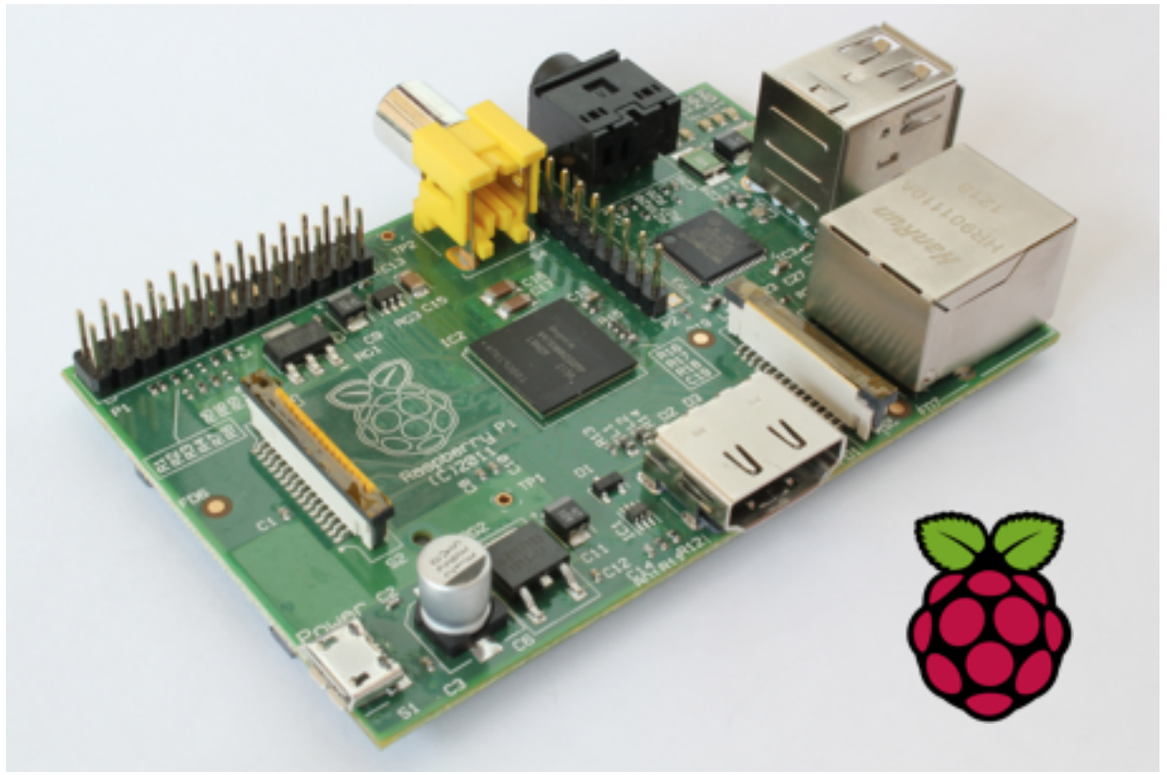
Lær Kidsa Koding

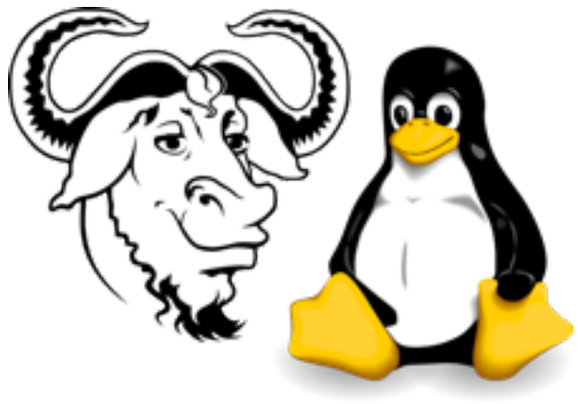
Olve Maudal, Cisco Systems



6-timers introduksjonskurs i programmering
for 24 barn/ungdom i alderen 9-16 år

Cisco Systems, Lysaker
Fredag 21. januar 2014

I dag vil du lære litt om:



GNU/Linux

Agenda

0930 Velkommen, dele opp i grupper
0945 Bli kjent med Raspberry Pi og GNU/Linux
1000 Introduksjon til C++
1005 Øvelser
1145 Lunsj
1230 Introduksjon til Python
1235 Øvelser
1415 Pause
1445 Øvelser
1530 Slutt

Deltagere

Brage, 11 år/6.
Oscar, 11 år/6.

