IN1000 Innlevering 5

Frist for innlevering: 11. Mars 2025, kl. 23.59 **Sist endret:** 26.02.2025

Introduksjon

Innleveringen består av 6 oppgaver. Les gjennom hver oppgave før du begynner å programmere, og forsøk gjerne å løse oppgavene på papir først! Hvis du sitter fast på en oppgave bør du prøve å løse øvingsoppgavene i Trix (se lenke under hver oppgave) og se på undervisningsmodulene før du spør om hjelp.

Pass på at oppgavene du leverer ligger i riktig navngitte filer, som vist under oppgavetittelen. For hvert program du skriver skal du legge ved en kommentar i toppen av filen som forklarer hva programmet gjør. Videre forventes det at du kommenterer koden underveis så det blir tydelig hva du har tenkt. Andre viktige krav til innleveringen og beskrivelse av hvordan du leverer finner du nederst i dette dokumentet.

Fristen for innleveringen er satt til å være to uker fra publiseringsdato, fordi innleveringen er noe større enn de andre dere har hatt frem til nå. Lykke til!

Læringsmål

I denne oppgaven skal du vise at du kan løse utfordringer som omhandler data ved hjelp av programmering. Du skal kunne lese inn verdier fra tekstfiler og bruke disse for å gjøre beregninger. Du skal også kunne skrive funksjoner som tar imot, manipulerer og returnerer data.

Oppgave 1: Parametere og returverdier

Filnavn: *funksjoner.py*

1. Skriv et program med en funksjon *adder(tall1, tall2)* som tar to heltall som parametere. Funksjonen skal summere tallene og returnere resultatet. Kall på funksjonen to ganger med argumenter du selv velger, og skriv ut resultatene med en passende tekst.

2. Be så brukeren om å skrive inn en tekststreng, og deretter en bokstav. Programmet skal så, ved å bruke en **loop**, skrive ut hvor mange ganger den oppgitte bokstaven forekommer i den oppgitte tekststrengen. Du skal ikke regne stor og liten bokstav som like (det vil for eksempel si at det er 0 forekomster av "*E"* i ordet *"hei"*).

3. Skriv en funksjon *tell_forekomst(min_tekst, min_bokstav)*. Funksjonen skal telle hvor mange ganger en bokstav *min_bokstav* forekommer i teksten *min_tekst*, og returnere dette tallet. Skriv deretter om kodelinjene fra punkt 2 til å benytte denne funksjonen, men sørg for at programmet fremstår helt likt for brukeren når det kjøres. Lever programmet slik det ser ut etter at du har gjort denne endringen (du trenger altså ikke å levere to versjoner av programmet).

Syntes du denne oppgaven var vanskelig? Se Trix-oppgave 5.17, 5.19, og/eller se gjennom undervisningsmodulene Parametre i prosedyrer og Retur-verdier. Syntes du

denne oppgaven var enkel? Se Trix-oppgave 5.21.

Oppgave 2: Å telle bokstaver og ord

Filnavn: *ordtelling.py*

I denne oppgaven skal dere lage to funksjoner som begge utfører operasjoner på strenger. Så skal dere lage et program som bruker disse funksjonene på input fra brukeren, og skriver ut informasjon i terminalen. Foruten tjenester gjennomgått på forelesning kan du benytte tjenesten **split()** som også tilbys av strengobjekter: om man har en variabel **tekst="hei du"** kan man skrive **ordene = tekst.split()** for å få en liste med hvert ord. Listen vil se sånn her ut: [“hei”, “du”] siden **split()** deler på mellomrom. Du kan skrive tegnsetting som “!”, ”.” og ”,”, etc. med mellomrom, og dere kan også behandle disse tegnene som ord.

1. Lag en funksjon som, gitt et ord, returner antall bokstaver i ordet.

2. Lag en funksjon som, gitt en setning, returnerer ei ordbok hvor hvert (unike) ord i setningen er en nøkkelverdi, og antall ganger ordet finnes i setningen som innholdsverdi. Du skal komme frem til antallet uten å bruke count().

3. Lag et program som tar inn en setning fra brukeren. Bruk så funksjonene fra deloppgave 1 og 2. Skriv også til brukeren hvor mange ord det er i setningen, og bruk så resultatene fra de to funksjonene til å skrive hvor mange ganger hvert unike ord forekommer, og hvor mange bokstaver hvert ord består av.

