Introduksjon til programmering
Bjørnegård Skole

Jonas van den Brink

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

January 10, 2015

> *"Everybody in this country should learn to program a computer... because it teaches you how to think."* –Steve Jobs

Dette skrivet hører til et introduksjonskurs ved Bjørnegård skole. Over 12 timer skal jeg prøve å gi dere en introduksjon til programmering. På så kort tid kan jeg selvfølgelig ikke lære dere alt, men jeg skal forhåpentligvis få vist dere en del av hovedpoengene i programmering. Idéen er det skal bli lettere for dere å lære mer i fremtiden, og det er et av målene mine med dette kurset.

Det er ingen tvil at programmering blir mer og mer viktig. Internet er såvidt 20 år gammelt, og datamaskiner litt over 60 år gammelt. Idag er vi omringet av datamaskiner, smarttelefoner og tablets—og de blir kraftigere og kraftigere i et enormt tempo. Datamaskiner brukes mer og mer, og fler og fler jobber kommer til å innehold hvertfall en del programmering. I fremtiden er jeg veldig sikker på at programmering kommer til undervises i skolen på lik linje med matematikk.

Men det er ikke bare fordi programmering er viktig og nyttig jeg ønsker å lære dere det, det er også utrolig morsomt. Når man programmerer, så lager man noe. Programmering er en evne som lar deg gjøre om idéer og tanker du har til en virkelighet. Om du har en idé om en fantastisk app til smarttelefonen din, så kan du faktisk lage den appen. Det å programmere er rett og slett en utrolig kreavtiv ting å drive med, man kan nesten dra det såpass langt at man kan sammenligne det med å spille et instrument, eller å male eller tegne.

Men til slutt, en liten advarsel. Programmering er en helt ny ferdighet du skal lære, og det er utfordrende å skulle lære noe helt nytt. Det kommer nok til å være vanskelig og litt frustrende til tider, men forhåpentligvis også veldig belønnende når du greier å skrive dine første programmer. Dette prosjektet er altså en real utfordring til deg, og jeg håper du tar den på strak arm og står på. Jeg kommer til å være her hele veien for å hjelpe deg gjennom det, men til syvende og sist er det du som må gjøre arbeidet.

Hva er programmering?

Å programmere betyr rett og slett å lage dataprogrammer. Fra hverdagen er vi vandt med at dataprogrammer ofte er store og kompliserte, enten det er spill, dokumentbehandlingsverktøy som Microsfot Word, bildebehandlingsverktøy som Photoshop, eller internetbrowsere som Google Chrome eller Mozilla Firefox. Alle disse er eksempler på programmer, men disse er veldig store programmeringsprosjekter, hvor mange personer har samarbeidet for å lage et helhetlig produkt som skal kunne gjøre veldig masse forskjellig.

De fleste programmer som skrives er i motsetning til disse veldig små og spesialiserte. Grunnen til at dere kanskje ikke har hørt så mye om sånne programmer, er at de enten aldri har blitt delt. En som kan programmere skriver ofte et program som man bare bruker selv, og aldri deler med noen. Eller så kan det være at dataprogrammet bare er skjult fra dere. Tenk for eksempel på mikrobølgovnen deres hjemme, eller ovnen, eller vaskemaskinen, eller TVen deres—alle disse har blitt *programmert* til å utføre spesialiserte oppgaver. Tusenvis av dataprogrammer ligger i bakgrunnen av hverdagen deres for å gjøre ting så lette som mulig for dere, og det er jo nettopp det at dere slipper å tenke på dem, som gjør dem så geniale.

Programmeringsspråk

Å programmere handler altså om å gi datamaskinen instrukser for at den skal utføre en bestemt oppgave, eller løse et bestemt problem. Vi gir disse instruksene til datamaskinen ved hjelp av spesielle programmeringsspråk, som rett og slett er et språk datamaskinen forstår. Eksempler på programmeringsspråk som brukes mye idag er Python, Java, C++ og JavaScript. Akkurat som vanlig språk, så finnes det mange hundre forskjellige programmeringsspråk, og de har mange fellestrekk. I motsetning til vanlig språk derimot, så er det mye lettere å lære seg flere programmeringsspråk hvis man allerede kan ett. I dette kurset kommer vi til å bruke Python, som er et av de aller mest brukte programmeringsspråkene idag—hvis dere har et godt grep om Python, så kan dere ganske lett lære dere de fleste andre programmeringsspråk på kort tid.

