# Introduksjon til vitenskapelig programmering
# Uke 1
# Sandvika vgs

Jonas van den Brink

j.v.d.brink@fys.uio.no

February 4, 2014

Denne teksten er ment som en kort oppsummering av det vi håper å ha kommet igjen etter første uke. Programmering er en ferdighet det tar tid å lære seg, og den beste måten å lære det på er rett og slett å prøve seg frem. Vi håper derfor at dere tar dere tid til å se litt på programmering på egenhånd, og skriver noen korte, enkle, og kanskje teite programmer.

## Installasjon

Vi kommer til å bruke programmeringsspråket *Python*. Det er i prinsippet gratis, og det finnes mange forskjellige versjoner, og mange forskjellige tillegspakker. Den enkleste måten å installere alt vi trenger for dette prosjektet er å installere en samlepakke som heter *Enthought Canopy*. Den laster du ned fra linken her

`https://www.enthought.com/downloads/`.

Når filen er ferdig lastet ned, kjører du den og følger instruksene. Du kan la alt av innstillinger stå på standard om du ikke har andre preferanser.

Når du har installert Canopy, kan du starte programmet. Siden det er en samlepakke er det en del funksjonalitet vi ikke kommer til å bruke, så ikke bli skremt av det kanskje ser litt komplisert ut.

## Hva er egentlig programmering?

Programmering går ut på å gi instrukser til datamaskinen. Disse instruksene skriver vi inn som kommandoer i en tekstfil. Denne tekstfilen kan så *kjøres* av datamaskinen, som da tolker kommandoene vi har skrevet. Det er det som er et dataprogram. Kommandoene vi skriver må følge et bestemt programmeringsspråk, og det finnes fryktelig mange slike språk idag. Vi kommer til å holde oss til Python, et av de mest brukte programmeringsspråkene idag. Python er også et fint språk å begynne med om man aldri har programmert før.

Moderne datamaskiner er fryktelig raske, men de er desverre ikke særlig smarte. Når vi programmerer må vi derfor være flinke til å gi helt riktige instrukser, om vi gir feil instrukser vil enten datamaskinen ikke skjønne hva vi mener, eller den vil rett og slett gjøre feil ting. Du må ikke være redd for å gjøre feil når du programmerer, det greier man rett og slett ikke å unngå. Det viktige er at du prøver å forstå *hva* som gikk galt, og hvordan man kan rette det opp. Selv de beste programmererene i verden bruker mye av tiden sin på å rette opp i feil i programmer. Feil i koden blir ofte kalt *bugs*, og det å rette opp i dem blir derfor kalt *bugfixing*.

## Vi setter igang

Om vi starter Canopy, og velger *Editor*, får vi opp et vindu der vi kan skrive et slikt dataprogram. Du skriver da koden inn i det største vinduet øverst. Du kan lagre programmet ditt med det navnet du vil, men du må legge på endelsen `.py`, for Python. Etter du har lagret programmet ditt, kan du kjøre det ved å klikke på *Run*-knappen, som er en grønn pil på toppen av editoren. En snarvei for Run er `ctrl+R`.

La oss se på et eksempel på et enkelt dataprogram:

```
# Calculating the area and volume of a football

from pylab import pi

r = 10

area = 4*pi*r**2
volume = (4./3)*pi*r**3

print "A football with radius:", r
print "Has an area of:", area
print "And volume of:", volume
```

Her er det ikke viktig at du skjønner alt som skjer i detalj, men la oss prøve å skjønne hovedtrekkene. Når vi trykker på Run-knappen, starter programmet med å tolke det som er skrevet, linje for linje. La oss derfor forklare det som skjer nedover i programmet, linje for linje.

```
# Calculating the area and volume of a football
```

Fordi den første linjen begynner med tegnet `#`, betyr det at denne linjen ikke tolkes av datamaskinen i det heletatt. Vi kaller en slik linje for en kommentar, og det er rett og slett en forklaringstekst til enten oss selv, eller andre som skal lese koden vår. Vi skjønner altså at dette enkle programmet skal regne ut arealet og volumet av en fotball. Merk også at hele linjen er en annen farge fra resten av koden, dette gjør Canopy for at det skal være lettere å tolke koden.

```
from pylab import pi
```

Denne linjen ser kanskje litt avansert ut, men her importeres rett og slett tallet $\pi$. Vi trenger $\pi$ for å regne ut arealet og volumet til en kule, men den er ikke originalt tilstede i Python, vi må *importere* den fra tillegspakken *pylab*. Det finnes veldig mange slike tileggspakker i Python, til mange forskjellige bruksområder. I vårt tilfelle inneholder Pylab alt vi kommer til å trenge.

```
r = 10
```

Her defineres det at det finnes en *variabel* som heter $r$, og at den skal være 10. Utifra kommentaren på starten av programmet skjønner vi at dette mest sannsynligvis er radiusen til fotballen. Denne linjen sier altså at radiusen er 10.

```
area = 4*pi*r**2
```

Her regnes arealet til fotballen ut fra den matematiske formelen $4\pi r^2$, og resultatet lagres i en *variabel* som kalles `area`.

```
volume = (4./3)*pi*r**3
```

Og her regnes volumet ut fra $\frac{4}{3}\pi r^3$, og resultatet lagres i `volume`.

```
print "A football with radius:", r
print "Has an area of:", area
print "And volume of:", volume
```

Til slutt kommer tre veldig like linjer. Alle bruker kommandoen `print`, som forteller Python at vi skal skrive et resultat til skjermen, slik at brukeren kan lese det. Tekst vi skriver omsluttes av fnutter: `"`, og vi ber Python skrive ut resultatene av utregningene etter teksten.