Henrik, 9 år
Samuel, 10 år

Oscar, 10 år/4.
Einar, 11 år/6.

Ingvild, 14 år/9.
Amanda, 15 år/10.

Bjørn Are, 16 år/vgs
Philip, 16 år/vgs

Sander, 15 år
Håkon, 15 år/10.

Hermine, 11 år
Maja, 10 år/5.

Markus, 12 år/7.
Sander, 11 år/6.

Leon, 13 år/8.
Marius, 13 år/8.

Iver, 10 år/5.
Jonathan, 11 år/6.

Andreas, 13 år
Jens, 12 år/7.

Louise, 13 år/8.
Kathinka, 13 år/8.

Bli kjent med Raspberry Pi og GNU/Linux (obligatorisk)

1) Start maskinen

2) Logg inn på maskinen (bruker: `pi`, passord: `raspberry`)

3) Se på alle filene i nåværende katalog

```
pwd
ls
```

4) Skift ned til underkatalogen python_games, og list alle filene som slutter på `.py`

```
cd python_games
pwd
ls *.py
```

5) Start spillet Wormy

```
python wormy.py
```

6) Se på kildekoden, prøv å forstå noe av det som står der

```
nano -v wormy.py
```

6) Gå tilbake til din hjemmekatalog

```
cd
pwd
ls -al
```

7) Start the grafiske brukergrensesnittet

```
startx
```

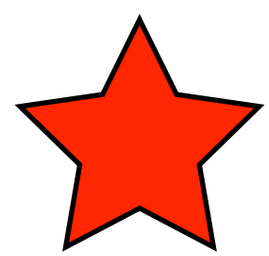
8) Start LXTerminal og skriv inn

```
ls
exit
```

9) Avslutt det grafiske brukergrensesnittet (trykk Ctrl-Alt-Backspace samtidig)

10) Restarte maskinen

```
sudo reboot
```



Introduksjon til C++

Hello (obligatorisk)

Logg inn på maskinen (pi/raspberry).

Gå til hjemmeområdet ditt:

```
cd
```

Lag en ny programfil med en teksteditor:

```
nano hello.cpp
```

og så skriver du inn programmet som du ser til høyre.

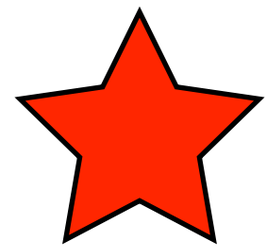
```
#include <iostream>

int main()
{
    std::cout << "Hello!" << std::endl;
    return 0;
}
```

Når du er ferdig med å skrive inn programmet kompilerer du det med en C++ kompilator:

```
g++ -o hello hello.cpp
```

og så kjører du programmet slik:

```
./hello
```



Hello, del 2

Logg inn på maskinen (pi/raspberry).

Gå til hjemmeområdet ditt:

```
cd
```

Lag en kopi av den gamle filen, og editer kopien med en teksteditor:

```
cp hello.cpp hello2.cpp
nano hello2.cpp
```

og så skriver du inn programmet som du ser nedenfor.

```
#include <iostream>

int main()
{
    std::string name;
    std::cout << "What is your name? " << std::flush;
    std::cin >> name;
    int i = 0;
    while (i < 10) {
        std::cout << "Hello " << name << "!" << std::endl;
        i = i + 1;
    }
    return 0;
}
```

Kompiler og kjør:

```
g++ -o hello2 hello2.cpp
./hello2
```

MyPaint, del 1 (obligatorisk)

Dette er et lite program som tegner to streker på skjermen.

Gå til hjemmeområdet ditt:

```
cd
```

Lag en ny programfil med en teksteditor:

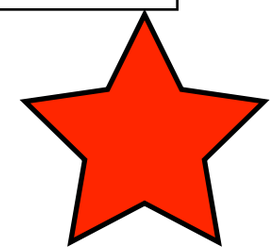
```
nano mypaint1.cpp
```

og så skriver du inn programmet, kompilerer og kjører:

```
g++ -lncurses -o mypaint1 mypaint1.cpp
./mypaint1
```

Klarer du å tegne noe som ligner på et hus?

```
#include <ncurses.h>

int main()
{
    initscr();
    noecho();
    curs_set(0);

    mvprintw(0,10,"My Paint");

    int r = 10;
    int c = 12;

    while (c < 30) {
        mvaddch(r,c,'X');
        c++;
    }

    while (r < 20) {
        mvaddch(r,c,'X');
        c--;
        r++;
    }

    getch();

    endwin();

    return 0;
}
```



MyPaint, del 2

Gjenbruk koden fra del 1:

```
cp mypaint1.cpp mypaint2.cpp
nano mypaint2.cpp
```

Endre slik at koden blir som den til høyre.

