Kursens mål

Önskemål från fortsättningskurser

- Felsökning
- Programmering i C
- Programmeringsteknik
- Kompatibel kurs
- Algoritmer och datastrukturer

Efter genomförd kurs 2D1322 ska du kunna följande:

- felsöka med spårutskrifter i program för att upptäcka fel,
- använda intuitiva variabelnamn
- identifiera hårdkodning och koduppreprening
- välja lämplig algoritm till ett givet problem,
- jämföra algoritmer med avseende på tids- och minnesåtgång,
- beskriva olika algoritmer för sökning och sortering och deras egenskaper,
- formulera och implementera rekursiva algoritmer,
- modellera verkliga problem som sökproblem och implementera algoritmer för breddenförstsökning, djupetförstsökning och bästaförstsökning,
- beskriva grundläggande komprimeringsalgoritmer och i vilka typer av komprimering dom används,
- konstruera en automat för textsökning och beskriva hur den fungerar,
- implementera och använda stackar och köer,
- implementera insättnings-, genomgångs- och sökoperationer i binära sökträd och allmänna träd samt använda dessa,
- använda prioritetsköer,
- identifiera problem där datastrukturerna ovan är användbara och konstruera enkla algoritmer med dessa,

för att du ska:

- få ett bra självförtroende för programmering
- bli bra på att lösa problem med programmering
- kunna felsöka och rätta befintliga program
- kunna använda datalogiska metoder i tillämpningsprojekt

få tillräckliga förkunskaper för att kunna läsa fortsättningskurser i datalogi.

Programmeringsteknik

Användarvänlighet

Informativa utskrifter

Programmet ska tala om för användaren vad programmet gör i varje steg och vad för inmatningen som förväntas. Ett dåligt exempel kan se ut som nedan till vänster.

Man ska inte behöva titta i en manual eller ännu värre själva programkoden för att förstå vad som händer. Att köra programmet måste vara självinstruerande.

```
Ge tal 1  : 26
och tal 2 : 54
29 31 37 41 43 47 53
```

```
Hej och välkommen till primtalsprogrammet. Programmet
skriver ut alla primtal i ett intervall du definierar.

Ange lägre gränsen i intervallet: 26
Ange högre gränsen i intervallet: 54
De primtal som finns mellan 26 och 54 är:
29 31 37 41 43 47 53
```

Det högra exemplet är mycket lättare att förstå när man kör programmet.

Enkel inmatning

Anmärk på programmet om inmatningen onödigt krånglig.

Förutom att vara väldigt sen med att kläcka ur sig att det inte finns något flyg den önskade resdagen så verkar det inte finnas något sätt att boka en mängd biljetter. Att boka en klassresa med det systemet skulle vara väldigt enerverande.

```
...
Vill du boka en biljett?ja
Varifrån åker du? Arlanda
Vart ska du åka? Kastrup
Vilken månad ska du åka? Mars
Vilken dag ska du åka? 25
Vill du boka returbiljett?ja
Vilken månad ska du tillbaka? April
Vilken dag ska du tillbaka? 5
Det går tyvärr inget flyg den 25 mars.
Försök igen

Vill du boka en biljett?ja
Varifrån åker du? Arlanda
Vart ska du åka? Kastrup
Vilken månad ska du åka? Mars
```

Programmerarvänlighet

Vettiga namn

Programmet ska ha intuitiva namn på variablerna. För den som skrivit programmet är allt ofta självklart inte för den som granskar.

Studera dessa två exempel, Det är samma kod som utförs. Vilken är mest intuitiv?

```
pelle = 0
for rut in tor:
    if pelle < rut:
        pelle = rut
print pelle
```

```
max = 0
for i in vek:
    if max < i:
        max = i
print max
```

Man ser nu lättare vad koden gör, koden sparar undan det högsta värdet i vektorn till variabeln max. Fortfarande är det inte optimalt bra namn på vektorn. Vad för slags värden innehåller den? Är det löner, skottstatistik eller vad?

Generellt brukar bra namn på metoder oftast vara verb som beskriver vad metoder gör. Klasser och variabler är ofta substantiv som beskriver vad klassen/variabeln är.