Problemløsning

Når vi skriver instrukser i form av programmeringsspråk, så kaller vi det kode. Når vi lager et dataprogram, så handler altså om å skrive den riktige koden, sånn at datamaskinen gjør det vi vil at den skal gjøre. Det som gjør programmering vanskelig, er at datamaskiner er veldig dumme. I motsetning til oss mennesker, kan datamaskiner kun forstå veldig enkle instrukser. Før vi kan sette igang med å skrive selve koden for programmet vår, må vi altså finne ut hvordan vi kan løse oppgaven vår med de enkle instruksene datamaskinen kan forstå. Vi må *bryte ned* problemet vårt til små, håndterlige biter som datamaskinen kan få til.

Når vi bryter opp et problem på denne måten, så lager vi en slags oppskrift for hvordan et problem kan løses. Du kan tenke på det litt som et mattestykke, for å komme i mål for å løse en matteoppgave, så man første gjøre en liten ting, og så en annen liten, og så en tredje liten ting, helt til man sitter med det siste

svarer til slutt. Hvert steg i seg selv er aldri særlig vansklig, utfordringen ligger i å finne ut hvilke steg man skal ta, og i hvilken rekkefølge man skal gjøre dem. I programmering så kaller man gjerne en slik 'oppskrift' for en algoritme. Det er litt rart ord, men en algoritme er altså egentlig bare oppskriften av hvilke steg man skal ta for å komme i mål.

Datamaskiner er dumme

Siden datamaskiner er så dumme, krever det også at koden vi skriver er helt riktig. Hvis du skriver et brev til en venn, og du har et par skrivefeil i teksten din, så er ikke det så farlig, for vennen din kommer mest sannsynlig til å skjønne det du har skrevet uten problemer, kanskje vennen din ikke engang legger merke til skrivefeilene dine. Med en datamaskin derimot, så vil en enkel skrivefeil være totalt uforståelig, og mest sannsynlig vil ikke dataprogrammet ditt funke i det heletatt hvis du har en skrivefeil i koden din.

Det hørest kanskje utrolig vanskelig ut, at en skrivefeil skal ødelegge hele programmet ditt. Hvordan kan du klare å skrive kode uten å ha noen småfeil? Vel, jeg kan love deg med en gang at du ikke kommer til å få til det. Ingen får til det, ikke engang de som har skrevet kode i mange, mange år. Det er derfor en stor del av det å programmere, er å finne feilene man har gjort, å rette på dem. Småfeil i kode kalles ofte for *bugs*, og det å lete etter feil og rette dem opp kaller vi derfor for *bugfixing*. Bugfixing er en viktig del av programmering, og er mye av det man bruker tiden sin på, så det er verdt å bli flink på det. Heldigvis er datamaskinen flink til å si ifra om hva det er den ikke forstår, så bugfixing kan faktisk være ganske lett og gøy, men la oss se mer på dette når du faktisk har lært litt kode.

Lightbot

Så langt har jeg jo fortalt en del om hvordan programmering funker, men det så langt har jo alt vært veldig generelt og vagt. La oss derfor snu oss til et eksempel på programmering, et spill som heter Lightbot. Jeg syns dette spillet er en flott måte å se nærmere på prinsippene i programmering, uten at vi trenger å sette oss inn i et helt programmeringsspråk helt enda.