Når programmet kjøres, får vi dette resultatet:

```
A football with radius: 10
Has an area of: 1256.63706144
And volume of: 4188.79020479
```

## Variabler og regning

I eksempelet vi nettopp så på, ble vi introdusert til et veldig viktig konsept i programmering, nemlig variabler. Variabler bruker vi når vi vil at datamaskinen skal huske på en verdi vi gir den, eller noe den regner ut.

Hvis vi for eksempel skriver

```
a = 6
```

vil datamaskinen opprette en variabel som heter `a`, som inneholder *verdien* 1. Den vil så huske på denne variabelen helt til programmet er ferdig å kjøre. Hvis vi for eksempel vil skrive ut innholdet av en variabel til skjerm, bruker vi `print` commandoen, så å skrive

```
print a
```

gjør at det skrives ut et ettall til skjermen.

Vi kan også regne med en eller flere variabler. Hvis vi for eksempel også definerer en variabel `b`, ved å skrive

```
b = 3
```

Så kan vi bruke regne med disse variablene, hvis vi foreksempel skriver

```
print a + b
print a - b
print a * b
print a / b
```

får vi følgende resultater

```
9
3
18
2
```

Merk at `a` og `b` ikke endrer seg når vi regner med dem på denne måten. Dette kan vi dobbeltsjekke ved å skrive dem ut på nytt:

```
print a
print b
```

som gir

```
6
3
```

Hvis vi ønsker å endre en variabel vi allerede har definert, kan vi redefinere den, så vi kan nå skrive

```
a = -2
```

Merk at dette *overskriver* den gamle verdien av `a`, slik at programmet ikke husker hvilken verdi den var før. Vi kan også definere en variabel ved for eksempel å skrive

```
x = -4
y = 6
z = x + y
```

Her definerer vi først `x` og `y`, og deretter `z` som blir satt til resultatet av utregningen `x+y`. Om vi nå skriver ut `z` ser vi at den blir `2`, som forventet. Merk at det som skjer her, er at høyresiden regnes ut, og resultatet lagres i `z`-variabelen. Python husker altså ikke hvor verdien 2 som lagres i `z` kommer fra, den bare husker selve verdien. Prøv for eksempel å forklare hva som vil skrives ut om vi kjører denne lille kodesnutten:

```
x = 3
y = -3
z = x + y
y = 6
print x
print y
print z
```

Vil `z` inneholde verdien 0, eller 9? Bare prøv, og se hva som skjer!

Merk at likhetstegner, `=`, har en ganske annerledes betydning i programmering enn den har i matematikk. I matematikk er vi vant til at den brukes for å angi en likhet, altså en ligning. Det har for eksempel ingenting å si om vi skriver

$$x^2 = 4 \qquad \text{eller} \qquad 4 = x^2,$$

i matte. I programmering fungerer derimot ingen av disse. Tegnet `=`, brukes i Python alltid til å definere (eller redefinere) variable. Den virker alltid ved at den regner ut det som er på høyre-side, og så lagrer resultatet i variabelen på venstre side. Det er kanskje altså mer fornuftig å tenke på det som en slags pil, som peker fra høyre mot venstre.Kodesnutten

```
x = 9
y = 4
z = x*y
```

Kan altså tolkes som

$$x \leftarrow 9$$
$$y \leftarrow 4$$
$$z \leftarrow x * y$$

Når du begynner å skjønne denne tankegangen kan vi begynne å gjøre ting som kanskje ser litt rare ut. Vi kan for eksempel skrive

```
x = 3
x = x + 10
```

Fra et matematisk ståsted ser dette fryktelig ut, ligningen

$$x = x + 10,$$

gir ingen menining. Men i programmering er dette ganske greit. Først sier vi at `x` skal være 3, så regner vi ut høyre-siden, som da vil si $10+3 = 13$, og så lagrer vi 13 i `x`. Du kan sjekke at det er dette som faktisk skjer, ved å printe ut `x` og sjekke selv.

Hitill har vi bare vist eksempeler på variabler som inneholder tall, men de kan også inneholde andre ting, som for eksempel tekst, sannhetsverdier, og andre, mer kompliserte ting. Vi kan også gi dem mer kompliserte navn, i praksis er dette ofte lurt, fordi det gjør det lettere for oss å huske på hva de forskjellige variablene er i et langt og rotete dataprogram. I eksempelprogrammet vi viste på starten brukte vi variabelnavn som `area` og `volume`.