Eksempel på bruk og utskrift:

```
>>>Skriv en setning:
>>>Jeg liker kake , jeg .
>>>Det er 6 ord i setningen din.
>>>Ordet 'jeg' forekommer 2 ganger, og har 3
bokstaver
>>>Ordet 'liker' forekommer 1 gang, og har 5
bokstaver
>>>Ordet 'kake' forekommer 1 gang, og har 4
bokstaver >>>Ordet ',' forekommer 1 gang, og har 1
bokstav >>>Ordet '.' forekommer 1 gang, og har 1
bokstav
```

Syntes du denne oppgaven var vanskelig? Se Trix-oppgave 3.02, 5.19, og/eller se gjennom undervisningsmodulene Objekter tilbyr tjenester, Parametre i prosedyrer, Retur-verdier, Kombinere løkker og samlinger eller Ordbøker. Syntes du denne oppgaven var enkel? Se Trix-oppgave 5.20, 6.07.

Oppgave 3: Regnefunksjoner

Filnavn: *regnefunksjoner.py*

1. Skriv en funksjon *addisjon* som har to parametre. Funksjonen skal returnere summen av disse. Skriv en kodelinje som kaller på *addisjon* med heltallsargumenter du velger selv, og skriv ut resultatet.

2. Skriv tilsvarende funksjoner for *subtraksjon*, der du trekker den andre parameteren fra den første, og *divisjon*, der du deler den første parameteren på den andre. Du skal teste disse funksjonene også, men denne gangen ved hjelp av *assert*. Skriv minst 3 *assert*-uttrykk for hver funksjon, der du sjekker at kall på funksjonen med forskjellige heltall du velger selv får verdien du forventer (minst ett kall med to positive tall, ett med to negative tall og ett kall med et positivt og et negativt tall).

3. Lag en funksjon *tommer_til_cm* med én parameter *antall_tommer*. Øverst i denne funksjonen skal du bruke *assert* for å sjekke at *antall_tommer* er større enn 0. Deretter skal funksjonen returnere hvor mange centimeter *antall_tommer* tilsvarer. For å regne om fra tommer til centimeter kan du gange *antall_tommer* med 2.54. Test funksjonen din.

4. Lag en prosedyre *skriv_beregninger()*. Den skal ikke ta imot noen argumenter, men i stedet bruke *input* for å hente inn verdier fra bruker til kall på funksjonene du skrev over. Ta høyde for at brukeren kan skrive inn flyttall.
    a. Først skal prosedyren hente inn to tall fra bruker. Disse tallene skal brukes som argumenter i kall på *addisjon*, *subtraksjon* og *divisjon*. Hver av disse funksjonene skal kalles og resultatet skrives ut til terminal.
    b. Deretter skal prosedyren hente inn et nytt tall som skal konverteres fra tommer til centimeter. Også dette resultatet skal skrives ut.

5. Test skriv_beregninger. Utskriften bør se omtrent slik ut:

```
Utregninger:
Skriv inn tall 1: 12
Skriv inn tall 2: 5

Resultat av summering: 17.0
Resultat av subtraksjon: 7.0
Resultat av divisjon: 2.4

Konvertering fra tommer til cm:
Skriv inn et tall: 23
Resultat: 58.42
```

Syntes du denne oppgaven var vanskelig? Se Trix-oppgave 6.03, og/eller se gjennom undervisningsmodulene Parametre i prosedyrer eller Sjekke antagelser med assert. Syntes du denne oppgaven var enkel? Se Trix-oppgave 5.25.

Oppgave 4: UiO-brukere

Filnavn: brukernavn.py

I denne oppgaven skal du lage et program for å legge til nye UiO brukernavn til en samling og fastslå deres e-post. Bonusoppgavene er en ekstra utfordring for de som ønsker det.

1. Lag en funksjon *lag_brukernavn()* som tar inn et fullt navn som en string (f.eks. “Kari Nordman”) og som returnerer et UiO brukernavn. Et brukernavn skal bestå av personens fornavn, etterfulgt av den første bokstaven i deres etternavn, alle bokstavene skal være små bokstaver (f.eks. vil “kari Nordmann” bli “karin”). Du kan anta at alle navn kun består av et fornavn og et etternavn, ingen mellomnavn.
   HINT: funksjonen .split() kan være hjelpsom.