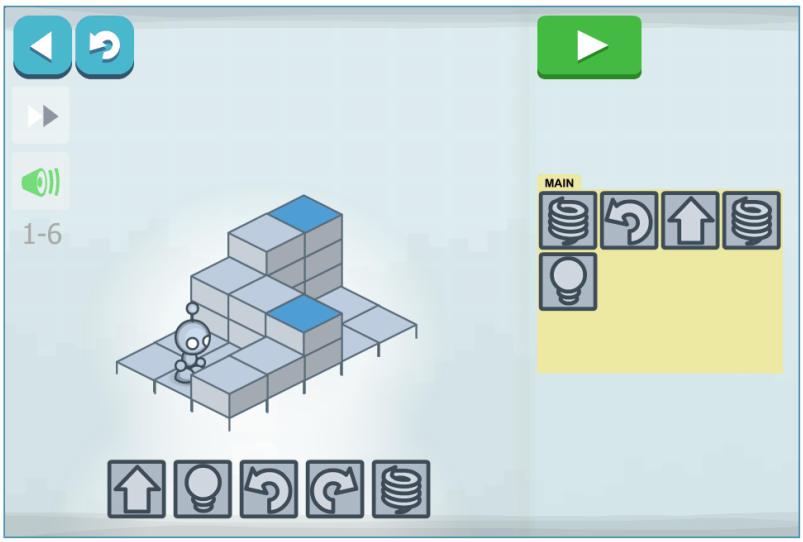
Du kan spille det gjennom internetbrowseren din ved å gå til denne linken:

`http://lightbot.com/hocflash.html`

Spillet er også tilgjengelig til smarttelefoner og tablets, både som en gratisapp og en lengre versjon som koster litt penger. Bare søk etter 'Lightbot' i din appstore, og last det ned.

Hvordan spiller du?

Under er et bilde fra spillet. Målet er å programmere den lille roboten sånn at alle de blå feltene på brettet lyser opp. De små ikonene på bunn av skjermen er de tilgjengelige kommandoene eller instruksene du kan bruke. På høyre siden av skjermen, i boksen som heter 'main', er selve programmet du lager. Du legger til instrukser i programmet ved å trykke på ikonene på bunn av skjermen. Hvis du er misfornøyd med det du har laget så langt kan du fjerne kommandoer fra programmet ved å trykke på dem. Du kan og enkelt bytte på rekkefølgen av instruksene ved å dra dem frem og tilbake.



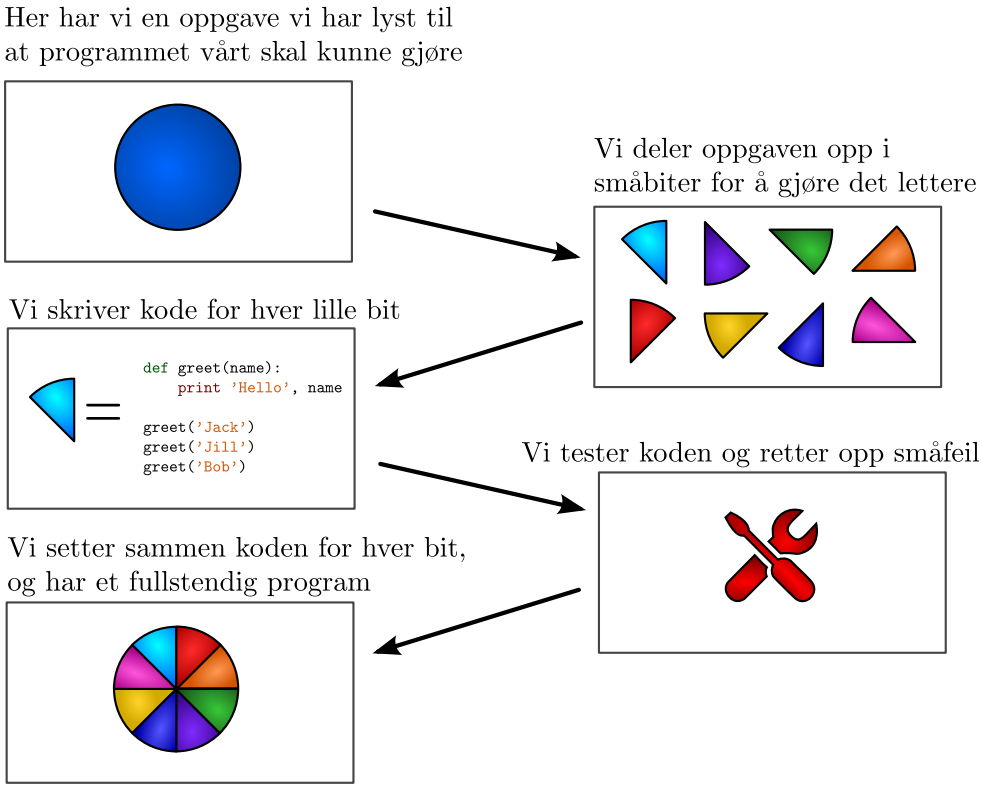
Når du er fornøyd med programmet du har laget til roboten trykker du på den grønne pilen på toppen av skjermen, da kjøres programmet. At programmet ditt kjøres, betyr at alle kommandoene du har lagt inn utføres i rekkefølge. Du kan nå følge med på hva roboten gjør, for å se om programmet ditt gjør som du ville. Enten greier roboten å lyse opp alle de blå feltene, isåfall har du greid brettet. Eller så mislykkes roboten, det gjør ingenting, da trykker du bare på den oransje knappen for å resette roboten, og du kan så gjøre endringer til programmet før du prøver å kjøre det igjen.