Kompiler og kjør:

```
g++ -lncurses -o mypaint2 mypaint2.cpp
./mypaint2
```

```
#include <ncurses.h>

int main()
{
    initscr();
    noecho();
    curs_set(0);

    mvprintw(0,10,"My Paint");

    int r = 10;
    int c = 12;
    bool quit = false;
    while (!quit) {
        mvaddch(r,c,'X');
        int ch = getch();
        if (ch == 'a')
            c--;
        else if (ch == 'd')
            c++;
        else if (ch == 'w')
            r--;
        else if (ch == 's')
            r++;
        else if (ch == 'q')
            quit = true;
    }

    endwin();

    return 0;
}
```

MyPaint, del 3

Gjenbruk gjerne programmet fra del 2.

Skriv inn og kjør programmet til høyre. Dette er et interaktivt program hvor du styrer "musen" med `a`,`d`,`w`,`s` og så setter du ned eller løfter pennen med `p`. `q` for å avslutte.

Legg merke til hvordan vi bruke switch isteden for mange if-else. Hva skjer hvis du glemmer å skrive break, feks etter `case 'd'`?

mypaint3.cpp

```
#include <ncurses.h>

int main()
{
    initscr();
    noecho();
    curs_set(0);

    mvprintw(0,10,"My Paint");

    int r = 10;
    int c = 12;
    bool paint = false;
    bool quit = false;
    while (!quit) {
        mvaddch(r,c,'X');
        int ch = getch();
        if (!paint)
            mvaddch(r,c,' ');
        switch (ch) {
        case 'a': c--; break;
        case 'd': c++; break;
        case 'w': r--; break;
        case 's': r++; break;
        case 'q': quit = true; break;
        case 'p': paint = !paint; break;
        }
    }

    endwin();

    return 0;
}
```

Mathquiz

Logg inn på maskinen (pi/raspberry).

Gå til hjemmeområdet ditt:

```
cd
```

Lag en ny fil og skriv inn programmet til høyre::

```
nano mathquiz.cpp
```

Kompiler og kjør:

```
g++ -o mathquiz mathquiz.cpp
./mathquiz
```

Gå igjennom hver eneste linje sammen med partneren og forklar høyt hva som skjer. Hvis du skjønner dette programmet så kan du allerede ganske mye C++.

```
#include <iostream>
#include <cstdlib>
#include <ctime>

int main()
{
    std::time_t start = std::time(0);
    std::srand(start);

    int count = 10;
    int errors = 0;
    double penaltytime = 0;

    std::cout << "Try " << count << " solves:" << std::endl;

    for (int i=0; i<count; ++i) {
        int a = std::rand() % 5 + 2;
        int b = std::rand() % 5 + 2;
        int expected = a * b;
        bool correct = false;
        while (!correct) {
            std::cout << a << " * " << b << " = " << std::flush;
            std::string inputstr;
            std::cin >> inputstr;
            int input = std::atoi(inputstr.c_str());
            if (input == expected) {
                std::cout << "Bravo!" << std::endl;
                correct = true;
            } else {
                std::cout << "Not correct!" << std::endl;
                penaltytime += 5.0;
                errors++;
            }
        }
    }

    std::time_t finished = std::time(0);
    double seconds = std::difftime(finished, start);
    double avgtime = (seconds + penaltytime) / count;

    std::cout << "Total time: " << seconds << std::endl;
    std::cout << "Penalty time: " << penaltytime << std::endl;
    std::cout << "Avg time per solve: " << avgtime << std::endl;

    if (errors == 0 && avgtime < 4.0)
        std::cout << "You are a genius!" << std::endl;
    else if (avgtime < 5.0)
        std::cout << "Respectable score!" << std::endl;
    else
        std::cout << "You need to practice more" << std::endl;
}
```