Det ska sägas att det är väldigt svårt att komma på bra namn.

Kommentarer

Alla klasser och metoder måste kommenteras. Berätta syftet med klassen/metoden. Det ska räcka att läsa kommentar och metodhuvud för att förstå hur en metod ska användas man ska alltså inte behöva titta i koden.

In och utdata till metoder måste kommenteras. Det gäller både **returvärden, parametrar** och eventuella **instansvariabler** metoden använder sig av. Alltså om metoden använder sig av instansvariabler ska detta kommenteras.

Kommentarer ska inte förklara hur Python fungerar. Förutsättningen är att den som läser källkoden redan vet hur man programmerar. Kommentarer som förklarar t ex att en if-sats gör ett val och att en slinga upprepar något ska inte vara med.

Konsekvent språk

Var konsekvent i ditt språkval. Alla variabel/metodnamn på ett språk. Alla kommentarer på ett språk. Det är OK att ha engelska variabel/metodnamn och kommentera på svenska. Undvik *svengelska (sejva svenska språket)*

Felhantering

Felhantering kan göras på lite olika sätt. Man kan till exempel kolla returvärden för varje funktionsanrop. Det blir väldigt jobbigt i längden.

```
z = f1()
if z > 0: z = f2()
else : print "ERROR!"
if z > 0: z = f3()
else : print "ERROR!"
```

```
try:
    z = f1()
    z = f2()
    z = f3()
except:
    print "Error"
```

Ett bättre sätt är att låta funktionen testa villkoret och slänga ett undantag - *exception*.

Ofta behövs en slinga, t.ex. vid inmatning för att programmet ska kunna gå vidare.

Har man många inläsningar från tangentbordet där fel kan uppstå är den bästa lösningen att man skriver en metod som tar hand om inläsningen.

```
Ange täljare: 1000
Ange nämnare: 0
Oj oj, nämnare får inte vara noll försök
igen.
Ange nämnare: 10

1000 delat med 10 blir 100
```

Lämplig uppdelning av funktioner

Sträva efter att ha specialiserade metoder som bara gör en sak.

Koden bredvid gör flera saker; frågar efter en fil, läser in allt data från filen, gör om datat till heltal, stoppar in heltalen i en heltalsvektor och returnerar denna.

```
def open_file():
    filnamn = raw_input("vad heter filen? ")
    filnamn = str(filnamn).strip()
    print "Öppnar", filnamn
    f = file(filnamn)
    vek = f.readlines()
    intvek = []
    for x in vek:
        intvek.append(int(x))
    print intvek
    return intvek
```

Det är bättre att dela upp dessa uppgifter
på flera metoder så att metodanropen blir:

```
filNamn = fragaFil()
fildata = laesFranFil(filnamn)
intresseantaTal = konvertera(fildata)
```

Programmet blir mer flexibelt. Metoden frågaFil kan skrivas om till ett grafiskt GUI där man klickar på rätt fil. Metoden laesFranFil kan användas i andra sammanhang då man vill läsa från fil. Man kan skicka fildatat till en ny metod som kontrollerar data innan man anropar konvertera.

Återanvändbara moduler

Funktioner kan man i sin tur gruppera ihop till moduler som man kan återanvända i andra sammanhang.

Ingen koduppprepning

Ett vanligt nybörjarfel när man programmerar är att använda taktiken *klippa och klistra*. Det leder dock till kod som är väldigt svår att underhålla. Om man ändrar på ett ställe måste man göra samma ändring på flera parallellställen.

Ingen hårdkodning

Undvik att hårdkoda värden så gott det går. Om du behöver använda siffervärden kan man deklarera dessa som konstanter.

I exemplet nedan finns både koduppprepning och hårdkodning samt dessutom ett fel.

```
for i in v:
    if i == 5:
        print "5 hittades"

for i in x:
    if i == 25:
        print "25 hittades"

for i in y:
    if i == 14:
        print "13 hittades"
```

Skriv om koden så att det räcker med
Enbart två anrop

```
find_int(v, 100)
find_int(x, 100)
```