Etter du har spillt igjennom de fire, fem første brettene, begynner du nok å få teken på hvordan du skal programmere roboten til å gjøre det du vil. Det du gjør når du spiller dette spillet, er faktisk programmering. For å være mer nøyaktig, så er spillet et eksempelt på noe vi kaller *symbolsk programmering*, fordi du bruker symboler, eller ikoner, for å lage programmet ditt, istedenfor kode. Men ellers så er dette akkurat det samme som vi kommer til å gjøre når du skal lære å skrive Python.

For hvert brett så har du en konkret oppgave du har lyst til å løse, du har lyst til å lyse opp alle de blå feltene på brettet. For å klare det er det en del hindre du må komme deg forbi. Du starter å løse hvert brett ved å finne ut hvilke kommandoer du må bruke for å komme deg forbi hindrene og lyse opp feltene. Etter du har skjønnt hvordan du skal løse brettet, så 'koder' du opp programmet ditt ved å velge de riktige instruksene. Når du har gjort dette, så tester du koden din ved å kjøre programmet. Hvis det er feil i koden, så retter du dem opp. Slik fortsetter du, helt til du har et helt program som løser hele brettet. Vi har nå oppsummert hva det vil si å programmere.

Oppsummering av programmeringstankegangen

For å oppsummere programmeringstankegangen, så har jeg laget en liten figur. Ta en titt på den og se om du greier å knytte opp spillet Lightbot til de forskjellige stegene.

