Løsningsforslag til Oppgaver dag 1

Dette er løsningsoppgaver til oppgavene som ble gitt til Dag av Programmering med Python 1. Det er lurt å gjøre et ordentlig forsøk på en oppgave før du ser på løsningsforslaget. Samtidig vil vi minne på at dette med programmering er helt nytt for de aller fleste som tar dette kurset, og det er derfor helt forståelig om det er vanskelig å få til oppgavene uten litt hjelp! Oppgavene vi gir er ganske utfordrende, og består av mange deloppgaver - det er derfor ikke plankekjøring! Om noen av løsningene er vanskelig å forstå seg på kan dere selvfølgelig alltid be oss forklare nærmere. Send meg gjerne en mail (jonas@simula.no), eller spør i en av live-øktene.

Oppgave 1 *Printing*

a) For å skrive ut en beskjed til skjermen bruker vi `print`, og vi må ha anførselstegn på hver side av selve beskjeden vi skriver ut.

```
print("Hello, World!")
```

b) Her kan vi skrive ut en beskjed som bruker variabelen på to ulike måter:

```
navn = "Jonas"
print("Hei på deg", navn)

# Eller med flette-printing
print(f"Hei på deg {navn}!")
```

c) For eksempel:

```
navn = "Jonas"
print(f"Hei {navn}! Du har {len(navn)} bokstaver i navnet
    ditt!")
```

d) Her er det viktig at vi bruker `input`, og lagrer variabelen på samme kodelinje - den er litt vrien, så her kan man gjerne se på jukselappen eller et tidligere kodeeksempel om man ikke husker helt syntaksen

```
navn = input("Hva heter du? ")
print(f"Hei {navn}! Du har {len(navn)} bokstaver i navnet
    ditt!")
```

```
Hva heter du? Jonas
Hei Jonas! Du har 5 bokstaver i navnet ditt!
```

Oppgave 2 *Konvertering av temperatur*

I denne oppgaven er utfordringen å implementere den matematiske formelen

$$C = \frac{5}{9}(F - 32).$$

som kode.

a)

```
F = 350
```

b) Pass på at Python skiller på små og store bokstaver. Siden jeg her har valgt å kalle variabelen `F`, så må jeg bruker stor `F` når jeg skriver inn det matematiske uttrykket.

```
C = (5/9) * (F - 32)
```

c) For å sjekke hvor mange grader 350° er i Celsius må vi skrive ut `C`-variabelen

```
print(C)
```

Da får vi isåfall

```
176.66666666666669
```

Det virker rimelig å skulle steke en kake ved ca 180 grader celsius. Vi kan lage en mye penere utskrift om vi ønsker det, f.eks

```
print(f"{round(F)} grader Fahrenheit er omtrent {round(C)}
    grader celsius.")
```

Her skriver vi ut en tekst-beskjed med mer informasjon, bruker flette-printing til å få inn verdiene av både `F` og `C`, og vi bruker `round` for å runde av til nærmeste hele grad.

```
350 grader Fahrenheit er omtrent 177 grader celsius.
```

d) For å snu formelen må vi bruke litt algebra, men da kommer vi frem til

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

Dette er jo en mer matematikk-oppgave enn programmering, men matematikk er en ganske viktig del av det å programmere ofte. Den viktige forskjellen i formelen er nå at det ikke er noen parentes rundt $C + 32$.

e)

```
C = 0
F = (9/5)*C + 32
print(f"{round(C)} grader celsius tilsvarer omtrent {round(F)
    } grader fahrenheit.")
```

f) Her kan du rett og slett endre første kodelinje og kjøre programmet på nytt, så først lar du `C = 0`, deretter `C = 100`. Da får vi

```
0 grader celsius tilsvarer omtrent 32 grader fahrenheit.
100 grader celsius tilsvarer omtrent 212 grader fahrenheit.
```

Her kunne du eventuelt slå opp frysepunktet og kokepunktet til vann på internett for å dobbeltsjekke at koden fungerer som den skal.

Oppgave 3 *Trille terninger*

Dette er en oppgave der dere prøver å tolke kode, fremfor å skrive den selv. Det er en god øvelse for å lære ny programmering, og man kan finne mange fine kodeeksempler for dette mest på nett eller i bøker. Om du ønsker å gjøre noe du ikke kan fra før, kan du alltid google opp et eksempel som gjør noe litt lignende, prøve å forstå den, og deretter tilpasse den.

a) Programmet bruker funksjonen 'randint' til å lage tilfeldige tall. Når vi skriver 'randint(1, 6)' får vi et tilfeldig tall mellom 1 og 6, og dette svarer altså til å rulle en vanlig terning. Vi oppretter to ulike variabler, som representerer hver sin terning. Programmet skriver så ut summen av verdien til disse terningene. Vi forventer altså at resultatet skal være et tall mellom 2 (1+1) og 12 (6+6). Trikset for å forstå denne koden er jo å finne ut av randint". Det kryptiske navnet kommer fra engelsk, der randinter en sammenslåing av *random integer*. Det er kanskje ikke så lett å skjønne når man aldri har sett denne funksjonen i bruk før, så her kan man enten prøve å kjøre koden, eller google litt for å prøve å finne ut av hva som foregår.

