoreringsprogrammer.

Proces listning

Hvordan ser man, om den proces kører?

```
# ps -ef | grep synd
```

Fordelen ved denne metode er, at man ikke behøver at læse så meget. En taskmgr, hvor man kan søge på programnavne, vil selvfølgelig gøre næsten det samme.

Proces overvågning

Hvordan ser man tilstanden - hvad kan man få at vide om en kørende proces? Kort eksempel (man kan ***meget*** mere!):

```
# ps -lC syndaemon
```

Det viser bl.a. nice value, altså, hvor højt eller lavt prioriteret processen er. Nice value -20 er *superpower* - den får det meste.

Hvordan ser man, hvad lige netop den proces har "kostet" i tid?

```
# ps u -C syndaemon
```

De fleste brugere af Unix kender ***top*** - programmet, som viser hvilken proces, der belaster mest. Men somme tider er det minimale output fra ps lettere at bruge fx. i et script:

```
# ps --no-headers -o pcpu -C syndaemon
```

Denne kommandolinie vil ***kun*** skrive en procent (procent cpu, pcpu) på skærmen. -o betyder brug de felter, som jeg angiver i en kommasepareret liste efter -o fx.:

```
# ps -o pcpu,pmem,pid,ppid,cmd
```

--no-headers instruerer om at man ikke ønsker linjen med betegnelser (typisk i et script, hvor man bare vil have tallet).

-C søger på kommandonavnet. Det kan give lidt hovedpine en gang imellem at stave til et program, som man måske har startet via en GUI menu. I så fald må man gribe til andre midler fx:

```
# ps -o pcpu -p $(pgrep synda)
```

hvor -p betyder *ud fra proces ID (PID),* og pgrep leverer PID ud fra et navn eller bare en del af et navn.

Hørt i support-afdelingen:

Idle-processen tager nogen gange 99% cpu tid! ***dovne maskine ...***

--o--

En supporter skulle have en bruger til at taste et "større-end"-tegn. Brugeren mener ikke, at han har et sådan tegn på sit tastatur. "Jeg har et lille N og et stort N, men jeg har ikke et større N!"

SUPERUSERS Skal du til at udvikle Web Apps?
www.superusers.dk/web_apps.htm

Web Apps har større fokus end nogensinde – Web Apps giver platformsuafhængighed.
Client-side i browser: Programmeres i HTML5, CSS3 og JavaScript samt JavaScript .js-frameworks.
Server-side på en/flere servere: Programmeres i PHP/Perl/Python/Ruby/C# eller JavaScript/Node.js

SU-901 Web App Programmering Grundkursus (inkl. Mobile Device)

Du lærer begreber og teknologier og arbejder med nogle af markedets førende tools inden for udvikling af Web Apps.

Varighed / Pris: 2 dage / 8.900,- kr. (ekskl. moms)

Afholdelsesgaranti på understregede datoer:

8–9/12 • 8–9/1 • 5–6/2 • 9–10/3 • 9–10/4

SU-902 Web App Programmering af Datahåndtering

Du udvikl. en prof. Web App m. fokus på datalagring på mobil-device i filer og SQL-db samt brug af web-services og netværk.

Varighed / Pris: 3 dage / 11.100,- kr. (ekskl. moms)

Afholdelsesgaranti på understregede datoer:

26–28/1 • 23–25/3 • 27–29/5

SU-904 Web App Programmering af Brugergrænseflader – Foundation og Bootstrap

Du laver professionelle brugergrænseflader og sender information imellem forskellige skærmbilleder. Du anvender grafik.

Varighed / Pris: 3 dage / 11.100,- kr. (ekskl. moms)

Afholdelsesgaranti på understregede datoer:

19–21/11 • 23–25/2 • 20–22/4

SU-093 jQuery – Det samlede client webudviklingsforløb

Du bruger jQuery, det centrale JavaScript framework på client-side mellem frontend-layout og JavaScript funktionalitet.

Varighed / Pris: 3 dage / 12.600,- kr. (ekskl. moms)

Afholdelsesgaranti på understregede datoer:

19–21/1 • 16–18/3

SU-905 Web App Programmering af Hardware / Services – Phonegap

Du laver Web Apps, der håndterer hardware/services: fx kontakter/kalender, GPS/maps, kamera/video, mikrofon, tlf, mail …

Varighed / Pris: 3 dage / 13.500,- kr. (ekskl. moms)

Afholdelsesgaranti på understregede datoer:

19–21/11 • 14–16/1 • 11–13/3 • 4–6/5

SU-095 Node.js – Det samlede server webudviklingsforløb (inkl. Raspberry Pi computer)

Du lærer at bruge Node.js, et højperformance server-framework, som gør at al server-side webudvikling foregår med JavaScript.

Varighed / Pris: 3 dage / 13.500,- kr. (ekskl. moms)

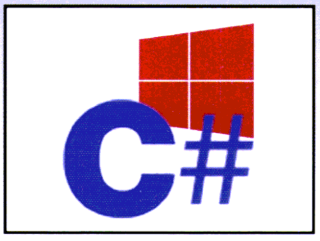
Afholdelsesgaranti på understregede datoer:

1–3/12 • 18–20/2

Bestil dine kurser nu, hvor du husker det, og mens der er plads

- **Via web:** superusers.dk
- **Via mail:** super@superusers.dk
- **Via telefon:** 48 28 07 06

Kurser i mobile platforme – se: www.superusers.dk/mobile-platforme.htm



Karlebogaard: Karlebovej 91 • 3400 Hillerød & **Kampehøjgaard**: Krajbjergvej 3, Vorre • 8541 Skødstrup
Tlf.: 48 28 07 06 • **Mail:** super@superusers.dk • **Web:** www.superusers.dk • **Alle priser er ekskl. moms**