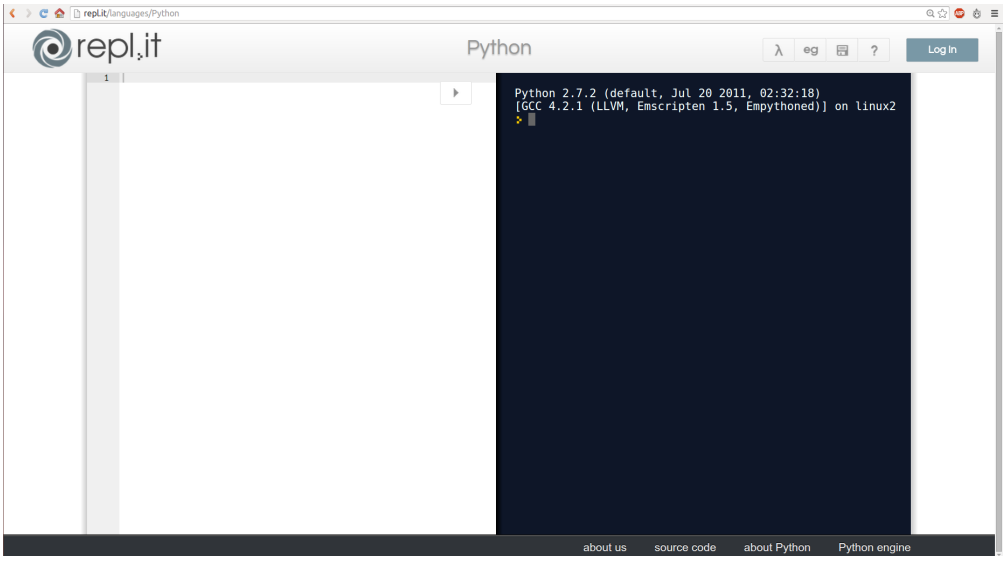


Pythonprogrammering

Nå som vi har fått inn grunntanken med programmering, la oss begynne å se på Python. Når dere skal programmere i Python så trenger dere en data hvor python er installert. Akkurat nå tar vi oss ikke tid til å installere det vi trenger, så vi velger å bruke en løsning på internet. Hvis vi går inn på nettsiden

`repl.it`

så kan du velge mellom mange forskjellige programmeringsspråk. Klikk på Python, og vi kommer inn i noe vi kaller en Python interpreter, der vi kan skrive og kjøre Python-kode. Nettsiden skal nå se ligne på dette



På venstre side av skjermen, i det hvite vinduet, er der vi skriver selve programmet vårt, vi kaller gjerne dette vinduet for editoren. I editoren skriver du koden din. På høyre side av skjermen, i det mørke-blå vinduet, er det resultatet av kjøringen din kommer. Vi kaller det blå vinduet for terminalen. Når du vil kjøre programmet du har skrevet i det hvite vinduet, så kan du trykke på pilen du ser øverst til høyre i det hvite vinduet (ca midt på skjermen), da kjøres programmet ditt. Når programmet kjøres, så utfører Python koden du har skrevet, linje for linje, fra toppen ned. Resultater fra kjøringen skrives ut i terminalen.

Ditt første Pythonprogram

Nå er det på tide at du skal skrive ditt første Pythonprogram. Vi lar det være et utrolig enkelt program, kun bestående av linjen

```
print 'Hello, world!'
```

Skriv inn koden i editoren (det hvite vinduet), og trykk på pilen for å kjøre programmet ditt. Hva er det som skjer? Forhåpentligvis så vises skriften *Hello, world!* i terminalen (det blå vinduet). La oss ta en titt på hva som har skjedd. Kodelinjen du har skrevet er en `print`-kommando. I Python så er `print` et nøkkelord som betyr at noe skal vises (skrives ut) til terminalen. Det koden vår gjør er altså å bruke `print`-kommandoen til å skrive ut en beskjed til terminalen.

Merk at i koden så har vi omringet beskjeden vi har lyst til å skrive ut med fnutter, `''`, dette er fordi Python skal kunne skille mellom teksten i beskjeden og resten av koden. Vi kaller teksten i fnutter for en tekststreng, og tekststrenger tolkes aldri av Python som kode, den behandles bare som en tekst. Hvis du ser på terminalen, så ser du at Python ikke har inkludert fnuttene i det den har skrevet ut, det er det som er mellom fnuttene som er viktig. Merk at hele tekststrenger har blitt farget en egen farge i editoren, sånn at det er lett å skille mellom teksten som er kode, og teksten som ikke er kode. Så lenge vi skriver beskjeden vi vil skrive ut som en tekststreng, det vil si mellom de to fnuttene, så kan vi skrive hvilken som helst tekst vi ønsker. Prøv selv å endre beskjeden et par ganger og se at alt fungerer som det skal.

La oss nå prøve å ha to printkommandoer etter hverandre

```
print "Hva skjera Baghera?"
print "Ingenting tingeling!"
```

Hva skjer når du kjører programmet nå? Hver av linjene i programmet er en `print`-kommando som skriver ut en tekststreng. Python tolker koden linje for linje nedover, så den vil først behandle den ene print-kommandoen, og så den andre. Utskriftene til Terminalen havner etter hverandre, på hver sin linje.