2. Lag en funksjon *lag_epost()* som tar inn et brukernavn og en e-post suffix (begge skal være stringer) som argumenter, og som returnerer en epost for brukeren. E-post adressen skal bruke brukernavnet som prefix og så legge til en @ mellom prefix og suffix (prefix@suffix). F.eks. ville personen med brukernavn “karin” og e.post suffix “student.matnat.uio.no” fått e-post adressen “karin@student.matnat.uio.no”

3. Lag en prosedyre *skriv_ut_eposter()* som tar inn en ordbok som argument. Denne ordboken skal ha UiO-brukernavn som nøkkelverdier og e-post suffix som innholdsverdier. Funksjonen skal iterere gjennom alle elementene i ordboken og kalle *lag_epost()* på hver bruker, og skrive ut resultatet.
   Du kan bruke denne ordboken for å teste {“olan”: “ifi.uio.no”, “karin”: “student.matnat.uio.no”}

4. Lag en tom ordbok, og lag en while-løkke som skal kjøres så lenge brukeren ønsker å fortsette (unngå å bruke while-løkker som alltid evaluerer til True - unngå f. eks. “while True:”). I løkken skal programmet ta i mot en streng fra brukerinput og:
   a. når brukeren skriver inn strenger “i”, skal programmet spørre brukeren om et navn og ett e-post suffix, og lagre dem i to variabler. Deretter skal programmet kalle funksjonen som lager UiO-brukernavn og lagre dette i en tredje variabel. Den nye brukeren skal bli lagt til i ordboken, med brukernavn som nøkkel og e-post suffix som verdi.
   b. når brukeren taster inn “p”, skal programmet kalle på *skriv_ut_eposter()* med ordboken med brukernavn som parameter.
   c. hvis brukeren taster “s” skal vi gå ut av løkken og avslutte programmet.

5. **Bonus:** Skriv tester for funksjonene som er skrevet i deloppgave 1 og 2. Se craftsmanship module “writing tests for functions” (week 4) for flere detaljer.

6. **Bonus (vanskelig):** Enn så lenge kan vi bare legge til nye brukere hvis deres UiO brukernavn ikke allerede eksisterer. Oppdater funksjonen som ble skrevet i deloppgave 1 slik at den lager unike brukernavn. Hvis brukernavnet allerede eksisterer legger vi til enda en bokstav fra etternavnet. F.eks. hvis “Ola Nordmann”, hvis “olan” eksisterer, må det brukernavnet til den nye brukeren bli “olano”, og hvis det allerede eksisterer “olanor” osv. **Hint:** for å vite hvilke brukernavn som allerede eksisterer trenger du å også sende inn ordboken med de eksisterende brukerne som et argument til funksjonen.

Syntes du denne oppgaven var vanskelig? Se Trix-oppgave 4.09, 6.04 og/eller se gjennom undervisningsmodulene Løkker, Ordbøker, Parametre i prosedyrer eller Retur-verdier. Syntes du denne oppgaven var enkel? Se Trix-oppgave 5.10, 6.07.

Oppgave 5: Temperatur

Filnavn: *temperatur.py*

I denne oppgaven skal du lese inn data fra en fil. Oppgaven kan løses i både Linux, MacOS og Windows, men vær klar over at det er noen forskjeller i hvordan filstier brukes. Legg for sikkerhets skyld datafilene i samme mappe som *temperatur.py*. Vi skal bruke værdata som er målt ved målestasjonen på Blindern. I begge filene er kolonnene separerte med komma (kolonnene har komma mellom seg).

- Den første filen heter max_temperatures_per_month.csv. Dette er en fil med to kolonner: Den første kolonnen er navnet på måneden, den andre er den varmeste (maksimale) temperaturen som ble målt i denne måneden (mellom 1750 og 2017).
- Den andre filen, max_daily_temperature_2018.csv, inneholder den varmeste (maksimale) temperaturen målt per dag i 2018. Her representerer den første kolonnen måneden, den andre dagen i måneden og den tredje den høyeste temperaturen.

1. Lag en funksjon som tar inn en filnavn (string). Funksjonen skal gå gjennom linjene i filen og del dem i kolonner og lage en ordbok der nøkkelen representerer månedene (første kolonne) og verdiene er temperaturene (andre kolonne) på typen float. Til slutt skal ordboken returneres. Kall funksjonen med filen, max_temperatures_per_month.csv, som et argument (parameter). Skriv ut ordboken som funksjonen returnerer.

2. Lag en prosedyre som tar inn to argumenter: En ordbok med de varmeste temperaturene (som laget av funksjonen i steg 1), og et filnavn for en fil som inneholder daglige temperaturer. Denne prosedyren skal lese den daglige temperaturen linje for linje. Hvis temperaturen for en dag er høyere enn temperaturen som finnes i ordboken, skal det skrives ut en beskjed til terminalen. For eksempel: *Ny varmerekord på 30 mai: 31.1 grader celcius (gammel varmerekord var 29.8 grader celcius)*. Kall prosedyren med to argumenter, en ordbok med de varmeste temperaturene per måned, og filen som heter max_daily_temperature_2018.csv.

3. Endre prosedyren som du lagde i steg 2 slik at det blir en funksjon som oppdaterer den høyeste temperaturen i ordboken istedenfor å skrive ut temperaturen. Funksjonen skal returnere den oppdaterte ordboken. Test funksjonen ved å kalle på den, og sjekk at funksjonen gjør det du ser for deg.