b) Om du kjører koden flere ganger ser du fort at du får et nytt (tilfeldig) tall hver gang, tallene blir mellom 2 og 12. Det er fordi programmet

c) Om vi skriver inn koden og kjører den kan f.eks tre kjøringer på rad gi følgende

```
8
```

```
11
```

```
8
```

Merk at hver kjøring av programmet er uavhengig av hverandre (akkurat som om du triller fysiske terninger). Det er derfor fullt mulig å få samme resultat flere ganger på rad, selv om det ikke er veldig sannsynlig. I dette eksempelet fikk vi 8 to ganger av tre.

d) Koden vi har gitt lagrer allerede resultatet av hver terning, så vi kan rett og slett skrive disse ut, f.eks

```
1 print(terning1, "+", terning2, "=", terning1 + terning2)
```

e) Om vi bruker `randint(2, 12)` får vi et tilfeldig tall mellom 2 og 12, men disse vil være *uniformt* fordelt, altså vil det være like sannsynlig og få 2, eller 5, eller 10 eller

12. Men om vi triller to terninger og legger dem sammen er det langt mer sannsynlig å få resultateter nært midten, altså 7, enn f.eks 2 eller 12.

f) Her skriver vi en betingelse som sjekker om de to terningene har samme verdi. Her må vi huske på doble likhetstegn!

```
    if terning1 == terning2:
        print("De to terningene er like!")
```

Oppgave 6 *Minutter og timer*

Dette programmet spør brukeren om å skrive inn et antall minutter. Koden regner så om dette til antall hele timer, og antall restminutter. F.eks blir 90 minutter til 1 time og 30 minutter. La oss forklare de ulike bitene hver for seg

a) Her spør vi brukeren om å skrive inn antall minutter. Vi gjør om svaret fra *tekst* til *tall* med `int()`.

b) Kodelinjene regner antall *hele* timer, og antall restminutter vi sitter igjen med.

c) Betingelsen (`if`-testen) bruker vi så kun for å få en penere utskrift. Vi ønsker f.eks at 120 minutter skal bli til "2 timer", ikke "2 timer og 0 minutter". Testen er altså mest kosmetisk, for å lage pen utksrift.

Oppgave 7 *Gangetabell-Quiz*

a) Koden trekker to tilfeldige tall mellom 1 og 10 (om du ikke kjenner til `randint()` bør du se på *Trille Terning*-oppgaven over. Programmet spør så hva de to tallene ganget med hverandre blir. Et kjøreeksempel kan se ut som følger

```
Hva er 2 ganger 6?
```

Her venter programmet på et svar. Merk at svaret gjøres om til tall med `int()`—det er viktig for neste oppgave, for en `if`-test som sammenligner tekst og tall oppfører seg litt rart.

b) En slik test kan se ut som følger. Husk to likhetstegn for å sammenligne i en betingelse (fordi ett likhetstegn brukes for å opprette variabler i Python)

```
if svar == a*b:
    print("Riktig!")
else:
    print(f"Det er desverre feil. ({a} x {b} = {a*b})")
```

c) Den siste deloppgaven er ganske utfordrende. Men trikset er å først løse de andre oppgavene, og deretter gi alle kodelinjene et innrykk og legge de på innsiden av en løkke så de gjentar seg. Du kan gi mange kodelinjer innrykk ved å velge dem i Spyder og trykke Tab. Da gir du alle et innrykk samtidig. En kode kan se ut som følger

```python
# Importeringen trenger vi ikke gjenta
from random import randint

# Vi lager en while-løkke som gjentar seg fem ganger
spørsmål = 0
riktige = 0
while spørsmål < 5:
    # Husk å øke tellevariabelen, ellers får vi en uendelig l
       økke
    spørsmål += 1

    # Resten av koden er lik som tidligere
    a = randint(1, 10)
    b = randint(1, 10)

    svar = int(input(f"Hva er {a} ganger {b}? "))

    if svar == a*b:
        print("Riktig!")
        riktige += 1
    else:
        print(f"Det er desverre feil. ({a} x {b} = {a*b})")

print(f"Du fikk til {riktige} av {spørsmål}.")
```

Oppgave 8 *Spambot*

a) Som oppgavens navn tilsier er dette en kode som lager litt *spam*. Det er to elementer av koden som kan være ekstra vanskelig å forstå seg på. Det første er selve `while`-løkka. Siden variabelen `count` begynner på 10, og reduseres med én hver gang løkka gjentar seg selv, så vil løkka kjøre ti ganger på rad. Det andre som kanskje er litt forvirrende er at vi skriver `print(10*"Spam!"`. Det som står mellom anførselstegnene er bare en tekstbeskjed, men utenfor dette ganger vi altså med 10. Hva vil det si å gange en tekst med 10? I Python betyr dette at teksten skrives ut 10 ganger på rad. Dette kan du teste selv ved å kjøre koden.

b) Når koden kjøres får vi skrevet ut ti linjer, med 10 SPAM! på hver linje.

```
SPAM! SPAM! SPAM! SPAM! SPAM! SPAM! SPAM! SPAM! SPAM! SPAM!
SPAM! SPAM! SPAM! SPAM! SPAM! SPAM! SPAM! SPAM! SPAM! SPAM!
SPAM! SPAM! SPAM! SPAM! SPAM! SPAM! SPAM! SPAM! SPAM! SPAM!
...
```

Hver linje svarer til en gjentakelse av løkka, og hver linje har 10 SPAM!fordi vi ganger med 10 inne i `print`-funksjonen.

c) Det er ti linjer med ti SPAM!per linje, så det vil være $10 \times 10 = 100$ utskrifter totalt.

