Bonusoppgaver

4. mai 2022

Dette er en samling med bonusoppgaver du kan prøve deg på om du ønsker. Om du ønsker enda flere oppgaver enn det som er delt i oppgavene for dag 1, dag 2 og disse bonusoppgavene anbefaler vi at du sjekker ut følgende nettsider:

- https://projecteuler.net/
- https://www.codewars.com/

Disse nettsidene har veldig mange programmeringsoppgaver å bryne seg på. Den første (Project Euler) inneholder matematiske spørsmål som er spesielt egnet for programmering, det er på en måte små mattenøtter. Du kan lese oppgavene uten å lage en bruker, men om du vil sjekke fasiten må du lage en bruker. Da får du også tilgang til et forum etter hver oppgave du løser hvor du kan se hvordan andre har løst oppgaven. Oppgavene er ikke knyttet til noe bestemt programmeringsspråk, men Python vil være et fint verktøy for å løse dem. CodeWars har mer generelle programmeringsoppgaver, og er en litt mer avansert nettside der du programmerer løsningene rett i nettleseren. Codewars er en nettside som vil gjøre programmeringen nesten som et spill, noe som kan passe visse mennesker meget godt. Nettsiden støtter mange programmeringsspråk, inkludert Python. Den har også en liten prøve man kan ta så den skal klare å tilpasse oppgavene til ditt ferdighetsnivå. Du må lage en gratis bruker for å bruke nettsiden.

Oppgave 1 *Jordkloden*

I denne oppgaven skal vi øve på å bruke Python som kalkulator, ved å regne litt på jordkloden. Husk at formelen for volumet av en kule er

$$V = \frac{4}{3}\pi R^3.$$

a) Jordkloden er tilnærmet en perfekt kule, og har en radius på 6371 km. Lag et kort program som først definerer en variabel `radius`, og deretter regner ut en variabel `volum`. Skriv til slutt ut svaret til brukeren med `print()`-funksjonen. La svaret være i km^3.

b) Endre programmet ditt så svaret istedet skrives ut i antall liter.

c) Den totale massen til jordkloden er omtrent $M = 5.972 \cdot 10^{24}$ kg. Regn ut hvor mange kg hver liter av jordkloden veier i gjennomsnitt. Virker svaret ditt rimelig?

Oppgave 2 *Brette ark*

Et vanlig ark er omtrent 0,1 mm tykt. Om vi bretter arket på midten dobbler vi tykkelsen av arket, så det er 0,2 mm tykt. Om vi bretter arket på nytt dobbler vi igjen tykkelsen, så det blir 0,4 mm tykt. Sånn kan vi fortsette å brette arket for å gjøre det tykkere. Om du prøver i praksis viser det seg nok fort at det er veldig vanskelig å brette arket noe særlig mer enn 6-7 ganger. Men om vi nå later som vi kunne brettet arket så mange ganger vi vil, er det ingen grense for hvor tykt arket kunne blitt.

Verdens høyeste bygning er Burj Khalifa i Dubai, som er 828 meter høyt. Vi ønsker å finne ut hvor mange ganger vi må brette arket vårt, før det er like tykt som høyden av denne bygningen.

a) Diskuter med sidemannen hvordan dere kunne funnet ut av dette med penn og papir. Hva er ulempen med fremgangsmåten deres?

Vi skal nå løse problemet ved hjelp av et kort Python-program. For å løse problemet bruker vi en løkke for å brette arket helt til vi har nådd den tykkelsen vi er ute etter.

b) Fyll inn skjelettkoden under for å finne antall brett vi trenger. Pass spesielt på at tykkelsen til arket er oppgitt i millimeter, mens bygningen er oppgitt i meter.

```
tykkelse = ...
antall_brett = ...

while ...:
    tykkelse *= ...
    antall_brett += ...

print(...)
```

c) Om du har klart å løse oppgaven. Finn antall brett som skal til før tykkelsen er like høyt som verdens høyeste fjell, Mount Everest, som er 8848 meter høyt.

d) Avstanden fra jorda til månen er ca 384400 km. Hvor mange ganger må arket brettes før det er like tykt som denne avstanden?

