IN1000: Obligatorisk innlevering 5

*Levering av denne oppgaven **kan erstattes med** å møte opp til én dobbeltime på laben i uke 9 eller 10. **For å få godkjent må du jobbe med obligen sammen med minst én annen** i 2 x 45 minutter på en av de tre gruppene under veiledning fra en hjelpelærer. Dere definerer dere som ei gruppe i Devilry og levere en kort rapport som kommentar for arbeidet som er gjort. Ta kontakt med en av hjelpelærerne på laben/gruppa som vil hjelpe deg/dere.*

For å få umiddelbar tilbakemelding: møt opp på laben (ei av gruppene) og be om samretting.

Les gjennom hver oppgave før du begynner å programmere, og forsøk gjerne å løse oppgavene på papir først (da kan det være lettere å konsentrere seg om en fremgangsmåte som gradvis beskrives i Python).

For hvert program du skriver, skal du legge inn 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. Dersom oppgaven sier du skal endre programmet, trenger du kun å levere den endrede løsningen. Andre viktige krav til innleveringen og beskrivelse av hvordan du leverer finner du nederst i dette dokumentet.

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: Parametre 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 må 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. Før du avslutter skriver du resultatene ut på terminal.
   Lever programmet slik det ser ut etter at du har gjort alle endringer (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*

Du skal lage et program som analyserer en tekst oppgitt av brukeren. Programmet skal bruke to egenskrevne funksjoner for å hente ut informasjon fra teksten. Du må definere og bruke disse funksjonene slik at programmet til slutt presenterer en oversikt over innholdet i teksten. Foruten tjenester gjennomgått på forelesning kan du benytte tjenesten **split()** som 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 slik ut: [“hei”, “du”] siden **split()** deler på mellomrom. Om **tekst="hei, du"** vil listen vil se slik ut: [“hei,”, “du”] fordi mellomrommet kommer etter ‘,’.

1. Lag en funksjon som, gitt et ord, returner antall bokstaver i ordet.
2. Lag en funksjon som tar imot en hel setning og undersøker hvilke ord som forekommer, og hvor ofte hvert av dem forekommer. Du skal komme frem til antallet uten å bruke count().
3. Lag et program som skal bruke funksjonene fra deloppgave 1 og 2:
   Les inn en setning fra brukeren. Skriv deretter ut til brukeren hvor mange ord det er i setningen, hvor mange ganger hvert unike ord forekommer, og hvor mange bokstaver hvert ord består av.

Eksempel på bruk og utskrift (etter punkt 4):

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

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: Temperatur

Filnavn: *temperatur.py* ***(levér også datafilene programmet leser fra)***

I denne oppgaven skal du lage et program til å undersøke om det ble satt nye varmerekorder i 2018, og å identifisere perioder med varmebølger. Du skal lese inn data fra fil. Oppgaven kan løses i både Linux, MacOS og Windows, men vær klar over at det er noen forskjeller i hvordan filstier (path) 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 .

- 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 først en funksjon som tar inn et filnavn og samler informasjon fra filen i en egnet type samling (for eksempel ordbok) som gjør det enkelt å slå opp temperaturer per måned. Filen skal ha samme format som i max_temperatures_per_month.csv. Funksjonen din skal returnere samlingen. Kall nå funksjonen med filen, *max_temperatures_per_month.csv*, som argument. Skriv ut samlingen som funksjonen returnerer på terminalen.

2. Programmet ditt skal nå sammenligne de daglige temperaturene i 2018 med de tidligere rekordene. Hvis du finner en dag i 2018 med høyere temperaturer enn den gamle månedsrekorden skal programmet fortelle dette:
   Lag en prosedyre som tar inn samlingen fra funksjonen i punkt 1, og et filnavn på en fil som inneholder daglige temperaturer. Prosedyren skal skrive ut de nye varmerekordene. Kall prosedyren med datastrukturen inneholdende de varmeste temperaturene per måned, og filen som heter *max_daily_temperature_2018.csv.*

3. Endre prosedyren du laget i punkt 2 slik at det blir en funksjon som oppdaterer den høyeste temperaturen i samlingen i stedet for å skrive ut temperaturen. Funksjonen skal returnere den oppdaterte samlingen. Test funksjonen ved å kalle på den og skrive ut resultatet, og sjekk at funksjonen gjør det du planla.
   a
4. Lagre nå de oppdaterte rekordene til en ny fil:
   Lag en funksjon som skriver den nye samlingen til en CSV-fil med samme format som den originale *max_temperatures_per_month.csv*. Test at koden din fungerer som forventet.

5. Bonus (ikke obligatorisk): 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 dem 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.