Calc, del 1

Alle større programmer består av flere filer som oversettes til maskinspråk hver for seg (slik at vi får en .o objektfil) og så lenkes disse objektfilene sammen med en såkalt linker slik at vi får et kjørbart program. Her får du også se et enkelt eksempel på objekt-orientert programmering i C++. Skriv inn filene under, og så kjører du disse kommandoene:

```
g++ -c my_calculator.cpp
g++ -c calc1.cpp
g++ -o calc1 my_calculator.o calc1.o
./calc1
```

Hva er svaret?

my_calculator.hpp

```
#include <stack>

class my_calculator {
public:
    void push(double value);
    double pop();
    double top() const;
    void add();
    void sub();
private:
    std::stack<double> stack;
};
```

calc1.cpp

```
#include "my_calculator.hpp"
#include <iostream>

int main()
{
    my_calculator calc;

    calc.push(35);
    calc.push(9);
    calc.add();
    calc.push(2);
    calc.sub();
    std::cout << calc.top() << std::endl;
}
```

my_calculator.cpp

```
#include "my_calculator.hpp"
#include <stdexcept>

void my_calculator::push(double value)
{
    stack.push(value);
}

double my_calculator::top() const
{
    if (stack.empty())
        throw std::runtime_error("stack error");
    return stack.top();
}

double my_calculator::pop()
{
    double value = top();
    stack.pop();
    return value;
}

void my_calculator::add()
{
    push( pop() + pop() );
}

void my_calculator::sub()
{
    double op1 = pop();
    double op2 = pop();
    push( op2 - op1 );
}
```

Calc, del 2

Nå skal vi lage et program med et brukegrensesnitt ved å gjenbruke modulen som vi laget i del 1.

Skriv inn filen til høyre og link med objektfilen my_calculator.o:

```
g++ -c my_calculator.cpp
g++ -c calc2.cpp
g++ -o calc2 my_calculator.o calc2.o
./calc2
```

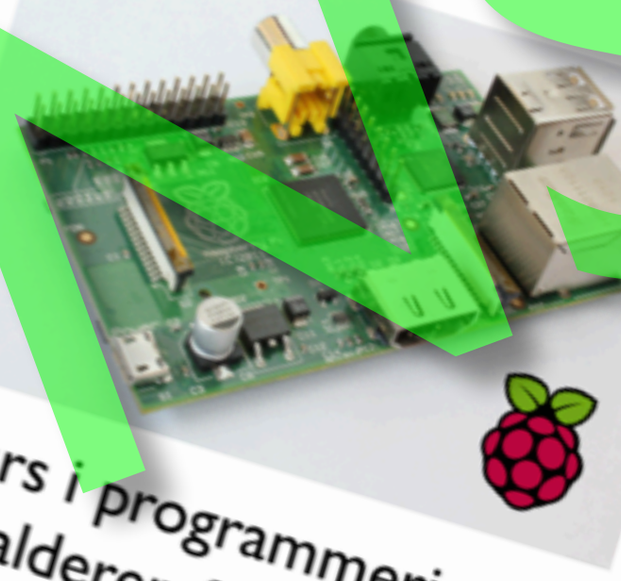
Prøv å forbedre programmet slik at det også håndterer multiplikasjon og divisjon. Da blir du nok nødt til å endre alle tre filene.

calc2.cpp

```
#include "my_calculator.hpp"

#include <iostream>
#include <sstream>
#include <stdexcept>
#include <stdlib.h>

double to_double(const std::string & str)
{
    std::istringstream ss(str);
    double d;
    ss >> d;
    if ( !(ss >> std::ws).eof() )
        throw std::invalid_argument("huh?");
    return d;
}

int main()
{
    my_calculator calc;
    std::string input;
    while (std::cin >> input) {
        try {
            if (input == "q")
                exit(0);
            if (input == "p")
                std::cout << calc.top() << std::endl;
            else if (input == "+")
                calc.add();
            else if (input == "-")
                calc.sub();
            else
                calc.push(to_double(input));
        } catch(std::exception & ex) {
            std::cout << ex.what() << std::endl;
        }
    }
    return 0;
}
```

Lær Kidsa Koding

Olve Maudal, Cisco Systems



6-timers introduksjonskurs i programmering
for 24 barn/ungdom i alderen 9-16 år

Cisco Systems, Lysaker
Fredag 21. januar 2014

Introduksjon til Python

Spillhacking (obligatorisk)