Oppgave 3 *Telleleken FizzBuzz*

FizzBuzz er en tellelek som er populær i den engelske skolen når man lærer om gangesystemet. I leken så skal man, enten alene eller som gruppe, telle oppover fra 1 til 100. Hver gang man skal si et tall som er delelig med 3, skal man si Fizz istedenfor tallet, og hver gang man skal si et tall som er delelig med 5 skal man si Buzz istedenfor tallet. Tall som er delelig med *både* 3 og 5, bytter man ut med *FizzBuzz*. Man skal altså telle oppover som dette

$$1, 2, \text{Fizz}, 4, \textit{Buzz}, \textit{Fizz}, 7, 8, \textit{Fizz}, \textit{Buzz}, 11, \textit{Fizz}, 13, 14, \textit{FizzBuzz}, 16, 17, \ldots.$$

FizzBuzz er blitt en meget kjent programmeringsoppgave. Dette er fordi det er et problem det er veldig enkelt å forstå seg på, men for å løse det med programmering må man klare å bruke både algoritmisk tenkning, løkker og betingelser. Derfor var det en stund en populær oppgave å gi i jobbintervjuer, for å sjekke at kandidaten behersket grunnleggende programmering. I dag er nok oppgaven blitt såpass velkjent at den ikke lenger brukes til dette formålet. Men det er fortsatt en god øvingsoppgave.

Du kan prøve å skrive programmet fra bunn av selv, eller løse deloppgavene slik de er formulert under for å gå frem stegvis. Uansett er det lurt å løse en bit av problemet av gangen og sjekke at det funker som du tror underveis

a) Lag et nytt program (f.eks `fizzbuzz.py`)

b) Lag en `for`-løkke som skriver ut tallene 1 til 100 etterhverandre (Hint: Her må du bruke `range()`)

c) Legg nå til en `if`-test inne i `for`-løkka, som sjekker om hvert tall er delelig på 3 før du skriver ut noe. Pass på innrykk her, se skjellettkoden vår for hvordan dette bør se ut. Selve testen for å sjekke om et tall er delelig på 3 er litt kryptisk, så denne kan du skrive av koden vår om du ønsker

```
for tall in range(...):
    if tall % 3 == 0:
        print(...)
    else:
        print(...)
```

Betingelsen er `tall % 3== 0`. Prosenttegnet er det vi kaller *modulo*-operatoren, den gir resten etter en divisjon. Så her sier vi at Python skal prøve å dele tallet på 3 og sjekke om resten vi sitter igjen med blir 0. F.eks vil 7 delt på 3 gi 2

med 1 i rest. Siden resten ikke er null ser vi at 7 ikke er delelig med 2. Derimot vil 9 delt på 3 bli 3, uten rest. Siden resten er null er 9 delelig med 3.

d) Etter at du sjekker at *Fizz*-biten av koden fungerer som du vil, kan du nå legge til `Buzz`-biten av koden. Det gjør du ved å legge til en `elif`-betingelse mellom `if`- og `else`-blokkene. Betingelsen i `elif`-blokka må sjekke om tallet er delelig med 5.

e) Sjekk om koden fungerer som du tror. Du er nå nesten helt i mål, men mest sannsynlig vil koden for tallet 15, som skal bli *FizzBuzz* gi kun *Fizz*. Prøv å finn en måte å ta hensyn til tall som er delelig med både 3 og 5. (Hint: Du kan la den første `if`-testen være følgende betingelse)

```
1  if tall % 3 == 0 and tall % 5 == 0:
```

Derimot må du være litt forsiktig med hva som er `if`, `elif` og `else`, og hvilken rekkefølge de kommer i. Prøv deg frem til det blir slik du vil ha det.

Oppgave 4 *Stein, saks, papir*

Du skal nå lage et program der vi spiller det kjente folkespillet *Stein, saks, papir* mot datamaskinen. Her er poenget altså at brukeren først blir bedt om å velge én av de tre valgene, så velger datamaskinen et annet tilfeldig vanlg—så sammenligner vi og kårer en vinner. Et kjøreeksempel av programmet kan se ut som følger:

```
Velg stein, saks, eller papir: stein
Du valgte: stein
Datamaskinen valgte: papir
Du taper!
```

Du kan prøve å lage et slikt program helt fra bunnen av selv, eller du kan følge deloppgavene under. Uansett vil du fort merke at det er ganske mange betingelser og holde styr på, og vi anbefaler derfor at du tegner en liten skisse eller et *flytdiagram* for deg selv med penn og papir, for ikke å gå i surr.

a) Begynn med å bruke `input` til å spørre om brukeren velger stein, saks eller papir.

