Workshop 3: Lær koding på Raspberry Pi

Olve Maudal, Cisco Systems

4-timers workshop
LKK-konferansen for lærere

UiO : Universitetet i Oslo

Fredag 15. november 2013

Raspberry Pi er en billig, men kraftig, datamaskin som ble utviklet med formål å la barn lære seg programmering. Den ble lansert tidlig 2012 og ble en umiddelbar slager. Over en million enheter solgt det første året. Samtidig har det blitt utviklet enormt mye fritt tilgjengelig læringsmateriell og programvare. (http://www.raspberrypi.org/)

I denne sesjonen vil du lære å sette opp en Raspberry Pi og så vil vi demonstrere hvordan den kan brukes til å lære bort "ordentlig" programmering. Ingen forkunnskaper er påkrevd, men her skal vi bruke en teksteditor og jobbe på kommandolinjen. Her får du muligheten til å lære litt om både Python og C, begge er programmeringspråk som brukes mye i industrien.

Olve er softwareguru i Cisco Systems hvor han jobber med utvikling av videokonferansesystemer. Tidligere erfaring inkluderer utvikling av mobile betalingsterminaler, banksystemer, og systemer for seismisk datainnsamling. Han holder ofte foredrag og kurs om programmering, embedded systems, programdesign, arkitektur og arbeidsmetodikk.

Deltagere

Kristian Bergaplass, Lærer, Kuben vdg.
Pål Iversen Holst-Jæger, IT-ansvarlig/Konsulent, OSF Brusetkollen AS
Thomas Johanson, Undervisningsinspektør, Hundsund ungdomsskole
Håkon Kaurel, Koordinator, Blussuvoll/Trondheim
Lars Klingenberg, Koordinator, Blussuvoll/Trondheim
Charlotte Kongshavn, Lektor i realfag, Teknologiskolen Hundsund
Tjerand Silde, Koordinator, Blussuvoll/Trondheim
Siri Ann Vatland, Lærer, Gosen ungdomsskole
Bjørn Helge Rømen, Lærer, Thor Heyerdahl VGS, Larvik
Bernd Pfeiffer, Lærer, Thor Heyerdahl VGS, Larvik
Alvin Alexander Ghouas, WeWantToKnow, Forskningsparken
Kristine Sevik, Senter for IKT i utdanningen
Vibeke Guttormsgaard, Senter for IKT i utdanningen
Håvard K Mack, Seljestad barneskole

Hjelpelærer:
Hanna Maudal, elev, Levre Skole

Agenda

1230 Velkommen
1245 Komme i gang med Raspberry Pi
1315 Spillhacking
1400 Pause
1415 Valgfrie øvelser
1545 Pause
1600 Oppsummering og Q&A
1630 Slutt

Læringsmål:

- sette opp en Raspberry Pi maskin
- jobbe fra kommandolinjen
- modifisere eksisterende programmer
- lære litt om programmering
- skrive sine egne småprogrammer

Agenda

1230 Velkommen

1245 Komme i gang med Raspberry Pi
- sette opp maskin
- logge inn og bruke kommandolinjen
- starte spill

1315 Hacking
- editere filer
- hacke et spill

1400 Pause

1415 Introduksjon til øvelsene

1430 Valgfrie øvelser (velg 2 eller 3)
- Hva heter du? (Python) *
- Mattequiz (Python) **
- Glosepugg (Python) **
- Sprettball (Python/pygames) **
- Primtall (C) ***
- Deep Thought (asm) **

1545 Pause

1600 Oppsummering og Q&A

1630 Slutt

Komme igang

Øvelse: Spillhacking

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 (**)
- d) gjør slik at du ikke krasjer i veggen, men kommer ut på den andre siden (***)

Pause

Valgfrie øvelser

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)

Mattequiz (**)

Her er et program som kan hjelpe deg med å lære deg gangetabellen.

Bruk en tekstediter for å skrive inn programmet:

```
nano mattequiz.py
```

og så kan du starte programmet slik:

```
python mattequiz.py
```

Oppgaver:

a) Skriv inn og kjør programmet.

b) Endre programmet så den spør om tall mellom 2 og 20.

c) Lag en versjon som tester addisjon, substraksjon, multiplikasjon og divisjon

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

import random

print "Velkommen til MatteQuiz! (q for å avslutte)"

count = 0
errors = 0

while True:
    a = random.randint(2,7)
    b = random.randint(2,7)
    expected = a * b
    correct = False
    count += 1
    while not correct:
        print "Hvor mye er {} * {}? ".format(a,b)
        input = raw_input()
        if input == "q":
            score = 100 * count / (errors + count)
            print "{}% score!".format(score)
            quit()
        guess = int(input)
        if guess == expected:
            correct = True
            print "Riktig! Du er flink! :-)"
        else:
            errors += 1
            print ":-( Prøv igjen"
```

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)
```

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.

```
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()
```

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)
{
    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 (**)

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?

