IN1000 Obligatorisk innlevering 5

Sist endret: 20.02.22

Introduksjon

Innleveringen består av 5 oppgaver, der hver oppgave teller 1 poeng. 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 fila 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 da denne obligen er noe større enn det 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 forstå hvor en gitt variabel er synlig og tilgjengelig (skop) og hvordan verdier kan overføres mellom forskjellige skop.

Oppgave 1: 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 det andre parameteret fra det første, og *divisjon*, der du deler det første parameteret på det 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 du får forventet verdi når du kaller på funksjonen med forskjellige heltall du velger selv. Her må du ha minst ett kall med to positive tall, ett med to negative tall og ett kall med ett positivt og ett negativt tall.

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

4. Lag en prosedyre *skrivBeregninger*. 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 skal skrives ut til terminal.

    b. Deretter skal prosedyren hente inn et nytt tall som skal konverteres fra tommer til centimeter. Dette resultatet skal også skrives ut til terminal.

5. Test *skrivBeregninger*. 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
```

Synes du denne oppgaven var vanskelig? Se Trix-oppgave 4.02, 6.03, og/eller se gjennom undervisningsmodulene Parametre i prosedyrer, Parameter-overføring og variabel-skop eller Sjekke antagelser med assert.
Synes du denne oppgaven var enkel? Se Trix-oppgave 4.16

Oppgave 2: UiO-brukere

I denne oppgaven skal du lage et program for å legge til nye UiO-brukernavn til en samling og fastslå deres e-post. Du trenger ikke løse bonusoppgavene for å få poeng.

1. Lag en funksjon *lagBrukernavn()* 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 Nordman” 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 *lagEpost()* som tar inn et brukernavn og en e-post suffix (begge skal være stringer) som argumenter, og som returnerer en e-post for brukeren.
   E-postadressen som returneres skal ha brukernavnet som prefix, etterfulgt av en @ mellom prefix og suffix (prefix@suffix).

For eksempel vil en person med brukernavn “karin” og e-post-suffix “student.matnat.uio.no” få e-postadressen “karin@student.matnat.uio.no”

3. Lag en prosedyre *skrivUtEposter()* 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 *lagEpost()* på hver bruker, og skrive ut resultatet.

   Du kan bruke følgende ordbok for å teste funksjonen din:
   {“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 taster inn “i”, skal programmet spørre brukeren om et navn og en 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 *skrivUtEpost()* 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 ble laget 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.
   Hvis vi for eksempel forsøker å lage et brukernavn for “Ola Nordmann” og brukernavnet “olan” eksisterer, må brukernavnet til den nye brukeren bli “olano”. Hvis “olano” allerede eksisterer så blir det “olanor” osv.
   HINT: For å vite hvilke brukernavn som allerede eksisterer trenger du også å sende inn ordboken med de eksisterende brukerne som et argument til funksjonen.

Synes 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.
Synes du denne oppgaven var enkel? Se Trix-oppgave 5.10, 6.07

Oppgave 3: Skop

Filnavn: *skop.py*

I denne oppgaven skal du beskrive programflyt.
Følgende kodesnutt og påfølgende beskrivelse er kun et eksempel:

```
def funk():
	y = 2
	y = y * 2
	return y

def hovedprogram():
	x = 5
	z = 4
	x = funk()

hovedprogram()
```

*"Først defineres funksjonen **funk**, som ikke skal ta imot noen parametere. Deretter defineres prosedyren **hovedprogram**, som heller ikke tar noen parametre. Deretter kalles hovedprogram. Inne i hovedprogram opprettes en variabel **x** med verdi 5. Deretter oppretter vi en variabel **z** med verdien 4. Så kaller vi på funk. I funk opprettes en variabel **y** med verdien 2. Så blir y lik seg selv ganger 2. Deretter returneres y, og x får tilordnet denne returverdien, dvs. 4."*

Selve oppgaven: Beskriv med ord hva som vil skje om følgende program kjøres. Forsøk først å løse oppgaven ved å gå gjennom hva du tror vil skje linje for linje. Kopier deretter koden over i en lokal Python-fil eller pythontutor.com og kjør programmet for å se om det oppfører seg slik du trodde.
Du kan lære mer om pythontutor i undervisningsmodulen Kjøre et program i debugger.

```
def minFunksjon():
	for x in range(2):
	    c = 2
	    print(c)
	    c += 1
	    b = 10
	    b += a
	    print(b)
	return(b)

def hovedprogram():
	a = 42
	b = 0
	print(b)
	b = a
	a = minFunksjon()
	print (b)
	print (a)

hovedprogram()
```

Synes du denne oppgaven var vanskelig? Se Trix-oppgave 5.02, 5.03, 5.04, side 282-285 i pensumboka og/eller se gjennom undervisningsmodulen Parameter-overføring og variabel-skop. Synes du denne oppgaven var enkel? Se Trix-oppgave 5.12

Oppgave 4: Temperatur

Filnavn: *temperatur.py*

For å løse 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 sikkerhetsskyld datafilene i samme mappe som temperatur.py.

I denne oppgaven skal vi bruke værdata som er målt ved målestasjonen på Blindern. Vi skal bruke to forskjellige tabulære filer. I begge filene er kolonnene separert med komma i 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 representer den første kolonnen måneden, den andre dagen i måneden og den tredje den høyeste temperaturen.

1. Lag en funksjon som tar inn et filnavn (string). Funksjonen skal gå gjennom linjene i filen og dele dem opp i kolonner, og lage en ordbok der nøkkelen representerer månedene (første kolonne), og verdiene er temperaturene (andre kolonne) av 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 de daglige temperaturen fra filen linje for linje. Hvis temperaturen for en dag er høyere enn temperaturen som finnes i ordboken skal det skrives ut en beskjed til terminalen. Eksempel: *"Ny varmerekord 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 i stedet for å skrive ut temperaturen. Funksjonen skal returnere den oppdaterte ordboken. Test funksjonen ved å kalle på den, sjekk at funksjonen gjør det du hadde forestilt 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 filen som ble oppgitt som det andre argumentet. Nøklene og verdiene skal være 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 minst 5 dager på rad 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 for hver varmebølge.

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

Oppgave 5: 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 *tommerTilCm* 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 skal inneholde kommentarer som forklarer hva programmet gjør.

- Programmet skal inneholde utskriftssetninger som gjør det enkelt for bruker å forstå.

Den obligatoriske innleveringen er minimum av hva du bør ha programmert i løpet av en uke. Du finner flere oppgaver for denne uken her.