Oppgave 9 *Håndhilseproblemet*

Her har vi fyllt inn i skjelettkoden

```
n = 19
total = 0

while n > 0:
    total = total + n
    n = n - 1

print(f"Det er {total} håndtrykk totalt")
```

```
Det er 190 håndtrykk totalt
```

Oppgave 10 *Sparekalkulator*

Når vi fyller inn i skjelettkoden kan programmet se ut som følger

```
spart = 0
måneder = 0

while spart < 6500:
    spart += 725
    måneder += 1

print(f"Siri må spare i {måneder} måneder")
print(f"Hun har da spart {spart} kr. (Det er {spart - 6500} kr
   til overs)")
```

Den siste printen er ikke nødvendig for å svare på oppgaven, siden vi egentlig bare spurte om antall måneder hun må svare. Men det kan være interessant informasjon å skrive ut. Når kode kjøres får vi:

```
Siri må spare i 9 måneder
Hun har da spart 6525 kr. (Det er 25 kr til overs)
```

Oppgave 11 *Tallrekker*

Tallrekkene kan lages på ulike måter. Under er en mulig kodesnutt for hver løkke, men de kan løses på andre måter. Merk også at koden for hvert eksempel i stor grad er lik, her kan det lønne seg og kopiere og lime kode for å være litt effektiv (og unngå skrivefeil)

a) Skriv ut tallene 1 til 100

```
tall = 1
while tall <= 100:
```

```
    print(tall)
    tall += 1
```

b) Skriv ut 7-gangen fra og med 0 til og med 70

```
tall = 0
while tall <= 70:
    print(tall)
    tall += 7
```

c) skriv ut rekka 1, 4, 7, 10, . . . , 31 (Hint: Den begynner på 1 og øker med 3 hver gang) tall = 1 while tall <= 31: print(tall) tall += 3

d) Skriv ut rekka 1, 3, 9, 27, . . . , 729 (Hint: Vi ganger med 3 hver gang)

```
tall = 1
while tall < 1000:
    print(tall)
    tall *= 3
```

Vi bruker i disse eksemplene notasjonen `+=` og `*=`. Om du syns dette er forvirrende kan du skrive f.eks `tall =3*tall` istedenfor `tall *=3`. De to formene er helt ekvivalente, poenget er bare at vi ønsker å redefinere variabelen, det vil si endre på den.

Oppgave 12 *While-løkker for hånd*

Løkkene skriver ut følgende

a) $2, 4, 8, 16, 32$

b) 5.26

c) $3, 6, 16$

Oppgave 13 *Fakultet*

Et ferdig program kan se ut som følger

```
n = int(input("Velg n: "))
original = n

fakultet = 1
while n > 0:
    fakultet = fakultet * n
    n = n - 1

print(f"{original}! = {fakultet}")
```

Vi får følgende utskrifter

```
Velg n: 5
5! = 120
```

```
Velg n: 52
52! = 80658175170943878571660636856403766975289505440883277824000000000000
```

Dette er et helt enormt antall mulige stokkinger av en vanlig kortstokk. Man kan derfor med mer eller mindre garantert sikkerhet si at dersom man stokker en kortstokk godt så vil den være i en rekkefølge som aldri noen gang før har vært i historien (sannsynligheten for å få samme stokking"av en kortstokk er mindre enn å vinne toppremien i lotto flere ganger på rad).

```
Velg n:  54
54! = 23084369733924138047209274268302758108327856457180794113228800000000000000
```

I dette siste eksempelet må vi strengt tatt dele svaret på to til slutt. Fordi de to jokerene vi har lagt til er ekvivalente. Men svaret blir fortsatt enoooormt stort.

En siste kommentar vi ønsker å gi er at når vi jobber med så store tall som dette kan det hjelpe å skrive dem ut på det som heter vitenskapelig notasjon, eller standard-notasjon. Dette kan vi gjøre med flette-printing ved å legge til enten `:g` eller `:e` etter variabelen, på innsiden av krøllparentesene. Dette kalles *print-formatternig* og sier til Python at vi ønsker variablene skrevet ut med en bestemt stil, i dette tilfellet vitenskapelig notasjon

```
1 print(f"{original}! = {fakultet:g}")
```

Da blir isåfall de tre utskriftene

```
5! = 120
52! = 8.06582e+67
54! = 2.30844e+71
```

Dette svarer altså til å si at f.eks

$$52! = 8.06582 \cdot 10^{67}.$$