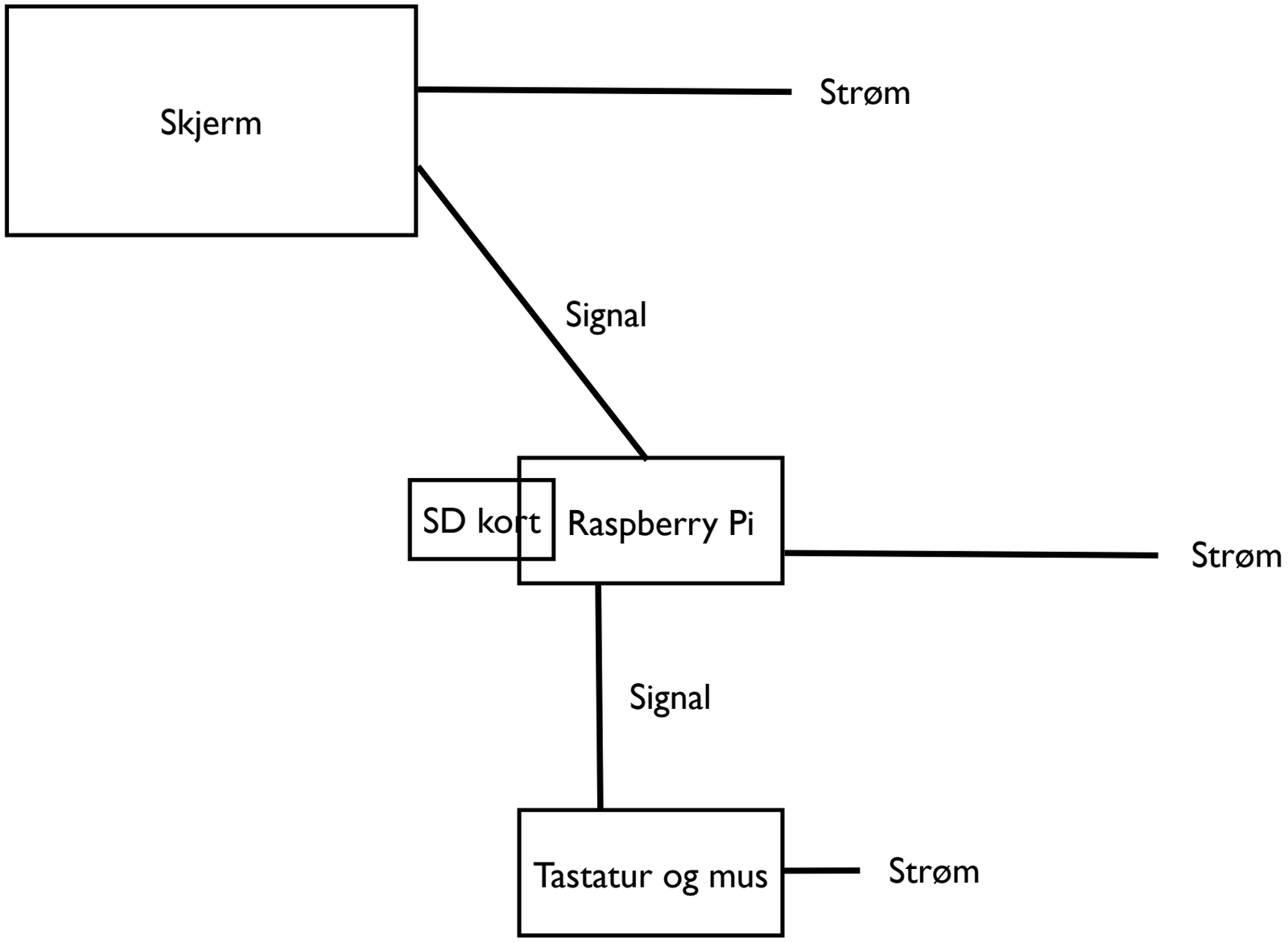
b) Prøv å forstå hva programmet gjør. Assembleroperasjonen for å multiplisere heter MUL og den tar tre operander. Klarer du å endre programmet slik at den bare multipliserer 6 og 7, og deretter returnere det riktige resultatet?

```
.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
```

Pause

Oppsummering og Q&A

!


Skjerm
Strøm
Signal
SD kort
Raspberry Pi
Strøm
Signal
Tastatur og mus
Strøm