Variable

Når vi skriver et dataprogram, så er det viktig at datamaskinen skal kunne huske på ting den trenger å vite. Vi får datamaskinen til å huske på ting ved å definere noe vi kaller variabler. Vi kan tenke på en variabel som en boks. La meg vise deg et eksempel, her er to linjer kode

```
name = "Jonas"
alder = 23
```

Hva er det denne koden gjør? Den første kodelinjen sier `name = Jonas`. Det jeg gjør med denne kommandoen er å fortelle Python navnet mitt. Mer teknisk kan jeg si at jeg lager en variabel som heter `name`, og den inneholder navnet mitt, som er teksten 'Jonas'. Det Python gjør når den behandler denne kodelinjen, er at den oppretter en variabel, tenk på det som en tom boks, som heter `name`. Så putten den teksten 'Jonas' inn i den boksen, så setter den boksen i arkivet. Tilsvarende sier den neste kodelinjen at det skal være en variabel som heter `age`, og der ligger tallet 23.

Prøv å skriv et lignende program for deg selv, med ditt navn og din alder. Kjør programmet, hva skjer? Ingenting skjer, ihvertfall ikke som du kan se. Terminalen er helt tom. Python oppretter variablene, akkurat slik vi spør om, men den viser ikke i terminalen at den har gjort noe. For å få noe til å skje, så kan vi legge til en `print`-kommando i bunn av programmet vårt

```
print name
```

Hvis du kjører programmet nå, så ser du at navnet ditt skrives ut i terminalen. Det er fordi du ber Python skrive ut `name`, det som skjer da er at Python sjekker om den har en variabel (en boks) som heter name, når den finner den riktige variabelen i minnet sitt (den riktige boksen fra arkivet), så sjekker den hva

innholdet er, og skriver ut innholdet til terminalen. På samme måte kan vi skrive ut variabelen `age` til terminalen med kommandoen

```
print age
```

Det du kan prøve på nå, er å skrive

```
print 'name'
```

eller

```
print 'age'
```

Hvis du gjør det, så ser du at du *ikke* får skrevet ut navnet eller alderen din til terminalen. Det er fordi Python tolker det du prøver å skrive ut som tekststrenger, fordi du har skrevet dem med fnutter på sidene.

Merk at hvis du prøver å opprette to variabler med samme name, så overskrives rett og slett den originale variabelen. Prøv for eksempel programmet

```
name = "Marius"
name = "Lise"

print name
```

Hva forventer du at skjer? Prøv selv!

Typer

Du har nå sett at vi kan lage variable. Variabler har altså navn og innhold, men de har også en *type*. Hvis vi ser på de to variablene vi nettopp lagde, så har vi `name`, som inneholder en tekststreng, og `alder` som inneholder et tall. Når Python oppretter en variabel, så husker den ikke bare på navnet og innholdet, men den merker seg også hva slags innhold variabelen har.

Vi kan bruke kommandoen `type` til å sjekke hva slags type en variabel har. Her er et eksempelprogram

```
location = "Oslo"
year = 2015
day = "4. januar"
temperature = -7.3

print type(location)
print type(year)
print type(day)
print type(temperature)
```

I dette programmet oppretter jeg først fire variable, så skriver jeg ut til terminalen hvilken type de har. Merk at kommandoen `type` sjekker typen til det du skriver inne i parantesen, sånn at `type(sted)` betyr typen til variabelen `navn`. Når du kjører programmet får du denne utskriften i terminalen

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

Vi ser altså at variabelen `location` har type 'str', dette er en forkortelse for 'string', og betyr altså tekststreng. Dette stemmer jo godt, fordi variabelen `location` inneholder jo en tekststreng. Variabelen `year` har type 'int' som er en forkortelse for 'integer'. Integer er engelsk og betyr heltall. Vi ser at `day` også er en 'str', altså tekstreng. Til sist har vi `temperature` som har typen 'float', som betyr flyttall. Et flyttall er egentlig bare et annet navn på et desimaltall. Vi ser altså at Python skiller på heltall og desimaltall.