4. **Bonus:** Lag en siste prosedyre som tar inn to argumenter: en oppdatert ordbok med de varmeste temperaturene per måned, og et filnavn. Prosedyren skal gå gjennom nøklene og verdiene til ordboken og skrive de til en ny fil (med navnet som ble oppgitt som det andre argumentet) skilt med komma, slik at filen har samme format som max_temperatures_per_month.csv. Test at prosedyrene din fungerer som forventet.

5. **Bonus (vanskelig)**: Lag en prosedyre som rapporterer på varmebølgene i 2018. For denne deloppgaven bruker vi denne forenklede definisjonen av en varmebølge: En periode hvor det minst har vært 5 dager på rad som har vært

over 25 grader celcius. Prosedyrene skal ta inn som input filen max_daily_temperature_2018.csv og lese den linje for linje. Print starten og sluttdatoen av hver varmebølge.

Syntes du denne oppgaven var vanskelig? Se Trix-oppgave 5.01, 5.05, 5.06 og/eller se gjennom undervisningsmodulene Parametre i prosedyrer, Retur-verdier, Ordbøker eller Lese fra og skrive til filer. Syntes du denne oppgaven var enkel? Se Trix-oppgave 5.11, 5.15.

Oppgave 6: Egen oppgave

1. Skriv oppgavetekst til en oppgave som handler om innlesing fra fil og funksjoner. Eller du kan følge dette forslaget: Skriv et beregningsprogram for skreddere med en funksjon som leser inn en fil (som du lager selv og leverer sammen med de andre filene) der hver linje beskriver et navn på et mål og selve målet i tommer. Formatet vil se slik ut:

   ```
   Skulderbredde 4
   Halsvidde 3.2
   Livvidde 10
   ```

   Hint: du kan bruke funksjonen *.split()* for å gjøre dette.

   La programmet legge disse målene i en ordbok med navn på målet som nøkkelverdi og returner ordboken. Lag deretter en prosedyre som tar imot en liste av mål og benytter seg av funksjonen *tommer_til_cm* som du skrev tidligere for å skrive ut målene i centimeter.

2. Løs oppgaven! Du skal levere både oppgaveteksten og besvarelsen (skriv oppgaveteksten som kommentarer over løsningen din).

Krav til innlevering

- Oppgaven må kunne kjøres på din maskin. Test dette før du leverer!
- Kun .py-filene og .txt-filen fra oppgave 6 skal leveres inn.
- Spørsmålene til innleveringen skal besvares i kommentarfeltet.
- Koden skal inneholde gode kommentarer som forklarer hva programmet gjør og kommentarer som forklarer hvordan du har tenkt underveis.
- Programmet skal ha gode utskriftssetninger som gjør det enkelt for bruker å forstå.

Hvordan levere oppgaven

1. Svar på spørsmålene under i kommentarfeltet i Devilry. De **skal** besvares.

   a. Hvordan synes du innleveringen var? Hva var enkelt og hva var vanskelig?
   b. Hvor lang tid (ca.) brukte du på innleveringen?
      Var det noen oppgaver du ikke fikk til? Hvis ja:
      i. Hvilke(n) oppave(r) er det som ikke fungerer i innleveringen?
      ii. Hvorfor tror du at oppgaven ikke fungerer?
      iii. Hva ville du gjort for å få oppgaven til å fungere hvis du hadde mer tid?

2. Logg inn på Devilry.

3. Lever alle *.py*-filene, og husk å svare på spørsmålene i kommentarfeltet.

4. Husk å trykke på *Add delivery or question*. Sjekk deretter at innleveringen din er komplett. Du kan levere så mange ganger du vil frem til fristen. Det er ikke mulig å slette innleveringer i Devilry. *Alle* filer må være med hver gang du leverer, og filene må ha samme navn som oppgitt i oppgaveteksten. Retteren din vil gi tilbakemelding på kun den siste innleveringen, og derfor er det viktig at alle filer er med i siste innlevering.

5. Innleveringen gir deg mulighet for en konstruktiv tilbakemelding på hva du har fått til, i tillegg til forbedringer/misforståelser og om du er omtrent i rute med faget. De fleste vil ha behov for å løse mange flere oppgaver enn de du finner i denne innleveringen. Det får du anledning til i gruppetimene og i Trix. Du finner Trix-oppgaver for denne uken her. Læreboken inneholder også mange oppgaver (noen går litt utenfor vårt pensum). Du finner dessuten mye på nett - og ikke minst kan du finne på egne programmer og utvidelser til oppgaver.