La oss se på et enkelt eksempel på hvordan vi kan bruke en variabel med tekst

```
name = "Jonas"
print "Hi", name, "! Hope you have a nice day =)."
```

Merk at for å opprette en variabel med tekst, så lar vi teksten stå i fnutter: `"tekst"`, dette er for å få Python til å skjønne at den ikke skal prøve å tolke teksten som kode. En slik tekst som ikke er kode kalles en *tekststreng*. Merk også at print kommandoen vi gi er litt mer komplisert en vanlig, fordi den skriver ut tre ting. Først skriven den ut tekstrengen `"Hi!"`, merk at fnuttene ikke vises på skjermen når utskriften kommer, så skriver vi ut innholden av variabelen `name`, og så skrives den siste teksstrengen.

### Datatyper og heltallsdivisjon

Så langt har vi sett at variabler kan ha forskjellig typer innhold. Python har en måte å sjekke hvilken type innhold en variabel har, som vi kan bruke som følger

```
a = 4
x = 3.14
name = "Jonas"
print type(a)
print type(b)
print type(c)
```

og resultatet blir som følger

```
<type 'int'>
<type 'float'>
<type 'str'>
```

Som vil si at `a` er av type *int*, som står for integer, altså heltall, `x` er *float*, som betyr desimaltall, og `name` er *str*, altså en tekststreng.

For vårt bruk er det ikke så viktig å ha oversikt over alle disse datatypene og hvordan de oppfører seg. Men vær obs på at dere kommer nok til å gjøre litt feil med disse, så det kan være greit å tenke litt over hvordan ting fungerer i detalj i blant.

En veldig vanlig feil å gjøre for eksempel, er noe som kalles *heltallsdivisjon*. Dette er når vi har to tall som Python tenker på som heltall, og prøver å dele disse på hverandre

```
a = 5
b = 3
print a/b
```

I dette tilfellet forventer vi kanskje resultatet

$$5/3 = 1.666667,$$

men det vi får er bare 1. Det er fordi Python tenker på `a` og `b` som heltall, og tror derfor at det vi er interessert i er et heltall! For å tvinge python til å skjønne at vi vil faktisk ha desimaltall, bør vi definere `a` og `b` som desimaltall, det kan vi gjøre ved å skrive

```
a = 5.0
b = 3.0
```

eller bare

```
a = 5.
b = 3.
```

Når vi gjør en divisjon med et desimaltall og et heltall, tolker Python det som at vi vil ha "vanlig" divisjon, så hvis vi er interessert i å regne ut kinetisk energi for eksempel, som matematisk sett har formelen

$$K = \frac{1}{2}mv^2,$$

kan vi bruke utrykket:

```
kinetic_energy = 1./2*m*v**2
```

Et par ting å merke seg her: Vi skriver nevneren i brøken med et punktum, for å unngå heltallsdivisjon (`1/2` hadde gitt 0), og vi skriver "opphøyd i" med `**`.

## Matematiske funksjoner

Om vi er interessert i å regne med vanlige matematiske konstanter og funksjoner, som for eksempel $\pi$, $e^x$, $\sin(x)$, osv, ligger disse lagret i pylab pakken. Vi kan da enkelt importere dem med kommandoer på formen:

```
from pylab import pi, exp, sin
```

og da bruken dem på følgende måte:

```
print sin(2*pi)
```

Vi kommer til å vise dere hvordan dere kan lage deres egne matematiske funksjoner i Python, som for eksempel:

$$f(x) = 4x^2 + 3x + 4,$$

i løpet av de neste ukene.

Hvis vi har lyst til å importere alt som ligger i pylab pakken, kan vi skrive

```
from pylab import *
```

## Oppgave 1

(a) Lag et skript som skriver ut teksten "Hello, World!" til skjermen.

(b) Lag et skript som skriver ut verdien av 4*5+2 til skjermen, blir resultatet annerledes om du skriver 2+4*5?

(c) Lag et skript som regner ut $2^{10}$ og skriver resultatet til skjermen.

(d) Lag et skript som regner ut $\sqrt{3}$ og skriver resultatet til skjermen. Her må du først importere `sqrt`-kommandoen (squareroot) fra pylab.

## Oppgave 2

For å regne oss om fra en temperatur oppgitt i grader Celsius $C$, til grader oppgitt i Fahrenheit $F$, bruker vi formelen

$$F = \frac{9}{5}C + 32.$$

(a) Lag et skript som regner ut temperaturen i Fahrenheit for en gitt temperatur i Celsius.

(b) Bruk skriptet til å finne ut hvor mange grader Fahrenheit disse tempraturene er: $20°$, $0°$, $-40°$.

## Oppgave 3

(a) Skriv et skript der du lagrer et fornavn og et etternavn. Skriv ut en beskjed til personen.

(b) Skriv også ut antall tegn i fornavnet og etternavnet. For å gjøre dette skal du bruke kommandoen `len(name)` som gir lengden av en tekststreng, det vil si, antall karakter. For eksempel gir `len("Jonas")`, resultatet 5.