Feilmeldinger

Nå som du har begynnt å skrive din første Pythonkode kan det jo være at det har oppstått noen feil, og hvis du har gjort alt rett så langt, så dukker det opp noen feil snart. La oss ta en titt på feilmeldingene vi får når vi gjør feil i Python, og prøve å tolke dem litt. Når du programmerer kommer du til å gjøre mange feil og det er viktig å prøve å forstå hvorfor det gikk galt, det å tolke sine egne feilmeldinger er nok den aller beste måten å bli god til å programmere.

La oss skrive en `print`-kommando feil med vilje

```
prnt "Hello, World!"
```

Når du prøver å kjøre programmet nå, så får du en feilmelding ut. Den ser ut noe som dette:

```
File "<stdin>", line 1
    prnt "Hello, World!"
                       ^
SyntaxError: invalid syntax
```

Det er alltid den nederste linjen i en feilmelding som er den viktigste, og der står det `SyntaxError: invalid syntax`. Feilen vi har gjort er altså en *syntaks*-feil. En syntaks-feil betyr at Python ikke skjønner det vi har skrever, vi har skrevet noe som rett og slett ikke gir mening. Når du får en syntaks-feil bør du altså sjekke at du har skrevet alt riktig, sånn som her ser vi at `print`-kommandoen er skrevet feil.

På linjene over prøver Python å informere oss hvor feilen er. Det står 'line 1' på toppen, akkurat nå er det jo åpenbart at feilen må være i kodelinje 1, for det er alt vi har skrevet! Men i et program på flere hundre kodelinjer er det veldig nyttig å få vite hvilken linje feilen er på.

La oss prøve en annen feil

```
location = "Oslo"
print place
```

Hvis du kjører dette programmet får du feilmeldingen

```
Traceback (most recent call last):
File "<stdin>", line 2, in <module>
NameError: name 'place' is not defined
```

Nå har du ikke lenger en syntaks-feil, fordi Python skjønner godt hva du har lyst til å gjøre her, det du har skrevet er helt riktig Python-kode. Problemet er derimot en `NameError`, det er ganske enkelt fordi programmet først oppretter

variabelen 'location', og prøver deretter å printe ut variabelen 'place'. Men det finnes ingen variabel med navn 'place', så derfor får du en navn-feil—programmet prøver å bruke en variabel som ikke finnes.

Mer om printing

Så langt har du sett at du kan skrive ut både tekststrenger og variable, la oss nå kombinere dem. Ta en titt på følgende program

```
name = "Silje"
print "Hei", name, "! Hvordan har du det idag?"
```

Her bruker vi `print`-kommandoen til å skrive ut 3 ting etterhverandre. Hvis du kjører programmet så ser du at de tre tingene vi skriver ut havner på samme linje, ikke hver sin linje, det er fordi de alle hører til samme `print`-kommando.

Hvis du ser nærmere på utskriften, så kan du legge merke til at Python har lagt inn et mellomrom mellom hver av tingene vi skriver ut, så i terminalen står det "Hei Silje ! Hvordan har du det idag?". Det er jo litt dumt at det er et ekstra mellomrom mellom 'Silje' og '!', så la oss se på en annen måte vi kan skrive ut en beskjed og en variabel samtidig.

```
name = "Silje"
print "Hei %s! Hvordan har du det idag?" % name
```

Prøv å kjør dette programmet? Nå ble beskjeden akkurat slik vi ønsket. Men hva er det vi egentlig har gjort her? Vi ser at vi prøver å skrive ut en tekststreng, men inne i tekstrengen står det `%s`. Når vi skriver `%s` inne i en tekststreng, så lager vi et slags hull i strengen, der vi kan fylle inn en variabel. Vi skriver `% name` bak tekstrengen, fordi det er denne variabelen vi ønsker å fylle inn i hullet. Grunnen til at vi skriver `%s` i strengen er fordi `s` står for streng, siden vi fyller inn med en variabel som er en tekststreng.