1) Start maskinen, se på hva som skrives ut på skjermen

2) Logg inn på maskinen (bruker: `pi`, passord: `raspberry`)

3) Se på alle filene i nåværende katalog

```
pwd
ls -al
```

4) Skift ned til underkatalogen python_games, og list alle filene som slutter på `.py`

```
cd python_games
pwd
ls -al
ls *.py
```

5) Start spillet Wormy, og se på kildekoden

```
python wormy.py
nano -v wormy.py
```

6) Sjekk ut de andre spillene i denne katalogen, feks

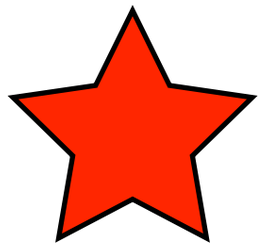
```
ls *.py
python squirrel.py
```

7) Gjør en veldig enkel endring i kildekoden til Wormy, feks endre FPS til 5

```
nano wormy.py
python wormy.py
```

8) Prøv å gjøre større endringer i Wormy-spillet, feks

   a) skriver navnet ditt når den starter (*)
   b) gjør slik at du ikke kan krasje i deg selv (*)
   c) gi deg selv ekstra mye poeng for hvert eple ormen spiser (**)



Hva heter du? (*)

Logg inn på maskinen (pi/raspberry).

Gå til hjemmeområdet ditt:

```
cd
```

Lag en ny programfil med en teksteditor:

```
nano hvaheterdu.py
```

og så skriver du inn programmet som du ser til høyre. Linjene som starter med # trenger du ikke å skrive inn, det er bare kommentarer til programkoden som maskinen ikke skal bry seg om.

```
print "Hva heter du?",

name = raw_input()

i = 0
while i < 10:
    print "Hei", name, "!"
    i = i + 1
```

Når du er ferdig kjører du programmet slik:

```
python hvaheterdu.py
```

Oppgaver:

a) skriv inn og kjør programmet. Hvor mange ganger skriver den ut navnet ditt?

b) Hvor mange ganger skriver den ut navnet hvis du endrer siste linje til:

```
i = i + 3
```

c) Hva skjer hvis du fjerner siste linje? (hint: trykk CTRL-C for å avslutte programmet)

Antalldager (**)

Her er et program som regner ut hvilken ukedag du er født og hvor mange dager siden det er.

Lag en ny programfil med en teksteditor:

```
nano antalldager.py
```

og så skriver du inn programmet som du ser til høyre. Linjene som starter med # trenger du ikke å skrive inn, det er bare kommentarer til programkoden som maskinen ikke skal bry seg om.

Når du er ferdig kjører du programmet slik:

```
python antalldager.py
```

Oppgaver:

a) Skriv inn og kjør programmet. Hvor mange dager er du?

b) Det ligger en test for ugyldig årstall. Legg inn tilsvarende tester for måned og dag.

c) Klarer du å lage en versjon hvor du kan skrive inn månedsnavn istedenfor 1-12?

```
# last inn biblioteket datetime slik at vi kan regne på datoer
import datetime

# spør brukeren om navn og fødselsdag
name = raw_input("Hva heter du? ")
year = int(raw_input("Hvilket år ble du født (1900-2010)? "))
month = int(raw_input("Hvilken måned har du bursdag (1-12)? "))
day = int(raw_input("Hviken dag har du bursdag (1-31)? "))

# sjekk at datoen er gyldig
if year < 1900 or year > 2010:
    print "Ugyldig årstall"
    quit()

# regn ut antall dager mellom i dag og fødselsdagen
today = datetime.date.today()
birthday = datetime.date(year, month, day)
days = (today-birthday).days

# lag en tabell med navn for ukedagene
weekdays = {0:'mandag',
            1:'tirsdag',
            2:'onsdag',
            3:'torsdag',
            4:'fredag',
            5:'lørdag',
            6:'søndag'}

# slå opp i tabellen for navn på ukedagene
birthday_weekday = weekdays[birthday.weekday()]
today_weekday = weekdays[today.weekday()]

# skriv ut en tekst til skjermen
print "Hei {}! I dag er det {}.".format(name, today_weekday)
print "Du ble født på en {}".format(birthday_weekday),
print "for {} dager siden.".format(days)

quit()
```

Glosepugg (**)

Her er et program som kan hjelpe deg med å pugge ukas engelske gloser. Legg merke til at den gradvis kan gi deg mer og mer hint hvis du ikke husker det engelske ordet.