b) Deretter får vi datamaskinen til å velge et tilfeldig valg. Her gir vi rett og slett to kodelinjer som gjør dette, fordi det kanskje ikke er så enkelt å finne ut av hvordan det skal gjøres

```
from random import choice
datamaskin = choice(["stein", "saks", "papir"])
```

c) Lag nå en betingelse som sjekker om brukeren har valgt stein, saks, papir eller noe annet. Om de har skrevet noe annet enn de tre (fanges opp av en `else` til slutt) bør programmet si at valget er ugyldig. F.eks

```
Velg stein, saks eller papir: glass
Ugyldig valg.
```

d) For hver av de tre valgene brukeren velger, sjekk hva datamaskinen har valgt og skriv ut riktig svar. Altså kan programmet se ut som følger:

```
if bruker == "stein":
    if datamaskin == "stein":
        print(...)
    elif datamaskin == "papir":
        print(...)
    elif datamaskin == "saks":
```

```
7           print(...)
```

og helt tilsvarende for brukerens andre valg. Det er ganske mye kode du må skrive ut her, så det kan lønne seg å kopiere og lime kode for å spare litt tid.

e) Etter du har skrevet inn all koden bør du teste at koden virker skikkelig.

f) Til slutt kan du innarbeide en juksekode om du ønsker. F.eks

```
Velg stein, saks eller papir: atombombe
Ingenting slår en atombombe! Du vinner!
```

Oppgave 5 *Fibonacci-rekken*

Fibonacci rekken er en kjent tallfølge som naturlig oppstår mange steder i naturen. Tallfølgen går som følger:

$$1, 1, 2, 3, 5, 8, 13, 21, ... \tag{1}$$

Vi kan skrive Fibonacci rekken som en rekursiv tallfølge etter denne formelen:

$$a_n = a_{n-1} + a_{n-2}, \tag{2}$$

hvor $a_0 = 1$ og $a_1 = 1$.

En interessant egenskap ved Fibonacci rekken er at forholdet mellom to etterfølgende tall i følgen går mot det gyldne snitt, $\phi = \frac{1+\sqrt{5}}{2} \approx 1.62$. Det vil si at

$$\frac{a_n}{a_{n-1}} \to \phi. \tag{3}$$

a) Du skal nå lage et program som skriver ut de første 10 tallene i Fibonacci rekka. Start med å opprette en variabel `forrige_tall =1` og en variabel `fibonacci_tall =1`.

b) Skriv så ut `forrige_tall` og `fibonacci_tall` til brukeren av programmet.

c) Bruk en `for`-løkke som repeteres 8 ganger (10 tall - 2 start-tall) som skriver ut de resterende 8 tallene i Fibonacci-rekka. HINT: Her kan det være lurt å opprette det nye Fibonacci tallet i en variabel `nytt_fibonacci_tall` før du oppdaterer verdien til `forrige_tall` og `fibonacci_tall`.

d) Endre programmet slik at du bruker input til å be brukeren om hvor mange Fibonacci-tall du skal skrive ut til skjermen.

e) Oppdater programmet slik at du og skriver ut forholdet mellom `forrige_tall` og `fibonacci_tall`. Blir dette forholdet ca lik 1.618?

f) Hvis du endrer start-tallene fra $a_0 = 1$ og $a_1 = 1$ til $a_0 = 2$ og $a_1 = 1$ får du en tallrekke som kalles for *Lucas-tallene*. Hvordan oppfører Lucas-tallene seg i forhold til Fibonacci-rekken? Konvergerer forholdet mellom to etterfølgende tall til det samme tallet som før?

g) Prøv så å endre til helt villkårlige start-tall – negative tall, store tall, desimaltall, etc. Endrer konvergensen seg?

Oppgave 6 *Over/Under i revers*

I oppgaveøkten for Dag 1 viste Terje hvordan du kan lage gjettespillet Over/Under i Python. I det spill vil datamaskinen velge et tall mellom 1 og 100, og brukeren gjetter seg frem til det. Det kan være en morsom utfording å lage det *motsatte* programmet. Altså et program der brukeren tenker på et tall på mellom 1 og 100 og datamaskinen gjetter seg frem til det. Du kan prøve å lage et slik program fra bunnen av selv, eller du kan følge deloppgavene under. Om du ikke har sett gjettespillet Over/Under enda, bør du ta en titt på opptaket fra oppgaveøkten, eller lage et slikt program fra bunnen av først.

a) Om du ikke kjenner til gjettespillet Over/Under som dataprogram, enten se på eksempelkoden som ligger under `dag1/eksempler/gjettelek.py` på nettsiden eller ta en titt på opptaket fra oppgaveøkten hvor Terje livekoder denne.

b) Definer to variabler som representerer grensene for hvor tallet kan være, disse bør initialiseres til 1 og 100.

c) Be brukeren tenke på et tall mellom 1 og 100 og skrive det ned. Bruk `input` til å vente til de er klare med å gå videre. F.eks

```
input("Trykk enter når du har bestemt deg for et tall."
    )
```

d) Få datamaskinen til å gjette på et tilfeldig tall mellom nedre og øvre grense og skrive dette ut til skjermen. Dette kan du gjøre med `randint`.

