IN1000 Obligatorisk innlevering 6

Frist for innlevering: 09.10 kl 12:00

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) 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.

Læringsmål

Målet med denne oppgaven er å gi deg trening i å lage og jobbe med klasser og objekter. I tillegg skal du kunne manipulere informasjon i objektene og forstå nytten av innkapsling av informasjon.

Oppgave 1: Salgsstatistikk

Filnavn: *telefonsalg.py*

I denne oppgaven skal du skrive et program som tar inn data fra fil og skriver ut statistikk for et telefonsalg-firma. For å gjøre dette skal du lese inn tekst fra en fil som er formatert slik at det er to ord på hver linje, adskilt av et mellomrom, der det første ordet er et navn og det andre ordet er et salgstall, slik:

```
Heidi 133
John 97
Kari 48
```

1. Skriv en funksjon *innlesing*, med et parameter *filnavn*. Denne funksjonen skal lese inn en fil ved hjelp av *filnavn* og legge alle linjene i filen inn i en ordbok, der den ansattes navn blir nøkkelverdien og antallet salg blir innholdsverdien (husk å konvertere innholdsverdien til et heltall). Funksjonen skal deretter returnere ordboken.

   Hint: Du kan bruke string-funksjonen *split* med " " (mellomrom) som parameter for å dele opp hver linje.

2. Lag en prosedyre *maanedensSalgsperson* som tar imot en ordbok, går gjennom denne og skriver ut navn og antall salg for den personen som solgte mest den måneden.

3. Skriv en funksjon *totaltAntallSalg* som tar imot en ordbok og returnerer summen av verdiene i ordboken.

4. Lag en funksjon *gjennomsnittSalg* som tar imot en ordbok og returnerer gjennomsnittet av verdiene dens. **Hint:** Funksjonen *totaltAntallSalg* gir deg allerede halvparten av løsningen her!

5. Til slutt skal du skrive en prosedyre *hovedprogram()*. Inne i *hovedprogram* skal du skrive ut den faktiske statistikken. Bruk funksjonene og prosedyrene du har laget, og skriv i tillegg ut antall selgere som ble lest inn. Filen du skal bruke for å teste programmet ditt er *salgstall.txt* (legg gjerne ved denne når du leverer oppgaven).

6. Kall på *hovedprogram()* for å teste programmet ditt. Her er et eksempel på hvordan utskriften kan se ut:

```
Maanedens ansatte er Tina med 178 salg.

Aktive selgere denne maaneden: 6
Totalt antall salg: 772
Gjennomsnittlig antall salg per salgsperson: 128.67
```

Synes du denne oppgaven var vanskelig? Repeter Trix-oppgave 6.01 og løs 7.06.
Synes du denne oppgaven var enkel? Se Trix-oppgave 7.01.

Oppgave 2: Motorsykkel

Filnavn: *motorsykkel.py* og *testMotorsykkel.py*

Du skal skrive en klasse *Motorsykkel* som skal modellere kjøringen av biler. En motorsykkel har et merke, et registreringsnummer og en kilometerstand som viser hvor langt den har kjørt.

1. Skriv klassen *Motorsykkel* med en konstruktør (init-metode) som initierer de instansvariablene klassen trenger.

2. Skriv en metode *kjor(self, km)* som øker kilometerstanden.

3. Skriv en metode *hentKilometerstand(self)* som returnerer motorsykkelens totale kilometerstand.

4. Skriv en metode *skrivUt(self)* som skriver ut merke, registreringsnummer og kilometerstand.

5. I denne deloppgaven skal du teste programmet ditt. Opprett en ny fil *testMotorsykkel.py*. Øverst i filen skal du importere klassen *Motorsykkel*. Deretter skal du skrive en prosedyre *hovedprogram()*, slik som i oppgave 1.

6. Inne i *hovedprogram* skal du opprette et objekt av klassen *Motorsykkel* med et merke, et registreringsnummer og en kilometerstand. Opprett to objekter til. Kall deretter metoden *skrivUt* på alle objektene du har laget.

7. Øk kilometerstanden på den motorsykkelen du laget sist med 10 km, og sjekk at kilometerstanden ble oppdatert ved å skrive ut resultatet av *hentKilometerstand*.

8. Husk å kalle på *hovedprogram* for å teste programmet.

Synes du denne oppgaven var vanskelig? Se Trix-oppgave 7.02.
Synes du denne oppgaven var enkel? Se Trix-oppgave 7.04.

Oppgave 3: Teorioppgave

Filnavn: *teori.py*

Gi korte svar på spørsmålene under:

1. Hva er innkapsling? Hvorfor er det nyttig?

2. Hva er grensesnittet til en klasse? Hvordan skiller det seg fra implementasjonen av en klasse?

3. Hva er en instansmetode, og hvordan skiller dette seg fra prosedyrene/funksjonene vi har møtt hittil?

Oppgave 4: Hund

Filnavn: *hund.py* og *testHund.py*

1. Skriv en klasse *Hund* med en konstruktør som tar imot og setter instansvariabler for *alder* og *vekt*. *Hund* skal i tillegg ha en instansvariabel *metthet* som får verdien 10.

2. Skriv metoder som lar brukere hente ut *alder* og *vekt*.

3. Skriv en metode *spring*. Den tar ingen argumenter. Metoden skal minske *metthet* med 1. Utvid metoden med en sjekk som minsker *vekt* med 1 dersom *metthet* er mindre enn 5.

4. Skriv en metode *spis*. Den tar et heltall og legger det til *metthet*. Skriv deretter en sjekk som setter opp *vekt* med 1 dersom metthet er større enn 7.

5. Skriv en *hovedprogram*-prosedyre i *testHund.py*, der du oppretter et hundeobjekt. Test objektet ved å kalle på *spring* og *spis* minst 2 ganger hver, og skriv ut hundens *vekt* hver gang. Husk å kalle på *hovedprogram* i filen din.

Synes du denne oppgaven var enkel? Se Trix-oppgave 7.05.

Oppgave 5: Egen oppgave

1. Skriv oppgavetekst til en oppgave som handler om klasser og objekter! Eller du kan prøve følgende forslag:

   Skriv en klasse Person med en konstruktør som tar imot navn og alder. I tillegg skal konstruktøren ha en tom liste *hobbyer*. Skriv en metode *leggTilHobby* som tar imot en tekststreng og legger den til i *hobbyer*-listen. Skriv også en metode *skrivHobbyer*. Denne metoden skal skrive alle hobbyene etter hverandre på en linje. Gi deretter Person-klassen en metode *skrivUt* som i tillegg til å skrive ut navn og alder kaller på metoden *skrivHobbyer*. La brukeren skrive inn navn og alder, og lag et Person-objekt med informasjonen du får. Deretter skal brukeren ved hjelp av en løkke få legge til så mange hobbyer de vil. Når brukeren ikke lenger ønsker å oppgi hobbyer skal statistikk om brukeren skrives ut.

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 IFI sine maskiner. Test dette før du leverer!
- Kun *.py*-filene og *README.txt* skal leveres inn.
- Koden skal inneholde gode kommentarer som forklarer hva programmet gjør.
- Programmet skal inneholde gode utskriftssetninger som gjør det enkelt for bruker å forstå.

Hvordan levere oppgaven

1. Lag en fil som heter README.txt. Følgende spørsmål **skal** være besvart i filen:
   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 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 samt *README.txt* og *salgstall.txt* i samme innlevering.

4. Husk å trykke lever og sjekk deretter at innleveringen din er komplett.

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