Bruk en tekstediter for å skrive inn programmet:

```
nano glosepugg.py
```

og så kan du starte programmet slik:

```
python glosepugg.py
```

Oppgaver:

a) Skriv inn og kjør programmet. Klarer du 100% ?

b) Legg inn dine egne ord.

c) Klarer du å endre koden slik at den tilfeldig spør litt både på engelsk og norsk?

```
# -*- coding: utf-8 -*-

import random

words = [
  ("hus","house"),
  ("mus","mouse"),
  ("bil","car"),
  ("vann","water"),
  ("vegg","wall")]

random.shuffle(words)

print "Velkommen til glosepugg! ",
print "(q for å avslutte)"

total_attempts = 0

for n, e in words:
  attempts = 0
  expected = e
  next_word = False
  while not next_word:
    input = raw_input("Hva er {} på engelsk? ".format(n))
    if input == 'q':
      quit()
    attempts += 1
    total_attempts += 1
    if input == e:
      print "Riktig. Du er flink!"
      next_word = True
    else:
      hint_len = attempts - 1
      hint = e[:hint_len] + "." * (len(e) - hint_len)
      print "FEIL! Prøv igjen. (hint: {})".format(hint)

score = 1.0 * len(words) / total_attempts
print "Din score = %2.2d%%" % (score*100)
```

Tegne (*)

Nå skal vi lære å bruke et bibliotek som egentlig er laget for å programmere spill. Vi skal tegne noen linjer, rektangler og sirkler på skjermen.

Lag en ny programfil med en tekstediter:

```
nano tegne.py
```

og så skriver du inn programmet som du ser til høyre. Linjene som starter med # trenger du ikke å skrive inn, det er bare kommentarer til programkoden som maskinen ikke skal bry seg om.

Når du er ferdig kjører du programmet slik:

```
python tegne.py
```

Oppgaver:

a) skriv inn og kjør programmet. Det oppstår noen merkelige mønstre der linjene tegner en vifte. Hva tror du det kommer av?

b) lag en grå bakgrunnsfarge (hint: 150,150,150)

c) kan du tegne et enkelt hus, en busk og en sol?

```
import pygame
import time

BLACK = (0,0,0)
WHITE = (255,255,255)
GRAY = (150,150,150)
BLUE = (0,0,255)
GREEN = (0,255,0)
RED = (255,0,0)

pygame.init()

size = (640,480)
screen = pygame.display.set_mode(size)

screen.fill(GRAY)
pygame.draw.line(screen, GREEN, [100,100], [150,200], 3)
pygame.draw.circle(screen, RED, [300,200], 60)
pygame.draw.rect(screen, BLUE, [400,200,10,20])

for i in range(0,100):
  pygame.draw.line(screen, BLACK, [500,10], [100+i*2,300], 1)

pygame.display.update()

done = False
while not done:
  for event in pygame.event.get():
    if event.type == pygame.KEYDOWN:
       done = True

pygame.quit()
```

```
import pygame

pygame.init()

width, height = 640, 320
speed_x = 1.22
speed_y = 1.33

BACKGROUND = (10,10,10)
BALLCOLOR = (100,10,255)

screen = pygame.display.set_mode((width,height))

ball = pygame.Surface((20,20))
pygame.draw.circle(ball, BALLCOLOR, (10,10), 10, 0)
ballrect = ball.get_rect()

while True:
    for event in pygame.event.get():
        if (event.type == pygame.QUIT or event.type == pygame.KEYDOWN):
            quit()

    ballrect = ballrect.move((speed_x,speed_y))
    if ballrect.left < 0 or ballrect.right > width:
        speed_x = -speed_x
    if ballrect.top < 0 or ballrect.bottom > height:
        speed_y = -speed_y

    screen.fill(BACKGROUND)
    screen.blit(ball, ballrect)
    pygame.display.flip()
```

Sprettball

Her skal vi få en liten ball til å sprette rundt på skjermen.
Lag en ny programfil med en teksteditor:

```
nano sprettball.py
```

Når du er ferdig kjører du programmet slik:

```
python sprettball.py
```

Oppgaver:

a) skriv inn og kjør programmet.

b) prøv å legg en ball til som kan sprette rundt på skjermen.