e) Spør nå brukeren om gjettet er for høyt, for lavt eller helt riktig. Bruk gjerne en `while`-løkke for håndtere feil svar, så programmet kan være sikre på at brukeren har oppgitt et gyldig svar.

f) Avhengig av brukeren juster variabelene for nedre og øvre grense. Om datamaskinen f.eks har gjettet på 47 og brukeren sier at dette er for høyt må den øvre grensen nå justeres til å bli 46.

g) Test koden din med ett gjett og sjekk at grensene blir justert riktig ved å skrive dem ut til slutt.

h) Når koden fungerer fint for ett gjett, utvid koden med en løkke så det fortsetter helt til datamaskinen treffer riktig tall.

i) Legg til en tellevariabel som holder styr på antall gjett datamaskinen trengte for å treffe riktig svar.

j) Etter du er fornøyd med programmet du har laget kan du kopiere koden inn i en ny fil, så du ikke overskriver det gamle programmet ditt. Endre nå koden i det nye programmet så koden bruker halveringsmetoden istedenfor å gjette tilfeldig. Da skal datamaskinen altså velge midtpunktet mellom den nedre og øvre grensen for hver iterasjon. Dette er den "optimalefremgangsmåten, siden du garanterer at du treffer riktig tall på 7 gjett.

Oppgave 7 *Cæsar-chiffer*

En krypteringsalgoritme kan brukes for å gjøre beskjeder uleselig for alle som ikke har den riktige nøkkelen, og kan brukes for å sende hemmelige beskjeder. Cæsar-chifferet er en av de eldste krypteringsalgoritmene som finnes, og går ut på at man forskyver hver bokstav i en tekst like mange hakk bortover i alfabetet. Om vi f.eks velger å skyve alle bokstaver syv hakk vil aller 'a' bli til 'h', alle 'b' til 'h', alle 'c' til 'j' og så videre.

Å lage et Python-program som krypterer (eller dekrypterer) beskjeder ved hjelp av Cæsar-chifferet er en morro utfordring.

Lær Kidsa Koding deler på sine nettsider en detaljert gjennomgang av en slik algoritme og hvordan denne lages i Python. Naviger deg inn på følgende nettside og gjennomfør opplegget

- `https://oppgaver.kidsakoder.no/python/hemmelige_koder/hemmelige_koder`

Til info har nettsiden til Lær Kidsa Koding mange andre fine oppgaver og prosjekter i Python.

Oppgave 8 *Trekanttall*

Et trekanttall er summen av tallrekken

$$1, 2, \ldots, n.$$

For eksempel så er

$$1 + 2 + 3 + 4 + 5 = 15,$$

så da er 15 et trekanttall. Siden det var summen av de 5 første tallene i tallrekka, så sier vi at det er det *femte* trekanttallet.

La oss si at vi vil vite hva det hundrede trekanttallet er, da må vi legge sammen alle tallene fra 1 til 100. Det blir fort slitsomt og kjedelig å gjøre for hånd. Så la oss bruke programmering.

For å gjøre det lettere å sjekke om programmet vårt, så startet vi med å prøve å regne ut summen fra 1 til 5.

a) Lag en løkke som skriver ut tallene

$$1, 2, 3, 4 \text{ og } 5$$

til skjermen.

b) Endre nå løkka så du isteden finner summen av tallene 1 til 5. Da må du først lage en variabel utenfor løkka, og for hvert tall, legge det til variabelen din. Husk at du kan legge noe til en varibel med `+=`.

c) Sammenlign svaret programmet ditt gir med det vi fant for hånd. Er de to like? Hvis de ikke er det så er det noe galt!

d) Hvis programmet ditt fungerte som forventet kan du nå endre sånn at du regner summen av de første 100 tallene

$$1 + 2 + 3 + \ldots + 100.$$

Det er en kjent matematiker, Carl Friedrich Gauss, som fikk denne oppgaven av sin mattelærer når han gikk på skolen på 1700-tallet. Læreren tenkte nok at dette skulle holde Gauss opptatt en god stund med å legge sammen tall etter tall. Gauss hadde ikke tilgang til en datamaskin, så han kunne ikke

automatisere jobben slik vi har gjort, men ha la merke til et mønster i tallene. Gauss la merke til at om vi starter på begge endene av rekka får vi et mønster

$$1 + 100 = 101, 2 + 99 = 101, 3 + 98 = 101, \ldots 50 + 51 = 101.$$

Fra dette mønsteret klarte Gauss å finne en formel for summen av tallene fra 1 til n, og uttrykket hans var