6. Bonus (ikke obligatorisk): Lag en prosedyre som rapporterer på varmebølger 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 celsius. Prosedyrene skal ta inn filen *max_daily_temperature_2018.csv* og gå gjennom 2018-dataene for å finne alle varmebølger. 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 4: Regnefunksjoner

Filnavn: *regnefunksjoner.py*

Du skal lage en liten beregningsverktøykasse, som utfører enkle matematiske beregninger for brukeren. Du bestemmer selv hvordan du vil organisere funksjonene dine og hvordan du sikrer at de gir riktig resultat. Du må dog ta hensyn til at brukeren kan gjøre feil! Derfor må funksjonene testes.

1. Start verktøyskassen din ved å lage tre regnefunksjoner: addisjon, subtraksjon og divisjon. Hver funksjon skal ta inn to tall og returnere resultatet.
   Test funksjonene med ulike kombinasjoner av tall. Inkluder positive og negative tall. Hva skjer hvis noen forsøker å dele på 0? Bruk assert for å sjekke funksjonene.
2. Lag enda et verktøy, som tar inn et tall i tommer og konverterer dette til centimeter. Hvordan kan du sjekke om brukeren skriver inn ugyldige verdier (null eller negative tall)? Håndter ugyldig input på en god måte. For å regne om fra tommer til centimeter kan du gange *antall_tommer* med 2.54.
3. 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 i punkt 1 og 2. Ta høyde for at brukeren kan skrive inn flyttall.
   - Først skal prosedyren hente inn to tall fra bruker. Disse tallene skal brukes som argumenter i kall på *regnefunksjonene dine*.
   - Hver av disse funksjonene skal kalles og resultatet skrives ut til terminal.
   - Deretter skal prosedyren hente inn et nytt tall som skal konverteres fra tommer til centimeter. Også dette resultatet skal skrives ut.

   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

   Kan koden din håndtere situasjoner der brukeren skriver inn tekst i stedet for tall?

4. Endre koden din slik at brukeren først blir presentert for verktøyskassens innhold.La deretter brukeren velge beregning og utfør bare denne.

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 5: UiO-brukere

Filnavn: brukernavn.py
Du skal lage et program som kan hjelpe en institusjon som UiO med å registrere nye brukere og produsere e-postadresser for dem. Dette programmet må ta inn navn og e-post suffix (det som kommer etter brukernavnet), lage brukernavn automatisk, og skrive ut e-postadresser for alle registrerte brukere. Dette må gjøres ved å bruke funksjoner.

1. Lag en funksjon *lag_brukernavn()* som tar inn et fullt navn (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.

2. Lag en funksjon *lag_epost()* som tar inn et brukernavn og en e-post suffix 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 med informasjon om brukernavn og e-post suffix for alle brukere. Du må gå gjennom alle brukere og skrive ut fullstendige e-postadresser. Du kan bruke denne ordboken for å teste {"olan": "ifi.uio.no", "karin": "student.matnat.uio.no"}
4. Nå skal du sette det hele sammen i et lite interaktivt program, som skal kjøre så lenge brukeren av programmet ønsker å fortsette! Start med en tom ordbok. La da brukeren av programmet velge mellom:
    - Å tilføje nytt navn og e-post suffix
    - Skrive ut alle e-postadresser
    - 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 "olano" 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 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

    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

- Koden din skal inneholde gode kommentarer som forklarer hva programmene gjør.
- Programmene skal ha gode utskriftsetninger som gjør det enkelt for bruker å forstå.

Hvordan levere i Devilry

Ved **samretting** kan du ta kontakt med en hjelpelærer på laben og trenger ikke gjøre dette, bortsett fra å levere filene i Devilry.

1. Du skal IKKE lage en .zip-fil
2. Logg inn i Devilry.
3. Last opp alle filer som angitt i besvarte oppgaver. Last også opp datafilene (.csv) til prorgammer som leser fra fil.
4. Du **skal** også besvare noen spørsmål i kommentarfeltet i Devilry:
    a. Hvordan syntes du innleveringen var? Hva var enkelt og hva var vanskelig?
    b. Hvor lang tid (ca.) brukte du på innleveringen?
    c. Var det noen oppgaver du ikke fikk til? Hvis ja:
        i. For hvilke(n) oppgave(r) fungerer ikke det innleverte programmet ditt?
        ii. Hvorfor tror du programmet ikke fungerer?
        iii. Hva ville du gjort for å få programmet til å fungere hvis du hadde mer tid?
5. Husk å trykke **«Legg til innlevering eller spørsmål»** og sjekk deretter at innleveringen din er komplett. Du kan levere så mange ganger du vil frem til fristen, men **siste innlevering må inneholde alle filer som skal vurderes**.
6. Innleveringen med tilhørende Trix-oppgaver er minimum av hva du bør ha programmert i løpet av en uke - de fleste vil trenge en del mer trening for å få god kontroll på stoffet. Du finner flere oppgaver for denne uken her.

Finner du feil i oppgaveteksten, send en epost til michael@uio.no.