Ekstraoppgaver

Primtall

Et primtall er et tall større enn 1 som bare er delbart med seg selv og 1. De ti første primtallene er:

```
2, 3, 5, 7, 11, 13, 17, 19, 23, 29
```

Vi skal lage et program som regner ut primtall. Bruk en teksteditor for å skrive inn programmet:

```
nano primtall.c
```

Programmet er skrevet i programmeringsspråket C og før det kan kjøres så må det konverteres til maskinkode av en såkalt kompilator:

```
gcc -O2 -o primtall primtall.c
```

gcc er navnet på programmet som kompilerer koden. "-O2" betyr at kompilatoren kan bruke litt ekstra tid på å optimalisere koden, "-o primtall" betyr at output-filen skal hete "primtall", og så oppgir vi navnet på filen som skal kompileres.

Hvis du skrev inn programmet nøyaktig som vist her så har du fått en såkalt eksekverbar objektfil som maskinen kan kjøre direkte.

```
./primtall
```

Oppgaver:

a) Skriv inn og kjør programmet. Hvor mange primtall er det mellom 1000 og 1100?

b) Bruk en stoppeklokke eller bruk kommandoen time for å måle tiden det tar å kjøre programmet, feks:

```
time ./primtall
```

Prøv å endre programmet til å regne ut større og større primtall intill du ser at det begynner å gå ganske sakte. Hvor stort må tallene være før det tar over et sekund å sjekke om det er et primtall? (trykk Ctrl-C for å avbryte et programm som kjører alt for sakte)

c) Et tall n som ikke er delelig med noen av tallene fra 2 til kvadratroten av n er et primtall. Klarer du å bruke den kunnskapen til å lage programmet mye raskere? (hint: predikatet i <= sqrt(n) kan også skrives i*i <= n)

d) Søk på nettet eller spør deg litt rundt. Hva er ansett som den raskeste måten å generere primtall på?

```c
#include <stdio.h>
#include <stdbool.h>

bool is_prime(int n)
{
    if (n < 2)
        return false;
    int i = 2;
    while (i < n) {
        if (n % i == 0)
            return false;
        i++;
    }
    return true;
}

int main(void)
{
    int from = 1000;
    int to = from + 100;
    int n = from;
    while (n < to) {
        if (is_prime(n))
            printf("%d\n", n);
        n += 1;
    }
}
```

Deep Thought (obligatorisk)

I Douglas Adams fantastiske bok "The Hitchhiker's Guide to the Galaxy", er det en datamaskin, Deep Thought, som regner ut det ultimate svaret på "Life, the universe, and everything". Det tok 7.5 millioner år å regne ut svaret, og det var et tall. Problemet var bare at ingen visste hva spørsmålet var...

Her skal vi skrive et lite program som også regner ut det ultimate svaret. Den bruker riktignok mye kortere tid en Douglas Adams maskin. Men problemet er fortsatt at vi ikke vet hva spørsmålet er.

Skriv inn koden akkurat slik den står, prøv å reflekter over hva hver linje betyr.

```
nano deepthought.s
```

Programmet er skrevet i assembler. Den kan lett oversettes til maskinkode med en såkalt assembler og deretter kan man lage en kjørbar fil med en linker:

```
as -o deepthought.o deepthought.s
ld -o deepthought deepthought.o
```

Hvis du skrev inn programmet nøyaktig som vist her så har du fått en såkalt eksekverbar objektfil som maskinen kan kjøre direkte, og så kan vi sjekke returverdien fra programmet for å finne ut hva svaret er:

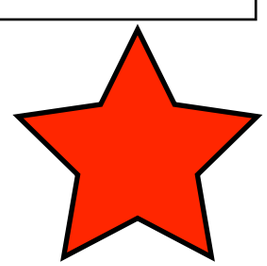
```
./deepthought
echo $?
```

Oppgaver:

a) Skriv inn og kjør programmet. Hva er det ultimate svaret?

b) Prøv å forstå hva programmet gjør. Assembleroperasjonen for å multiplisere heter MUL og den tar tre operander. Klarer du å endre programmet slik at den bare multipliserer 6 og 7, og deretter returnere det riktige resultatet?

```
.globl _start

_start:
mov r1, $6
mov r2, $7
mov r3, $0

loop:
cmp r2, $0
beq exit
add r3, r3, r1
sub r2, r2, $1
b loop

exit:
mov r0, r3
mov r7, $1
svc $0
```



LKK Cisco 21. februar 2014