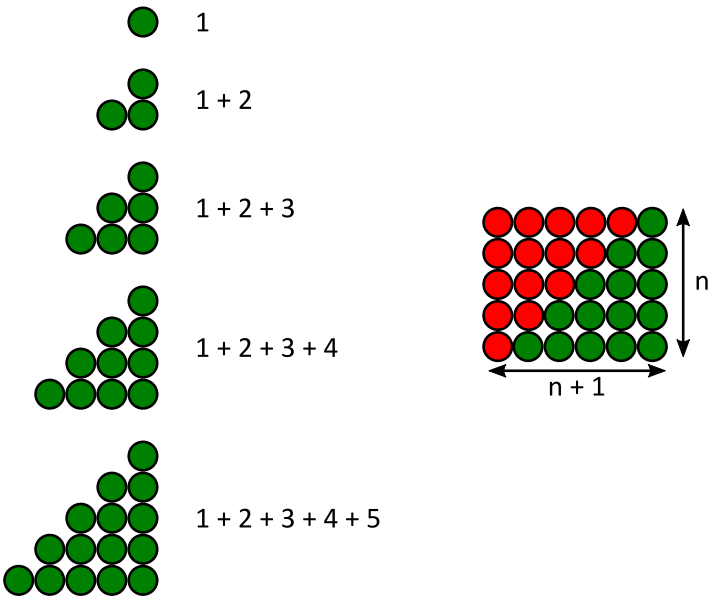
$$T_n = \frac{n(n+1)}{2}.$$

---

e) Bruk formelen til Gauss og sjekk at du får samme svar som programmet ditt for $n = 100$.

f) Gjør det samme for $n = 1000$, så $n = 1000000$ (én million).

For å skjønne hvorfor denne formelen er som den er, så kan det lønne seg å skjønne hvorfor de kalles trekanttall. Om vi tegner opp summene som antall baller, og tenger først 1, så 2, og så 3 bortover, sånn som dette:



Så ser vi at de ulike summene blir trekanter. Vi kan så gjøre om en slik trekant til en firkant ved å legge på like mange nye. Sånn som vist på høyre side av figuren. Denne firkanten har n baller i høyden, og $n+1$ baller bortover. Da er

antall baller i hele firkanten $n(n+1)$. Men vi har jo doblet antall baller for å få en firkant, så om vi bare skal telle de grønne ballene må vi dele på to.

---

g) Hvordan kan kan du sjekke om et tall er et trekanttall? Skriv et program som sjekker om et heltall er et trekanttall, og hvis ja, skriver ut faktorene i dette. For eksempel bør programmet skrive ut `1 + 2 + 3 + 4 + 5` hvis det blir gitt input `15`.

Oppgave 9 *Tverrsum*

Tverrsummen av et tall er tallet du får dersom du plusser alle sifrene i tallet med hverandre.

a) Hvor mange 3-sifrede tall finnes det som har tverrsum 5? Løs for hånd.

b) Skriv et program som finner alle 3-sifrede tall med tverrsum 4. Start med å skrive en løkke for alle tall fra 1 (hvorfor fra 1?) til 9; i denne løkken lager du enda en løkke; og i denne løkken trenger du enda en. I den innerste løkken trenger du en betingelse, og til slutt en `print`.

c) Legg inn en teller i programmet ditt, som først er 0, og deretter øker hver gang du finner et tall med tverrsum 4. Hvor mange tall er det tilsammen? Stemmer det med det du fikk når du regnet for hånd?

d) Tror du det er flest 3-sifrede tall med tverrsum 4, eller flest 4-sifrede tall med tverrsum 3 – eller er det like mange? Utvid programmet ditt til å telle begge deler, og sjekk om du hadde rett.

Oppgave 1 `and` *og* `or`

Skriv om de følgene programlinjene blir `True` eller `False`. Du trenger ikke skriv noe kode i denne oppgaven, men er du usikker på hva svaret er kan du selv prøve å se hva som skjer ved å taste det inn i Python.

```
tall = 5 #setter variabelen tall lik 5

tall == 5
tall == 7
tall > 8
tall > 2
tall >= 5
tall > 5
not True
True != False

True or False
False and True
True and True

tall == 5 or tall == 6
tall > 5 and tall == 1
tall < 10 and tall > 2
tall < 10 or tall > 6

(tall > 4 or tall == 2) and tall > 6
(tall == 5 or tall > 4) or True
(tall == 5 and tall > 4) or (tall < 4  or tall > 1)

not (tall == 5)
tall != 4
not (tall != 10)

(not (tall < 5) and (tall > 1)) or not tall != 5
```