Vi kan lage så mange hull i en tekst som vi ønsker, her er et eksempel

```
name = "Silje"
age = 18
location = "Drammen"

print "Jeg heter %s, er %i og kommer fra %s." % (name, age, location)
```

Nå ser vi at det er 3 hull i teksten, det er to strenger merket med `%s` og et heltall merket med `%i` (husk at integer betyr heltall på engelsk). Bak strengen har vi listet opp tre variable som skal fylles inn i teksten. Merk at vi har satt parantes rundt dem, og at de kommer i samme rekkefølge som hullene i teksten.

Dataprogrammer som snakker med brukeren

Så lang har vi bare laget programmer som gjør noe enkelt og slutter helt av seg selv. Men de fleste programmer dere bruker i hverdagen er jo laget for å ha en interaksjon med brukeren. La oss derfor stille brukeren av programmet noen spørsmål, det kan vi gjøre med kommandoen `raw_input`. Her er et eksempel

```
weather = raw_input('Hi! How is the weather today?')
print "The weather seems to be %s today!" % weather
```

Når Python utfører denne kodelinja, så skrives spørsmålet i parantesen ut i terminalen, og så venter den på at brukeren som har kjørt programmet skal skrive inn et svar. Prøv å kjør programmet, skriv inn et svar på spørsmålet og trykk enter for å fortsette videre i programmet. Det du (brukeren) svarer på spørsmålet blir så lagret i variablen `weather`. Etter du har trykket enter, fortsetter programmet. I vårt tilfelle går den da videre til å skrive ut en beskjed, hvor svaret brukeren ga brukes.

Oppsummering uke 1

- Å programmere betyr å lage et dataprogram. Dette gjør vi ved å skrive kode i et bestemt programmeringsspråk. Før vi kan gå igang med å kode må vi ofte bryte ned oppgaven vi skal løse i små biter.
- Datamaskinen husker ting i form av *variabler*, vi kan opprette variabler ved å gi dem navn og et innhold. Python gir også variabelen en type, som vi kan sjekke ved å bruke kommandoen `type()`.
- For å skrive ut noe til terminalen kan vi bruke `print`-kommandoen, vi kan skrive ut både variabler og tekststrenger.
- Vi kan skrive ut flere ting etterhverandre hvis vi skiller dem med komman: `print ting1, ting2`, eller vi kan skrive ut tekster som vi fyller inn med variabler: `print "Hei, jeg heter %s" % name`.
- Vi kan spørre brukeren et spørsmål med kommandoen `raw_input()`, i parantesen skriver du spørsmålet som en tekststreng.
- Det er lett å gjøre feil i programmering, men det gjør ingenting. Når vi kjører et program med feil i, får vi en feilmelding som prøver å fortelle oss hva som har gått galt.

Oppgaver til uke 1

Oppgave 1 — Printing

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

(b) Spør brukeren om navnet dems, og deretter skriv en beskjed tilbake som bruker navnet de har gitt.
Hint: Du kan spørre brukeren et spørsmål med `raw_input`-kommandoen.

Oppgave 2 — Adjektivhistorie

(a) Spør brukeren om 5 adjektiv og lagre dem i 5 forskjellige variable.
Hint: Adjektiv er beskrivende ord som for eksempel 'liten' og 'stor'.

(b) Test programmet ditt ved å skrive ut alle 5 adjektivene i terminalen.

(c) Skriv en adjektivhistorie og print den slik at de 5 adjektivene brukeren har gitt fylles inn i historien.

(d) Test programmet ditt.

(e) Hvis alt funker som det skal kan du nå gjemme editoren din, kjør programmet og få en venn til å fylle ut adjektivene.

Oppgave 3 — Lightbot

(a) Spill igjennom alle brettene.

(b) **Utfordring**: Klarer du å løse brett 2–1 med bare 7 kommandoer? Hva med brett 2–2?

(c) **Utfordring**: Hvor få kommandoer klarer du å bruke for å løse 2-6? Det beste jeg har fått til er 15 kommandoer!